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

Ricostruzione 3D delle traiettorie veicolari da immagini di una o più telecamere

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

Relatore: Ing. Matteo Matteucci
Correlatore: Ing. Davide Rizzi

Tesi di Laurea Specialistica di:
Andrea Romanoni, matricola 749976

Anno Accademico 2010-2011

A mio padre che...
...beh ...a mio padre

Sommario

Nell'ultimo decennio si sta diffondendo sempre più lo studio di tecniche per elaborare immagini e video. La disciplina che si occupa di quest'area di ricerca è la computer vision. Uno dei problemi a cui viene dedicato grande interesse è il tracking dei veicoli ripresi da una telecamera. Lo scopo di questa tesi è quello di fornire una stima 3D della traiettoria dei veicoli, a partire da una traiettoria sul piano immagine nota in precedenza. Le soluzioni proposte sono due algoritmi di tracking 3D model-based, che eseguono smoothing con un approccio Montecarlo: un algoritmo è basato sull'algoritmo di Viterbi e uno sul particle filter. Abbiamo testato entrambi gli algoritmi sia su dati simulati sia su dati reali: tra i due quello basato sul particle filter offre prestazioni migliori e una maggior flessibilità per sviluppi futuri.

Ringraziamenti

Questa tesi è il compimento di cinque anni intensi di studi. Voglio ringraziare tutte le persone che mi sono state vicine. Ringrazio quindi Sara, che ha sempre creduto in me, supportandomi e sopportandomi e con cui sono cresciuto e continuo a crescere, ringrazio mia madre che mi ha sostenuto per arrivare fino a questo punto e ha appoggiato le mie scelte. Un grazie a Gianni e ai miei famigliari che mi hanno accompagnato nei passi importanti della mia vita. Ringrazio i miei amici con cui ho condiviso le gioie e i dolori universitari e non.

Non posso dimenticare di ringraziare il prof. Matteucci che con il suo entusiasmo mi a trasmesso l'interesse per i temi affrontati, e Davide che con la sua disponibilità mi ha aiutato a superare numerosi ostacoli.

Indice

Capitolo 1

Introduzione

Negli ultimi decenni la ricerca ingegneristica ha rivolto sempre maggior attenzione alle tecniche automatiche di elaborazione delle immagini. Questo interesse è in costante aumento grazie al basso costo di telecamere e fotocamere, e all'elevata flessibilità e maturità raggiunta dagli algoritmi esistenti. La disciplina che utilizza questi metodi al fine di estrapolare informazioni sui soggetti di una scena è la *computer vision*, o *visione artificiale*.

1.1 Stima della traiettoria dalle immagini

Uno dei problemi classici della visione artificiale, noto come *tracking*, è la stima delle traiettorie degli oggetti in movimento in una scena ripresa da una o più telecamere. Questo compito può essere utile a diversi scopi: riconoscere la presenza di persone per sistemi videosorveglianza, localizzare oggetti nell'ambiente per robot autonomi, scoprire l'interazione tra i veicoli in un contesto urbano per monitorare il traffico o per rilevare incidenti.

In letteratura, il tracking viene modellato principalmente come un problema di stima bayesiana, in cui la posizione ed eventualmente l'orientamento degli oggetti osservati rappresentano lo stato del sistema. L'obiettivo del tracking diventa quindi trovare la sequenza di stati per cui sia massima la probabilità date le misure degli oggetti, ovvero la probabilità a posteriori. Il calcolo della probabilità a posteriori coinvolge la stima di due termini: la verosimiglianza, che indica quanto è probabile la misura all'istante corrente noto lo stato stimato per il medesimo istante, e il termine a priori, che descrive il legame tra due stati in istanti successivi, e modella quindi il moto dell'oggetto.

Un problema di stima bayesiana può essere affrontato con metodologie differenti: in genere, si esegue ciò che prende il nome di filtraggio, ovvero per ogni istante, viene stimato lo stato a partire dalle stime precedenti e note le misure fino all'istante corrente. Per questo scopo è largamente utilizzato il filtro di Kalman lineare e la sua versione estesa al caso non lineare. Oltre

al filtraggio, un altro metodo per ottenere una stima bayesiana va sotto il nome di smoothing. A differenza del filtraggio, lo smoothing tiene conto degli stati passati e futuri per ogni istante in cui esegue la stima, permettendo una stima più robusta.

Indipendentemente dall'oggetto tracciato i sistemi di tracking si dividono in due grandi categorie: sistemi 2D che eseguono tracking sull'immagine ripresa dalla telecamera e sistemi 3D che stimano lo stato dell'oggetto nella scena. I primi sono i più semplici da implementare, e in questi casi, la traiettoria stimata è la successione di posizioni del baricentro dell'oggetto nel tempo. Non conoscendo la dimensione e l'orientamento reale dell'oggetto e come si proietta sull'immagine, la stima fornita è un'approssimazione a volte grossolana del baricentro reale. Un sistema di tracking 3D a fronte di una maggiore complessità di implementazione, offre una stima della posizione e dell'orientamento dell'oggetto nel mondo, quindi i risultati sono generalmente più precisi del caso 2D. La maggior parte degli algoritmi di tracking 3D esistenti utilizza un modello dell'oggetto per confrontare i valori dello stato dell'oggetto stimati con le misure 2D estratte dall'immagine. Questi metodi prendono il nome di model-based.

Nel nostro caso per esempio questo modello è un semplice parallelepipedo le cui dimensioni sono variabili, il che rende possibile inoltre una grezza classificazione del veicolo distinguendo tra automobili, camion e motocicli.

1.2 Scopo della tesi

In questa tesi proponiamo due sistemi model-based che effettuano la stima della traiettoria 3D di veicoli che percorrono un'intersezione rotatoria ripresi da una o più telecamere. Rispetto ai sistemi esistenti di tracking 3D di veicoli che utilizzano usualmente il filtro di Kalman per la stima dello stato per ogni istante, gli algoritmi sviluppati eseguono smoothing con un approccio basato sull'algoritmo di Viterbi e uno basato sul particle filter. La novità del nostro lavoro è l'uso dello smoothing, non utilizzato solitamente nel tracking di veicoli. Inoltre, un altro elemento di originalità nel nostro lavoro è l'utilizzo di tecniche di stima Montecarlo per stimare la distribuzione delle posizioni e degli orientamenti dei veicoli sul piano della rotonda. Queste tecniche non sono ancora molto diffuse per il tracking di veicoli: lo strumento maggiormente utilizzato è il filtro di Kalman. Nel caso in esame non riteniamo sufficiente questo strumento per due motivi. La condizione per cui il sistema stimato deve essere lineare è un'approssimazione troppo semplice per poter modellare il moto di un veicolo in modo adeguato. Inoltre, il metodo Montecarlo ci permette di gestire il confronto tra stime e misure dei veicoli in modo più naturale rispetto al filtro di Kalman.

Nei nostri algoritmi non processiamo direttamente i video della rotonda come accade solitamente in letteratura, ma ricostruiamo la traiettoria 3D

a partire dalla traiettoria 2D sul piano immagine estratta da un algoritmo sviluppato in precedenza a questa tesi [48, 45]. Quest'ultimo algoritmo gestisce alcune problematiche tipiche del tracking come la data association e la gestione delle occlusioni. La data association ha il compito di riconoscere quali misure appartengono a quale veicolo; quando viene utilizzato un filtro di Kalman per stimare la traiettoria di un veicolo, la data association permette di associare la misura corrette al filtro corretto. Per occlusione si intende invece quel fenomeno che accade quando, vedendo una scena, un veicolo viene coperto da un oggetto o da un altro veicolo per un certo periodo di tempo. In questi casi, la misura del veicolo risulta disturbata, o nei casi peggiori assente. L'utilizzo delle traiettorie 2D ci permette di non preoccuparci di queste due problematiche, e di indirizzare la nostra attenzione soprattutto al problema di ricostruzione 3D. Vedremo, comunque, come in futuro questi aspetti potrebbero essere trattati nell'ottica della stima nello spazio 3D per ottenere risultati migliori.

Lo scopo della tesi è dunque quello di migliorare le stime 2D esistenti e ottenerne una stima precisa della traiettoria nelle coordinate del mondo reale.

Dei due sistemi proposti, il primo semplifica e sviluppa l'algoritmo proposto in [58] tenendo conto dei nostri differenti dati in input e aumentandone la complessità del modello del veicolo. Il sistema, per ogni singolo frame e ogni veicolo, parte dalla stima del centro 2D per estrarre un insieme di campioni della pose del veicolo, ovvero la posizione e l'orientamento nel piano della rotonda. Tra tutti i campioni scegliamo la sequenza migliore, nel senso che massimizza la probabilità a posteriori, utilizzando l'algoritmo di Viterbi, implementato come ricerca di cammino minimo con l'algoritmo di Dijkstra. Il secondo sistema invece è un'implementazione del particle filter che esegue lo smoothing tramite due esecuzioni successive del filtro. La prima insegue la traiettoria del veicolo che avanza dal primo all'ultimo frame. La seconda esecuzione parte dall'ultimo gruppo di particelle stimate, con cui viene inizializzato un filtro che insegue il veicolo all'indietro, ovvero le cui misure regrediscono dall'istante finale all'istante iniziale.

Gli algoritmi sono stati implementati in un primo momento per gestire il caso in cui il veicolo viene osservato da una telecamera. Successivamente abbiamo esteso il sistema per gestire anche il caso con più telecamere che riprendono lo stesso veicolo e che indicheremo come multicamera. In entrambe i casi gli algoritmi utilizzati sono essenzialmente gli stessi quindi nell'esposizione ci riferiremo in particolar modo al sistema a singola telecamera, ma dove ritenuto necessario, abbiamo messo in rilievo le differenze nel caso multicamera.

1.3 Struttura della tesi

La tesi è strutturata nel seguente modo.

- Nel Capitolo 2 presentiamo una panoramica sul problema del tracking, concentrandoci sugli algoritmi che effettuano tracking 3D model-based. Illustriamo le problematiche connesse e i metodi utilizzati in letteratura per affrontarle.
- Nel Capitolo 3 illustriamo invece i motivi per cui abbiamo sviluppato un sistema 3D partendo da un algoritmo 2D, analizzando gli strumenti che abbiamo utilizzato per processare le informazioni 2D esistenti e tradurle in una stima 3D: presentiamo in questa sede gli algoritmi del particle filter, e di Viterbi utilizzati.
- Nel Capitolo 4 presentiamo in modo approfondito i due sistemi illustrando le scelte compiute e le motivazioni che ci hanno spinto a compierle. Ci soffermiamo soprattutto sul descrivere la progettazione e l'implementazione delle funzioni che calcolano il termine a priori e il termine di verosimiglianza.
- Nel Capitolo 5 commentiamo la precisione delle stime delle posizioni e degli orientamenti ottenute dai due algoritmi, considerando un caso di studio simulato e uno reale.
- Il Capitolo 6 completa l'esposizione presentando alcune possibili direzioni di sviluppo futuro.
- In Appendice A illustriamo il progetto logico dei due sistemi.
- In Appendice B elenchiamo i grafici degli errori compiuti nel tracking su dati simulati .
- In Appendice C elenchiamo i grafici degli errori compiuti nel tracking su dati reali non riportati nel Capitolo 5.

Capitolo 2

Il Tracking

Quando si parla di *tracking* ci si riferisce al problema di inseguire uno o più oggetti in movimento ricostruendone quindi la traiettoria. I sistemi di tracking si differenziano per gli strumenti di misura utilizzati: possono essere usati sensori GPS, infrarossi oppure, nel nostro caso, telecamere. Gli oggetti tracciati possono essere di varia natura: manufatti, facce di persone, corpi che si muovono nella scena oppure veicoli in autostrade o in intersezioni rotatorie, come nel caso oggetto di questa tesi. A volte si muovono gli oggetti, altre volte la telecamera, altre volte ancora si è in una situazione in cui si muovono sia gli oggetti che la telecamera. Questa tesi si pone nel primo caso. Solitamente, le tecniche studiate in letteratura vengono pensate per un dominio specifico, ma gli approcci principali per affrontare il problema sono spesso comuni. Data la vastità degli studi al riguardo, in questo capitolo, abbiamo preferito concentrare l'attenzione sulle pubblicazioni riguardanti il tracking 3D dei veicoli tramite telecamera, non tralasciando gli aspetti più generali che coinvolgono tale processo del tracking.

2.1 Il Tracking 3D con telecamera

In primo luogo, bisogna chiarire cosa si intende per *tracking 3D*. I sistemi più semplici di tracking inseguono gli oggetti che si muovono sul piano immagine senza preoccuparsi della loro interazione con il mondo e l'ambiente circostante. In questi casi si parla di *tracking 2D*. Questo è un primo approccio al problema del tracking, semplificato, rispetto al caso 3D, dal fatto che non si vuole ricostruire la traiettoria nel mondo reale, quindi non si deve affrontare il problema di trovare la corrispondenza tra i punti 2D dell'immagine e i punti 3D del mondo. Alcuni problemi che si devono affrontare si ritroveranno poi nel caso 3D: riconoscimento del moto, inizializzazione del sistema, occlusioni. Il lavoro presentato in questa tesi, è l'evoluzione di un sistema che esegue il tracking 2D, presentato in [48] e [45] e, a partire dai dati estratti ricostruisce le posizioni 3D dei veicoli.

Il tracking 3D ha come fine quello di ricavare le informazioni sul movimento dell'oggetto nel mondo tramite la ricostruzione 3D della traiettoria. Per poter affrontare questo problema è conveniente semplificarlo, dato che trovare l'insieme di posizione e orientamento a 6 gradi di libertà, detto *pose*, nello spazio 3D è un problema oneroso da un punto di vista computazionale. Quindi, è usuale aggiungere delle conoscenze ulteriori a ciò che è il semplice video della scena. A questo proposito, nei lavori presentati in questo capitolo si presuppone che la camera sia calibrata (vedi Paragrafo 2.2.1). Inoltre, a seconda del dominio particolare, si possono aggiungere vincoli per semplificare il compito. Nel caso dei veicoli che si muovono in strada il vincolo più utilizzato è il *Ground Plane Constraint* (GPC) [6, 10, 22, 34, 38, 50, 57, 58, 60].

Il ground plane constraint modella un comportamento basilare dei veicoli della scena. Si suppone infatti che essi siano sempre appoggiati sulla strada, e quindi si vincolano le possibilità di movimento semplificando la stima della pose. Un corpo generico nello spazio ha 6 gradi di libertà. Il GPC permette di considerare solo 3 gradi di libertà per ogni veicolo, riducendo notevolmente lo spazio dei parametri da stimare. Ogni veicolo può venire rappresentato con una terna: (x, y) per la posizione sul piano stradale e θ per la rotazione attorno all'asse perpendicolare al piano.

2.2 Strumenti matematici

Prima di analizzare gli approcci principali al tracking 3D, vengono presentati alcuni degli strumenti più utilizzati in questo ambito. La calibrazione della telecamera è necessaria per poter avere una corrispondenza tra mondo 3D e immagine 2D. Gli strumenti di minimizzazione dell'errore sono usati per stimare la pose, o ogni qualvolta è necessario minimizzare una cifra di merito. Infine, viene presentato il problema di tracking come un problema bayesiano. A questo proposito vengono presentati i filtri che rendono possibile la stima delle traiettorie degli oggetti tenendo in considerazioni le caratteristiche del moto e gli errori della misura. Questo tema viene approfondito nel capitolo successivo, e viene riproposto nell'ottica dei sistemi sviluppati.

2.2.1 Geomeria della telecamera

Avendo a che fare con il tracking 3D, quindi con la ricostruzione di cosa succede nel mondo data l'immagine della scena, è utile capire il modello della telecamera utilizzata, e come questa fa corrispondere punti dello spazio con punti dell'immagine. Il modello di telecamera solitamente utilizzato in computer vision è il cosiddetto modello *pin-hole*. La telecamera viene pensata come in Figura 2.1. Se $\mathbf{M}$ è il punto in 3D in coordinate omogenee,

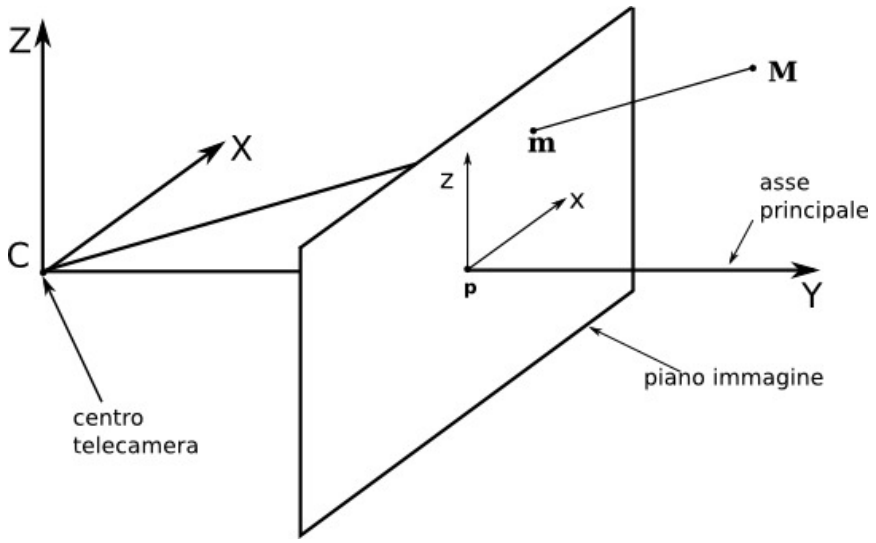


Figura 2.1: Rappresentazione della telecamera.

e $\mathbf{m}$ è il punto proiettato nel piano immagine, si può scrivere:

$$s\mathbf{m} = \mathbf{PM}. \tag{2.1}$$

Ovvero $\mathbf{M}$ viene proiettato sul punto $\mathbf{m}$ tramite la matrice $\mathbf{P}$ a meno di un fattore di scala s. La matrice $\mathbf{P}$, detta di proiezione, è definita a meno del fattore di scala s, come composizione di una matrice $\mathbf{K}$ più una matrice di rototraslazione:

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]. \tag{2.2}$$

La matrice $\mathbf{K}$ è detta matrice di calibrazione: contiene i parametri che dipendono solo dalla telecamere, ovvero i parametri intrinseci. La matrice di rototraslazione contiene i cosiddetti parametri estrinseci, non dipendenti dalla particolare camera, ma dalla posizione e dall'orientamento, ovvero dalla pose, della telecamera nel mondo. Per avere una corrispondenza 2D-3D è quindi necessario conoscere direttamente la matrice $\mathbf{P}$ oppure conoscere sia la matrice di calibrazione che la matrice di rototraslazione.

Per conoscere la matrice $\mathbf{P}$, un algoritmo semplice è il *Direct Linear Transformation* (DLT). Conoscendo un certo numero di corrispondenze 2D-3D a priori, si ottiene un sistema con equazioni del tipo $\mathbf{m} = \mathbf{PM}$. Il sistema può essere riscritto sotto forma di prodotto vettoriale come: $\mathbf{m} \times \mathbf{PM} = \mathbf{0}$. Ora, indichiamo con $\mathbf{p_i^T}$ l'i-esimo vettore riga della matrice $\mathbf{P}$, e definiamo il vettore:

$$\mathbf{p} = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} \tag{2.3}$$

. É possibile raggruppare i fattori di $\mathbf{m} \times \mathbf{PM}$ in una matrice $\mathbf{A}$ e riscrivere $\mathbf{m} \times \mathbf{PM} = \mathbf{Ap} = \mathbf{0}$. Nei casi reali la corrispondenza tra i punti del mondo e quelli dell'immagine non è mai perfetta, quindi questo termine non è mai nullo: la matrice $\mathbf{P}$ si ricostruisce dal vettore $\mathbf{p}$ che minimizzando il prodotto $\mathbf{Ap}$. Un altro modo per trovare la matrice di proiezione $\mathbf{P}$ consiste nel minimizzare l'errore geometrico della riproiezione, o retroproiezione, dei punti 3D sul piano immagine, e dei punti 2D misurati. In altre parole, se $\mathbf{m_i}$ è la misura di un punto di cui conosco la corrispondenza $\mathbf{M_i}$ in 3D, trovo $\mathbf{P}$ come:

$$\mathbf{P} = \arg\min_{\mathbf{P}} \sum_i dist^2(\mathbf{PM_i}, \mathbf{m_i}). \tag{2.4}$$

In molti casi però è utile avere le informazioni sui parametri estrinseci ed intrinseci separate. Questo perché solitamente quando si esegue tracking 3D, i parametri intrinseci che caratterizzano la telecamera rimangono fissi. Può capitare, invece, che ci siano dei movimenti della telecamera e quindi che varino i soli parametri estrinseci, in questi casi è meglio stimare indipendentemente le due matrici.

Quando si parla di una camera calibrata, usualmente significa che sono stati stimati in qualche modo i parametri intrinseci e quindi si conosce la matrice $\mathbf{K}$ che compone $\mathbf{P}$. Per la spiegazione delle tecniche utilizzate, che sono ormai quasi standardizzate, si rimanda a [26], che presenta anche la geometria necessaria per comprendere i metodi sviluppati.

Per stimare la matrice di rototraslazione non esistono invece algoritmi standard, ma a seconda del caso si può utilizzare l'algoritmo più consono. In alcuni casi può essere utilizzata la stessa quantità in (2.4) minimizzata rispetto alla matrice di rototraslazione.

In seguito, quando parleremo di calibrazione, intendendo, per semplicità che, delle camere utilizzate si conosce sia la matrice $\mathbf{K}$ che quella relativa alla pose della camera.

2.2.2 Minimizzazione dei minimi quadrati

Nei sistemi di tracking 3D accade spesso di avere a che fare con problemi di minimizzazione di errori. Ne è un esempio proprio l'equazione (2.4). La distanza tra il punto proiettato e la misura non è nient'altro che l'errore, o *residuo* della stima della proiezione di un punto 3D sul piano immagine rispetto alla misura dello stesso punto sul piano immagine. Minimizzare la somma dei quadrati degli errori significa fare la cosiddetta minimizzazione dei minimi quadrati, nota come *Least-Square Minimization*.

Si supponga di avere un sistema di cui si conoscono le misure $\mathbf{x} \in \mathbf{R}^n$. Chiamiamo $f(\mathbf{p})$, la funzione che definisce il modello del sistema e $\mathbf{p} \in \mathbf{R}^m$ il vettore di parametri che caratterizza la funzione. Chiamiamo il vettore delle stime date dal modello $\hat{\mathbf{x}} \in \mathbf{R}^n$, con $\hat{\mathbf{x}} = f(\mathbf{p})$. Nel problema dei minimi

quadrati viene ricercato il vettore di parametri $\mathbf{p}$ che minimizza l'errore di stima della funzione f rispetto alle misure date.

Indicando con x_i l'i-esima misura e con $\hat{x}_i = f(\mathbf{p})$ la stima relativa alla misura, l'i-esimo residuo è:

$$r_i = x_i - \hat{x}_i \tag{2.5}$$

e il problema di minimizzazione dei minimi quadrati viene descritto con l'equazione:

$$\mathbf{p} = \arg\min_{\mathbf{p}} \sum_i r_i^2. \tag{2.6}$$

Esprimendo l'operazione di norma con la notazione $\|.\|$Questa equazione può essere riscritta in forma vettoriale come:

$$\mathbf{p} = \arg\min_{\mathbf{p}} \|f(\mathbf{p}) - \mathbf{x}\|^2 \tag{2.7}$$

. Quando la funzione f è lineare il problema di minimizzazione diventa il problema di risolvere un sistema lineare del tipo $\mathbf{Ap} = \mathbf{x}$. La soluzione di questo è semplicemente $\mathbf{p} = \mathbf{A}^+\mathbf{x}$ con:

$$\mathbf{A}^+ = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}. \tag{2.8}$$

Nella maggioranza dei casi di interesse della computer vision la funzione f non è lineare. A questo proposito è utile introdurre tre dei metodi più utilizzati per risolvere il problema di minimizzazione dei minimi quadrati: la *discesa del gradiente*, il metodo *Gauss-Newton* e il metodo *Levenberg-Marquardt*.

I tre metodi sono iterativi e partono da una soluzione $\mathbf{p}_0$ per arrivare alla stima del punto di minimo di una funzione f. Ad ogni iterazione k-esima la stima dei parametri che minimizzano la somma dei minimi quadrati viene aggiornata in questo modo:

$$\mathbf{p}_k = \mathbf{p}_{k-1} + \Delta_{k-1}. \tag{2.9}$$

I tre metodi si differenziano per il modo con cui viene calcolato Δ_{k-1}

Con il metodo della discesa del gradiente viene fissato un valore γ. All'iterazione k-esima viene calcolato il valore del gradiente nel punto trovato all'iterazione precedente. Il gradiente calcolato indica il verso in cui la funzione cresce. Percorrendo la funzione nel verso opposto, di una quantità proporzionale all'intensità del gradiente si trova la nuova soluzione. In sintesi, in riferimento a (2.9) si ha:

$$\Delta_{k-1} = -\gamma \nabla f(\mathbf{p}_{k-1}). \tag{2.10}$$

In Figura 2.2 viene data un'idea grafica. L'algoritmo termina quando si raggiunge un punto fisso, ovvero da cui non ci si sposta per iterazioni successive.

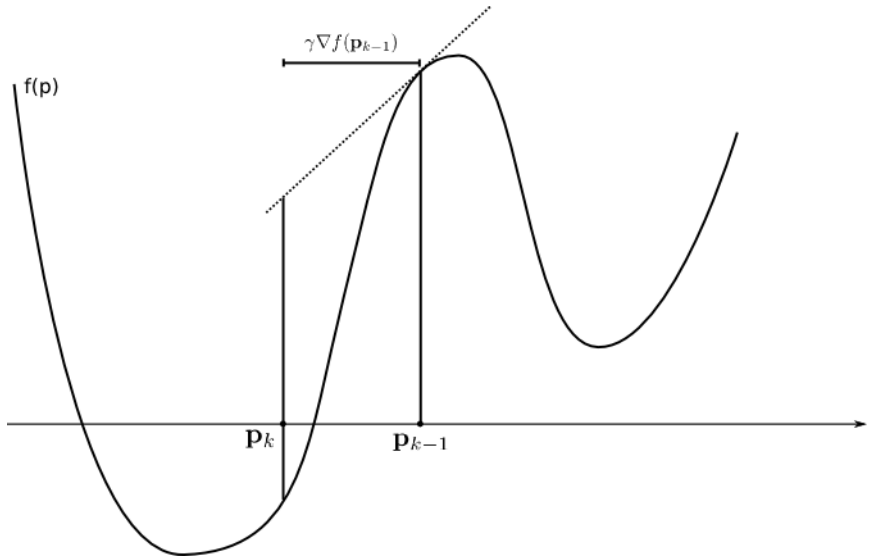


Figura 2.2: Visualizzazione grafica del metodo della discesa del gradiente.

Per il metodo Gauss-Newton, indicando con $\mathbf{J} = \frac{\partial f}{\partial p}$ lo Jacobiano e approssimando la funzione f in serie di Taylor al primo ordine, si deriva il valore di Δ_{k-1} in questo modo:

$$\begin{aligned} \Delta_{k-1} &= \arg\min_{\Delta} \|f(\mathbf{p}_{k-1} + \Delta_{k-1}) - \mathbf{x}\| \\ &= \arg\min_{\Delta} \|f(\mathbf{p}_{k-1}) + \mathbf{J}\Delta_{k-1} - \mathbf{x}\|. \\ &= \arg\min_{\Delta} \|f(\mathbf{r}_{k-1}) + \mathbf{J}\Delta_{k-1}\| \end{aligned} \tag{2.11}$$

L'ultima forma del termine da minimizzare è lineare nella variabile Δ quindi la soluzione è:

$$\Delta_{k-1} = -(\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\mathbf{r}. \tag{2.12}$$

Il metodo Levenberg-Marquardt combina i metodi di discesa del gradiente e Gauss-Newton. Fissando un valore λ variabile a ogni iterazione, il valore che permette di aggiornare Δ viene definito come:

$$\Delta_{k-1} = -(\mathbf{J}^T\mathbf{J} - \lambda\,\mathrm{diag}(\mathbf{J}^T\mathbf{J}))^{-1}\mathbf{J}^T\mathbf{r}. \tag{2.13}$$

Il fattore λ viene detto fattore di *damping*. Quando è molto elevato, rende il termine $(\mathbf{J}^T\mathbf{J} - \lambda\,\mathrm{diag}(\mathbf{J}^T\mathbf{J}))$ approssimativamente diagonale, quindi il metodo si comporta come la discesa del gradiente. Con un λ molto piccolo il metodo tende a comportarsi come Gauss-Newton. Ad ogni iterazione viene fissato il valore λ più opportuno per l'iterazione successiva: se la soluzione trovata all'iterazione corrente diminuisce l'errore della stima rispetto al passo

precedente, λ viene diminuito, altrimenti viene aumentato. Questo permette di utilizzare un metodo simile alla discesa del gradiente quando si è lontani dal punto di minimo, mentre, quando l'algoritmo è nei pressi del minimo, si utilizza Gauss-Newton per una convergenza più rapida [39].

Si noti come questi algoritmi presuppongono una buona inizializzazione poiché potrebbero incontrare dei minimi locali, e nessun meccanismo permette di capire se esiste un altro minimo "migliore" della funzione f.

2.2.3 Tracking Bayesiano

Si è visto che il problema del tracking consiste nel trovare la traiettoria compiuta da un oggetto, tracciandola nel tempo. La traiettoria è la serie di posizioni che il veicolo assume nel tempo. Se considerassimo la posizione come stato, allora il tracking è la stima dell'andamento dello stato dell'oggetto nel tempo. In termini statistici, significa stimare la densità dello stato istante per istante basandosi sulle osservazioni compiute. Spesso non si vuole tracciare solo la traiettoria, ma l'interazione dell'oggetto con il mondo, quindi nello stato vengono incluse tutte le informazioni che si ritengono necessarie da stimare, oltre alla posizione. Per il tracking 3D dei veicoli, vengono spesso inclusi ad esempio l'orientamento nel piano definito dal GPC e la velocità.

Indicando lo stato al tempo t con s_t, l'osservazione con z_t e con w e v due rumori gaussiani, rispettivamente, dello stato e dell'osservazione, il problema è modellato come un sistema dinamico:

$$s_{t+1} = f(s_t, w_{t+1}) \tag{2.14a}$$
$$z_{t+1} = h(s_{t+1}, v_{t+1}). \tag{2.14b}$$

A ogni istante $t+1$, lo scopo è quello di trovare la densità dello stato s_{t+1}, ovvero la distribuzione di:

$$Bel(s_{t+1}) = p(s_{t+1}|z_{1:t+1}) \tag{2.15}$$

dove $z_{1:t+1} = \{z_{t+1}, z_t, \ldots, z_1\}$. A volte questa quantità è chiamata *belief*, o *probabilità a posteriori*. Per semplificare il calcolo, viene assunta vera l'ipotesi di Markov, ovvero la misura dipende solo dallo stato corrente, e lo stato corrente dipende solo dallo stato precedente. In pratica questo porta a scrivere la seguente equazione di predizione:

$$\begin{aligned} Bel^{-}(s_{t+1}) &= p(s_{t+1}|s_t, z_{1:t}) \\ &= \int p(s_{t+1}|s_t) p(s_t|z_{1:t}) ds_t \\ &= \int p(s_{t+1}|s_t) Bel(s_t) ds_t \end{aligned} \tag{2.16}$$

dove $p(s_{t+1}|s_t)$ è nota come *probabilità a priori*, o *termine a priori*, dato che indica la probabilità di uno stato noto il precedente, indipendentemente dalla misure. In seguito ad una nuova osservazione z_t, viene aggiornata la stima della densità dello stato al tempo corrente con la regola di Bayes, per questo il tracking che usa queste relazioni è detto *Bayesiano*. Il belief aggiornato è:

$$\begin{aligned} Bel(s_{t+1}) &= \alpha_t p(z_{t+1}|s_{t+1}) p(s_{t+1}|s_t, z_{1:t}) \\ &= \alpha_t p(z_{t+1}|s_{t+1}) Bel^-(s_{t+1}) \end{aligned}, \tag{2.17}$$

dove $p(z_{t+1}|s_{t+1})$ viene detta *probabilità di verosimiglianza* e $\alpha = \frac{1}{P(z_{t+1}|z_{1:t})}$ è costante di normalizzazione.

Il metodo presentato è iterativo, ma ancora astratto perché non definisce il modo in cui viene rappresentato lo stato s_{t+1} da stimare. A seconda di come viene rappresentato lo stato, o più precisamente la distribuzione di $Bel(s)$, esistono diversi strumenti per risolvere le equazioni presentate, e quindi per fare tracking [20].

Filtraggio alla Kalman

In numerosi sistemi, lo stato e la misura sono rappresentate come distribuzioni gaussiane. Lo strumento utilizzato per il tracking 3D è il *filtro di Kalman*.

Il filtro di Kalman standard può essere utilizzato solo quando le funzioni f e h delle equazioni (2.14a) e (2.14b) sono lineari. A volte questo vincolo non rende possibile modellare alcune dinamiche dell'oggetto tracciato. Per questo motivo viene spesso utilizzata la versione estesa del filtro di Kalman, ovvero l'*Extended Kalman Filter* (*EKF*) o la versione detta *Iterated Extended Kalman Filter* (*IEKF*). Nell'EKF viene linearizzato il sistema definito da (2.14), e al sistema linearizzato vengono applicate le equazioni del filtro di Kalman di predizione e aggiornamento. Il filtro IEKF cerca di diminuire l'errore di linearizzazione applicando più volte l'equazione di aggiornamento per il sistema linearizzato.

Questo strumento è ottimale ([20]) sotto l'ipototesi che il processo stimato sia gaussiano e unimodale, e viene implementato in modo efficiente attraverso l'utilizzo di semplici operazioni tra matrici. Lo svantaggio sta nel fatto che lo stato può essere rappresentato solo con una distribuzione unimodale gaussiana e che il sistema modellato deve essere lineare. In alcuni casi queste assunzioni risultano troppo limitanti.

Multihypothesis Tracking

Un modo per trattare problemi di tracking in cui lo stato da stimare è più complicato, ovvero ha una distribuzione multimodale, consiste nel rappresentare lo stato stesso come una mistura di gaussiane come proposto da Reid

[52]. Lo stato viene cioè rappresentato con una somma di distribuzioni gaussiane pesate; ad ognuna di esse è associato un filtro di Kalman che predice lo stato e aggiorna la stima. Quanto più un filtro predice correttamente, ovvero tanto più la predizione è vicina alla misura, tanto più il peso della gaussiana corrispondente è maggiore. Questo metodo è noto in letteratura come *Multihypothesis Tracking* (*MHT*). Questo metodo permette dunque di superare una limitazione di unimodalità. Lo svantaggio è l'aumento significativo della complessità computazionale.

Particle Filter

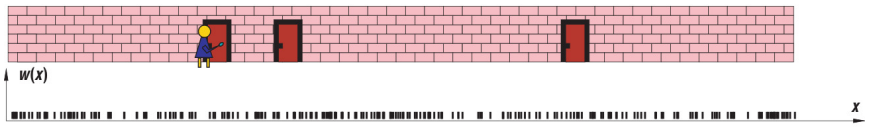
Un approccio differente per superare le limitazioni di unimodalità e linearità del filtro di Kalman viene proposto con il *particle filter*. La distribuzione dello stato viene rappresentata attraverso un insieme pesato di ipotesi, dette *particelle*.

Il passo di predizione consiste nel campionare la distribuzione di probabilità data dai pesi associati agli stati. L'aggiornamento del filtro avviene pesando i campioni trovati nel passo di predizione in base alla verosimiglianza $p(z|s)$. All'inizio del tracking tutti i possibili stati hanno la stessa probabilità, quindi, lo spazio degli stati viene campionato uniformemente: i pesi delle particelle sono uguali. In Figura 2.3 è illustrato un esempio tratto da [20]. Un robot, dotato di un sensore che rileva la presenza di una porta, cammina nel corridoio. All'inizio lo stato, ovvero la posizione del robot lungo s, viene inizializzato con probabilità uniforme, quindi la distribuzione dei campioni è uniforme lungo s e i pesi $w(s)$ sono uguali (Figura 2.3a). A seguito della prima osservazione viene aggiornato il filtro incorporando la verosimiglianza $p(z|s)$ nel termine di $belief$. Questa viene utilizzata per calcolare i pesi delle particelle (Figura 2.3b). Il robot si sposta. Ora la predizione viene fatta campionando non più uniformemente ma in accordo con i pesi ottenuti al passo precedente, aggiornando i pesi in modo che siano tutti uguali (Figura 2.3c). Il robot osserva una porta, quindi aggiorna (Figura 2.3d). quando si sposta si nota come la distribuzione predetta e campionata (Figura 2.3e) è più "concentrata" dove si trova il robot. [1]

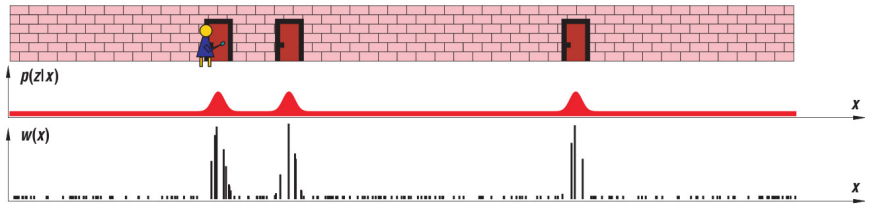
2.3 Tecniche model-based

Tra gli approcci al tracking 3D di maggior efficacia si trovano quelli che utilizzano un modello degli oggetti da tracciare. Attraverso il confronto del modello con l'immagine ripresa si ottiene una stima della posizione e

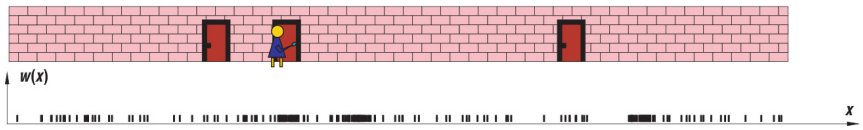
[1] Nel prossimo capitolo vediamo una trattazione più sistematica del particle filter dato che è uno strumento utilizzato in uno dei nostri due sistemi.



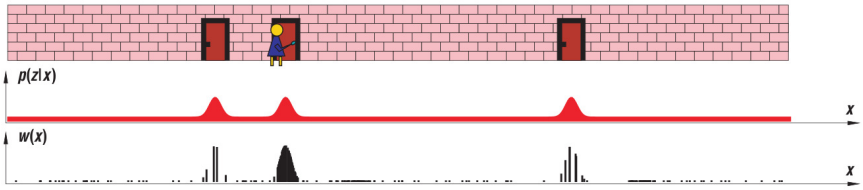
(a) *Inizializzazione filtro.*



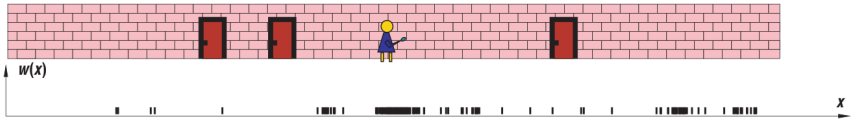
(b) *Aggiornamento del filtro con la prima osservazione.*



(c) *Predizione.*

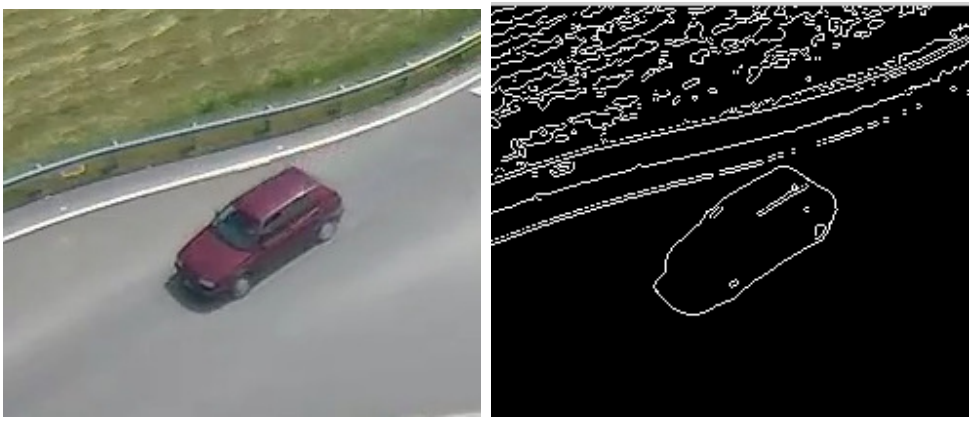


(d) *Aggiornamento.*



(e) *Predizione.*

Figura 2.3: Immagini tratte da [20].



(a) (b)

Figura 2.4: Esempio di edge calcolati su una porzione di immagine.

dell'orientamento dell'oggetto. Raccogliendo la storia di tutte le pose e correlando le pose degli oggetti nel tempo in modo opportuno è possibile ricostruire la traiettoria 3D. Il tracking effettuato secondo questo genere di tecniche prende il nome di *model-based*. [2]

2.3.1 Edge-based

L'approccio edge-based al tracking 3D è apparso per primo. Esso consiste nel confronto tra i segmenti del modello 3D dell'oggetto proiettati sull'immagine, e gli edge, ovvero i bordi dell'oggetto stesso osservato nell'immagine.[3] In Figura 2.4 viene presentato un esempio di cosa si intende per edge (calcolati sulla Figura 2.4b).

Nei sistemi che seguono questo approccio si ha a che fare con una algoritmo che esegue il tracking, detto *tracker*, implementato, ad esempio con il filtro di Kalman, che, nota la posizione iniziale degli oggetti, predice la pose dell'oggetto negli istanti successivi. Questo permette di partire dalla pose predetta per ricercare la posizione e l'orientamento per cui il modello corrisponde il più possibile agli edge dell'immagine.

Esistono due modi per tener conto dell'informazione associata ai contorni dell'immagine. Un metodo consiste nel valutare il gradiente dei pixel dell'immagine e cercare i valori più significativi nell'intorno della pose predetta. Confrontando questi gradienti con il modello viene raffinata la stima della pose, ipotizzando che nell'intorno della pose predetta si trovino i contorni dell'oggetto i quali hanno un gradiente elevato. L'altro metodo prevede l'e-

[2] In questo paragrafo la classificazione utilizzata per presentare le varie tecniche ricalca in buona parte quella presentata in [36].

[3] Di seguito verranno utilizzati indistintamente i termini *edge*, *lati* e *contorni*

strazione esplicita degli edge dell'immagine, confrontati poi con i segmenti proiettati del modello dell'oggetto.

Edge-based con gradiente

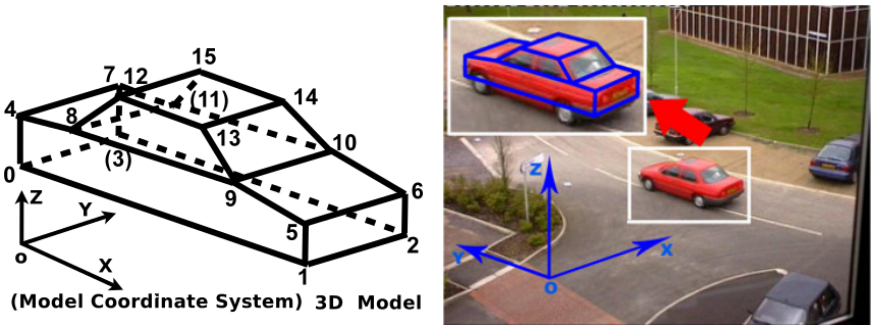
Un esempio del primo metodo è il sistema RAPID [24]. In questo sistema vengono campionati gli edge del modello dell'oggetto da tracciare ottenendo dei punti chiamati control point. Supponendo di conoscere la pose predetta dal tracker vengono proiettati i control point sull'immagine. Tramite due control point vicini si ottiene un'approssimazione della proiezione degli edge sull'immagine. Partendo da ognuno dei control point viene cercato, nell'immagine, il gradiente di maggiore intensità lungo la direzione perpendicolare alla direzione approssimata dell'edge proiettato. Viene calcolata quindi la distanza tra ogni control point e il punto di maggiore intensità, che si suppone appartenga a un edge nell'immagine. Infine, partendo dalla pose stimata, si stima il moto necessario per raffinare la pose in modo che venga minimizzato il quadrato di queste distanze, con il metodo dei minimi quadrati. Il tracker utilizzato, detto alpha-beta, ingloba l'informazione sul moto trovata per stimare la pose successiva.

In [69] viene presentato un sistema di localizzazione di veicoli a partire da una pose stimata. Viene proiettato il modello (Figura 2.5a) sull'immagine. Per ogni segmento del modello proiettato, di orientamento α, viene controllata la regione quadrata di dimensioni L, la lunghezza del segmento, e $2w$, con w che indica la distanza massima dal segmento che si vuole controllare (Figura 2.6). La pose del modello approssima tanto meglio la pose del veicolo relativamente al segmento in esame quanto più il gradiente dell'immagine in questa regione d'intorno è perpendicolare al segmento. Per esprimere ciò, si considera l'intensità $m(x, y)$ e l'orientamento $\beta(x, y)$ del gradiente di ogni punto interno alla regione. Chiamando S_i i pixel nella regione attorno al segmento, e d_i la distanza dell'i-esimo punto dal segmento corrente, il contributo di un segmento l alla funzione di valutazione della bontà è:

$$E_l = \sum_{i|S_i \in RectangularRegion} |m(S_i) \cdot \sin(\beta(S_i - \alpha)| \cdot G_{0,w}(d_i) \tag{2.18}$$

.

La funzione G è una gaussiana il cui contributo permette di pesare l'apporto di ogni pixel alla valutazione, in base alla distanza dall'edge proiettato, dando maggior importanza ai pixel vicino al segmento proiettato. In Figura 2.6 vengono illustrati i contributi alla funzione di valutazione. La somma dei logaritmi dei contributi di ogni segmento è la funzione di valutazione usata dal sistema, la quale viene minimizzata con la discesa del gradiente. In Figura 2.5b viene presentato un esempio del risultato dell'algoritmo. In [70] invece questa funzione viene usata per pesare le particelle di un particle filter e fare tracking sui veicoli.



(a) *Modello del veicolo utilizzato.* (b) *Esempio del risultato della stima della pose.*

Figura 2.5: Immagini tratte da [69].

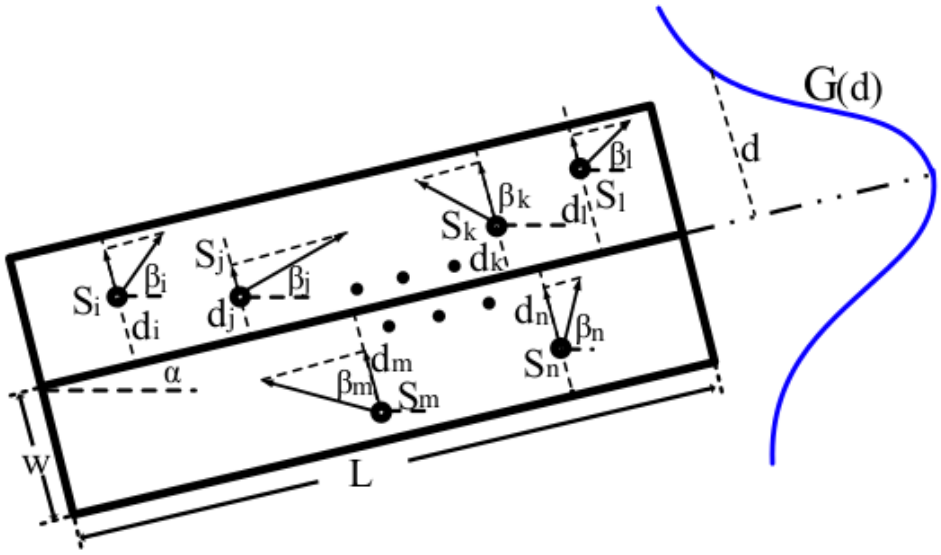


Figura 2.6: Regione rettangolare di cui si calcola il contributo dei pixel per valutare la bontà della pose del modello proiettato. Immagine tratta da [69].

Edge-based con estrazione esplicita degli edge

Di seguito vengono presentati alcuni sistemi che utilizzano l'approccio edge-based calcolando esplicitamente gli edge. Il calcolo degli edge di un'immagine avviene solitamente attraverso il Canny edge-detector [11]. Un caso particolare si trova invece in [33]. Avendo a che fare con veicoli, i cui contorni solo definiti approssimativamente da segmenti di retta, piuttosto che estrarre tutti gli edge con Canny, vengono solo trovati i segmenti di retta con la trasformata di Hough.

In [32] viene presentato un primo sistema di tracking edge-based con estrazione del contorno dall'immagine. Nel sistema proposto vengono utilizzati modelli parametrizzati di diversi tipi di veicoli. Per ogni immagine vengono estratti gli edge, i quali sono confrontati poi con i segmenti del modello proiettati sull'immagine 2D in accordo con la posizione 3D predetta dal tracker. Gli edge sono rappresentati dal vettore $\mathbf{X} = (c_x, c_y, \theta, l)$ e il confronto tra modello e dati estratti dall'immagine avviene tra i segmenti più vicini secondo la distanza di Mahalanobis:

$$d = (\mathbf{X}_m - \mathbf{X}_m)^T(\Lambda_m + \Lambda_d)^{-1}(\mathbf{X}_m - \mathbf{X}_m) \tag{2.19}$$

, dove $\mathbf{X}_m$ e $\mathbf{X}_d$ sono rispettivamente il segmento del modello e il segmento estratto. Λ_m Λ_d sono le matrici di covarianza del segmento del modello, dipendente dalla matrice di covarianza della stima dello stato del veicolo, e del segmento estratto, dipendente dall'incertezza delle posizioni delle due estremità.

La stima della nuova pose avviene minimizzando una cifra di merito che tiene conto sia della distanza media tra i segmenti del modello e dei dati sia della lunghezza dei segmenti, considerando più importante la corrispondenza tra edge lunghi. Un esempio del risultato di questo algoritmo si vede in Figura 2.7. In [32] viene proposto un sistema completo per fare tracking che utilizza un approccio analogo a quello descritto per stimare la pose. Per il tracking sono stati testati tre tipi di tracker: l'EKF, l'IEKF e un'implementazione modificata del filtro IEKF, in cui viene adottata la minimizzazione Levenberg-Marquardt, invece del metodo Gauss-Newton. I risultati sperimentali hanno mostrato che L'IEKF e L'IEKF modificato hanno prodotto migliori risultati.

In [38] e [68] viene definita una metrica basata sulla distanza PLS (Point to Line Segment) per valutare la bontà della pose del modello di un veicolo (Figura 2.8). La distanza misura semplicemente la prossimità di un punto a rispetto ad un segmento l con vertici b_1 e b_2 in questo modo:

$$D(a,l) = \begin{cases} |\overrightarrow{aa'}| & \text{se la proiezione di a su l giace tra } b_1 \text{ e } b_2 \\ \min_{b \epsilon (b_1, b_2)} |\overrightarrow{ab}| & \text{altrimenti} \end{cases} \tag{2.20}$$

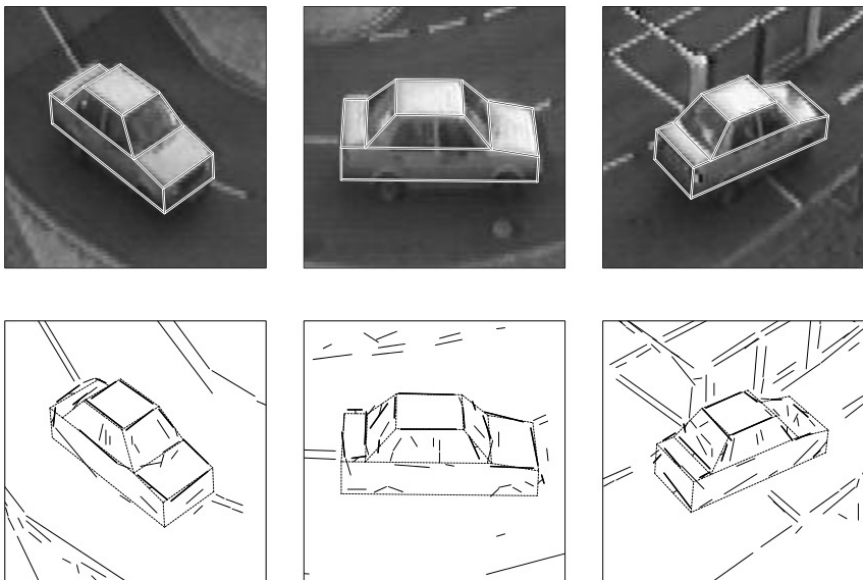


Figura 2.7: Immagine esempio del risultato dell'algoritmo di stima della pose tratta da [32].

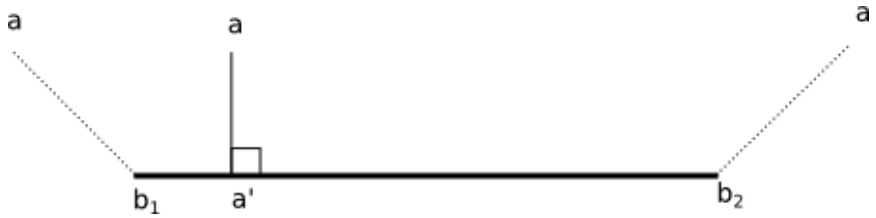


Figura 2.8: Point to Line Segment.

. Tale funzione di valutazione viene definita tenendo in considerazione la distanza PLS dei punti degli edge nell'immagine rispetto ai segmenti del modello proiettati. Successivamente, il problema di trovare la pose del veicolo nel mondo 3D viene scomposto in due sotto-problemi.

Considerando il GPC, e partendo dalla pose predetta dal filtro di tracking, si raffina la pose del modello prima trovando la traslazione 3D del centro del modello, poi la rotazione 3D attorno all'asse del modello stesso. Minimizzando la funzione di valutazione definita sulla distanza PLS, si trova la traslazione del modello proiettato nel piano immagine. Infine, calcolando gli angoli formati dagli edge dell'immagine con gli edge del modello proiettati, si calcola la rotazione del modello in 3D mappando l'angolo trovato con una corrispondente rotazione in 3D. Mentre [68] si occupa solo del problema della localizzazione del veicolo, in [38] viene eseguito anche il tracking e il filtro utilizzato in questo caso è una versione dell'EKF.

I due sistemi presentati affrontano il problema del calcolo della pose ba-

sando il confronto tra il modello e l'immagine su due distanze differenti. Altri lavori presentano altre distanze utilizzate come cifre di merito per valutare la bontà della proiezione del modello [40, 21, 53, 60, 46]. In questi lavori viene stimata la pose raffinando la pose predetta dal tracker, implementato nel modo ritenuto più opportuno dagli autori, minimizzando la cifra di merito con Gauss-Newton oppure con Levenberg-Marquardt. Il tracker provvede poi a integrare le pose ottenute come nuove misure, e a predire la prossima pose in accordo con il modello di moto definito sul modello 3D.

Rispetto ad altri, gli approcci edge-based offrono robustezza a cambiamenti di luminosità interni alla scena. Un aspetto che influenza negativamente questi approcci, invece, è la presenza di disturbi sugli edge estratti dall'immagine. Questo porta a una corrispondenza non corretta dei segmenti del modello sugli edge dell'immagine, in particolare, durante la minimizzazione della cifra di merito, si incontrano minimi locali dovuti al rumore.

2.3.2 Region-based

Gli approcci region-based consistono nel confronto tra la regione occupata dall'oggetto nell'immagine e la regione occupata dal modello proiettato. Essi sono simili agli edge-based, ma se da un lato questo metodo riduce il problema dei minimi locali e dei contorni non ben definiti dall'altro è più difficile calcolare la regione di interesse rispetto agli edge [8]. A questo punto è utile introdurre una tecnica molto utilizzata nella visione artificiale per individuare gli oggetti che si muovono nella scena.

Background Subtraction Attraverso la *background subtraction* vengono individuati gli oggetti in movimento come quelle regioni connesse di immagine, dette *blob*, che si differenziano dall'immagine di sfondo. L'immagine dei soli pixel considerati non appartenenti allo sfondo viene chiamata *foreground* (Figura 2.9). Un lavoro che sintetizza diversi approcci si trova in [49]. Il problema principale di questo metodo è modellare l'immagine di sfondo. La letteratura è ricca di lavori in questo senso [59, 16]. Se l'immagine di sfondo è nota a priori, il problema non si pone. Nei casi reali però, lo sfondo deve essere appreso in qualche modo perché non è noto.

Un modo semplice per stimare lo sfondo è quello di calcolare la media di ogni pixel per un arco temporale sufficientemente lungo. Se, però, un oggetto si ferma per troppo tempo in una posizione, la stima effettuata può non risultare affidabile. Esistono altri metodi più sofisticati, tra cui uno dei più famosi è la Gaussian Mixture Model (*GMM*) [71] che modella la storia di ogni pixel come mistura di gaussiane e da essa ricava l'informazione necessaria a modellare lo sfondo.

I problemi che sorgono nella stima dello sfondo sono dovuti al cambiamento di luminosità nel tempo. Tenendo uno sfondo fisso, l'algoritmo non

(a) *Immagine di partenza .*

(b) *Foreground risultante dopo la background subtraction. In bianco i pixel classificati come appartenenti al foreground.*

Figura 2.9: Esempio di background subtraction.

si comporta bene quando la luminosità complessiva della scena aumenta o diminuisce. Inoltre, sottraendo lo sfondo, spesso vengono incluse le ombre nelle regioni classificate come in movimento: questo è un problema perché non permette di misurare facilmente alcune proprietà dell'oggetto tracciato, come ad esempio, il centroide e la forma. A causa del rumore, della prospettiva della scena e delle ombre, capita che una regione considerata in movimento comprenda al suo interno più oggetti.

I sistemi proposti in questa tesi partono da dati estratti da un tracker 2D sviluppato in un lavoro precedente, il quale estrae dal filmeto le informazioni sugli oggetti in movimento adottando la background subtraction con la stima dello sfondo come media dei pixel e aggiornamento del background per far fronte ai cambi di luminosità.

Sistemi region-based L'approccio region based è spesso utilizzato nell'ambito del tracking 2D. Il sistema sviluppato precedentemente a questa tesi individua le regioni occupate dagli oggetti in movimento ad ogni istante. Inseguendo queste regioni esegue la stima della traiettoria. Questo tipo di tracking viene detto region-based, ed è molto diffuso nei sistemi che lavorano sul piano 2D dell'immagine. Nell'ambito del tracking 3D non si trovano invece un gran numero di sistemi che lavorano solo con i blob estratti dalla background subtraction. A volte infatti questo approccio viene utilizzato principalmente per fare classificazione e poi inferire la pose dell'oggetto di cui fare tracking (Paragrafo 2.3.6). Oppure si trova in combinazione con

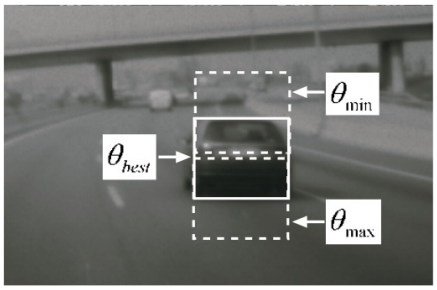


Figura 2.10: I rettangoli tratteggiati indicano l'intervallo entro il quale viene ricercata la faccia dell'auto e in rettangolo con i bordi continui indica la posizione classificata come migliore per rappresentare la faccia del veicolo.

altri approcci (Paragrafo 2.3.5). Nonostante ciò si trovano alcuni sistemi che implementano esclusivamente il tracking 3D region-based.

Un'applicazione del tracking region-based si trova in [50]. In questo articolo viene presentato un sistema per tracciare i veicoli visti dall'interno di un'auto. Il modello utilizzato è un semplice parallelepipedo. Partendo dalla stima della pose predetta, il sistema proietta la faccia del parallelepipedo sul piano immagine (Figura 2.10). Nell'intorno di questa regione viene ricercata la faccia posteriore o anteriore della macchina. Questo passo avviene con un classificatore che permette di capire in quale zona si ha la parte di immagine più simile alla parte anteriore o posteriore di un'auto. In questo modo è possibile stimare la nuova pose del veicolo, integrando la misura nel tracker. Il tracker è implementato con l'Unscented Kalman Filter, una particolare implementazione del filtro di Kalman.

Anche [10] utilizza l'informazione proveniente dalla background subtraction, implementata come GMM, per eseguire tracking 3D di veicoli. Le regioni estratte sono confrontate con modelli di diversi tipi di veicoli proiettati sul piano immagine con diverse pose attorno alla pose stimata. Il confronto che ha maggior successo permette, sia di stimare la pose, sia di classificare il veicolo. Attraverso il filtro di Kalman viene realizzato il tracker per tracciare il veicolo nel Ground Plane.

In [57] viene presentato un sistema di tracking di veicoli visti di profilo. Un lavoro successivo, [58], generalizza il sistema non solo per veicoli visti di profilo. Il tracking avviene in due fasi. Per ogni frame si ricercano le pose dei veicoli con un approccio Montecarlo confrontando la proiezione del modello a parallelepipedo con la regione calcolata con la background subtraction. Poi con l'algoritmo di Viterbi viene trovata la sequenza di pose più verosimile. Un esempio del risultato di questo algoritmo si trova in Figura 2.11. L'articolo in questione è stato il punto di partenza per lo sviluppo di uno dei sistemi proposti in questa tesi. Per una spiegazione più dettagliata si rimanda al Paragrafo 2.6.

Figura 2.11: Esempio dei risultati ottenuti nel sistema [58]. Immagine tratta da [58]

2.3.3 Optical-flow-based

L'optical-flow è il moto apparente di un punto, o più in generale di un oggetto, che si muove sul piano immagine. L'assunzione basilare è che tra un frame e l'altro la luminosità della proiezione dello stesso punto rimanga costante. Questo porta a scrivere il seguente vincolo (2D Motion Constraint Equation):

$$\frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = 0 \tag{2.21}$$

, dove $I = I(x, y, t)$ è l'intensità del pixel (x, y) dell'immagine al tempo t. L'optical flow viene calcolato minimizzando il termine a primo membro dell'Equazione (2.21). I due approcci più noti per il calcolo dell'optical-flow sono l'algoritmo Horn-Schunck [27] e l'algoritmo Lucas-Kanade [41]: essi si differenziano per il metodo di minimizzazione utilizzato. Lucas-Kanade è un algoritmo locale, infatti, vengono usati i minimi quadrati nell'intorno di ogni pixel. Horn-Schunck è invece globale e utilizza la discesa del gradiente.

I sistemi che utilizzano l'optical flow, in linea di principio, non hanno bisogno di un tracker che stimi le posizioni come si è visto per i precedenti approcci. Infatti conoscendo la pose iniziale di un oggetto e conoscendo il moto tramite l'optical flow, è possibile seguire il movimento globale dell'oggetto.

Per esempio, in [4] viene tracciata la testa di una persona. Il modello della testa è un ellissoide, di cui viene definito un modello del moto in 3D. Questo modello permette di generare dei vettori di moto in 2D. Questi vettori vengono confrontati con i vettori calcolati con l'optical flow. Per ogni istante viene quindi stimato il moto 3D come quel moto che minimizza, con la discesa del gradiente, l'errore tra la proiezione del moto e i vettori dell'optical flow.

Il calcolo dell'optical flow può, però, essere soggetto ad errori, ovvero il moto calcolato dall'algoritmo non è effettivamente corrispondente a ciò che avviene nel piano immagine. In realtà, quindi, il metodo di tracking presentato non è robusto in quanto gli errori nel calcolo dell'optical flow si sommano con l'andare del tempo. Facendo un parallelo con l'automatica, è come se il sistema funzionasse ad anello aperto, senza un'indicazione sulla bontà dell'optical flow calcolato. É opportuno dunque sviluppare un metodo per far fronte a questo problema, noto come *drifting*.

In [37] si utilizza l'Equazione (2.21) per inferire la stima del moto della testa di un uomo. Tramite il modello di moto della testa, vengono calcolati i termini $\frac{dx}{dt}$ e $\frac{dy}{dt}$ come la proiezione del moto 3D (ovvero i vettori dell'optical flow) della testa sul piano immagine. Sostituendo i termini così trovati in 2.21 viene calcolata la cifra di merito da minimizzare per ottenere la stima del moto che ha permesso un certo spostamento della testa analogamente al caso precedente. L'algoritmo però, cerca di arginare il drifting. Per fare ciò integra nello stesso sistema: un modulo per la predizione del moto,

un modulo che stima il moto con l'optical-flow, un modulo che sintetizza l'immagine e infine un modulo che corregge il moto. Il modulo che sintetizza l'immagine, permette di attenuare il drifting; attraverso il confronto dell'immagine con texture dell'oggetto da tracciare si inferisce la posizione del modello più verosimile rispetto all'immagine. Questo permette, in un certo senso, di chiudere il l'anello del sistema con la retroazione sull'errore della pose.

Un altro problema tipico dell'approccio optical flow based è dovuto al fatto che il moto 3D degli oggetti presenti nella scena, viene approssimato linearmente nel piano immagine. Nel caso in cui il moto degli oggetti non è sufficientemente lento quest'approssimazione può risultare grossolana. Inoltre, dato che il vincolo 2.21 si basa sull'ipotesi che tra due frame l'intensità di un punto della scena sia costante, quando ci sono repentini cambi di luce nella scena questa assunzione può introdurre errori macroscopici nella stima del moto.

In [9] vengono affrontati gli ultimi due problemi presentati. Per diminuire gli errori dovuti alla linearizzazione nel vincolo 2.21, viene utilizzato il vincolo non linearizzato:

$$I(x, y, t) = I(x + u, y + v, t + 1) \tag{2.22}$$

, dove u e v rappresentano il vettore del moto del pixel dal frame al tempo t al frame al tempo $t + 1$. Per affrontare i problemi dovuti all'assunzione di luminosità costante, vengono considerati i valori del gradiente di ogni immagine piuttosto che i valori di luminosità dell'immagine. L'equazione 2.22 diventa quindi:

$$\nabla(x, y, t) = \nabla(x + u, y + v, t + 1) \tag{2.23}$$

.

2.3.4 Feature-based

Un approccio che condivide lo spirito dell'optical-flow-based è il *feature-based*. In questo caso vengono individuate le features e il tracking avviene ricercandone la corrispondenza tra frame successivi. Per feature si intendono zone nell'immagine che hanno caratteristiche particolari, facilmente distinguibili tra loro e dalle zone vicine. L'idea è sempre quella di confrontare il moto sul piano immagine con la proiezione del moto di un modello 3D degli oggetti da tracciare. Le feature da tracciare possono essere diverse, ad esempio i corner, trovati come in [25] oppure le Good Features to Track del filtro KLT ([41],[61],[56]).

In letteratura si trovano lavori feature-based soprattutto per il tracking del volto umano. Un esempio è il lavoro in [67]. Il tracking avviene sui corner dell'immagine. Essi vengono tracciati e viene fatto muovere un modello 3D

in accordo con il moto registrato. Ad ogni istante la pose stimata viene rifinita grazie all'utilizzo di un altro tipo di feature, ovvero delle piccole parti di immagini attorno ai corner, dette patch. La texture di queste patch viene memorizzata nel modello e resa indipendente ai cambiamenti di luminosità, quindi per ogni corner viene confrontata la texture dell'immagine con la texture registrata per il modello nei pressi del corner.

In [44] viene implementato un altro algoritmo di tracking che utilizza le patch. In questo caso viene campionato il modello del volto umano. Poi vengono tracciate direttamente le patch, centrate non più sui corner, ma sui campioni del modello. Il vincolo che permette di tracciare le texture delle patch è analogo a quello dell'optical flow 2.21. La pose viene quindi trovata minimizzando l'errore dovuto a questo vincolo con Gauss-Newton, e muovendo virtualmente il modello in accordo con il moto calcolato.

L'approccio in esame è simile all'optical-flow-based e, come tale, condivide anche il problema del drifting già citato. Un modo per far fronte a questo problema consiste nell'introdurre i cosiddetti *keyframe* ([14, 63, 66]). Questi sono un insieme di frame che ritraggono l'oggetto da tracciare in varie pose. Per ogni frame si definiscono manualmente le corrispondenze 2D-3D delle feature con il modello. I keyframe vengono usate come riferimento per il tracking. Le feature dell'immagine corrente vengono confrontate con quelle del keyframe in cui l'oggetto ha la pose più simile (per questo si utilizza la pose calcolata nel frame precedente). Dal confronto si evince lo spostamento dell'oggetto rispetto al keyframe e quindi si deduce la nuova pose.

Questo metodo però ha due svantaggi: non sempre si possono avere abbastanza corrispondenze tra il frame corrente e il keyframe; inoltre non è assicurata la consistenza temporale nel tracking tra un frame e l'altro, dato che ogni frame computa indipendentemente la pose. In [66] si cerca di superare anche questi svantaggi considerando l'informazione dei keyframe e dei frame precedenti. A questo scopo formula il problema del tracking come bundle-adjustment. Ovvero viene minimizzato l'errore di retroproiezione dei punti tracciati frame per frame e dei punti confrontati con il keyframe:

$$\min_{\mathbf{P}^t, \mathbf{P}^{t-1}, \mathbf{N}_i} \left\{ r^t + r^{t-1} + \sum_t s_i^t \right\} \tag{2.24}$$

dove s_i^t è l'errore di retroproiezione del punto $\mathbf{N}_i$ al tempo t sul modello 3D rispetto al moto misurato nella corrispondenza frame per frame delle feature; r^t è l'errore di retroproiezione dei punti al tempo t confrontati con il keyframe.

2.3.5 Approcci ibridi

Finora i lavori presentati sono costituiti da un algoritmo di tracking implementato secondo un approccio singolo. Per cercare di superare i problemi

che sorgono con un approccio specifico, alcuni ricercatori hanno implementato algoritmi in cui vengono combinati più metodi. Approcci ibridi tra optical-flow-based ed edge-based per il tracking 3D di veicoli si trovano ad esempio in [22], [13], [65] e [1].

In [22] viene implementato un tracker con l'IEKF in cui la misura della pose viene stimata sia con gli edge che con l'optical flow con metodi simili a quelli presentati precedentemente. Combinando queste due misure è possibile avere sia un'informazione globale sul moto grazie all'optical flow, sia un'informazione locale sugli edge che pemettono solitamente una stima molto precisa, ma sensibile all'errore di modellazione dell'oggetto, in questo caso il veicolo. [13] integra in modo analogo i due approcci in un filtro di Kalman.

Un approccio che utilizza sia gli edge che le feature è [65]. L'integrazione avviene utilizzando il metodo feature-based in [66], sostituendo nell'equazione (2.24), i termini r^t e r^{t-1} con gli errori di retroproiezione dovuti alla stima della pose tramite gli edge. Infine [8] esegue il tracking combinando tre differenti approcci: optical-flow, feature e region. Anche in questo caso le tecniche utilizzate per i tre approcci sono fondamentalmente quelle presentate in precedenza, le feature utilizzate sono estratte con SIFT, ma la logica di tracking rimane la stessa. L'obiettivo di [8] è quello di integrare le tre tecniche in un tracker unico, tale che pesi la stima della pose tra i tre metodi a seconda del livello di confidenza che offrono frame per frame.

2.3.6 Classificazione

Un problema collegato con il tracking 3D è quello della classificazione dei veicoli. Alcuni sistemi presentati in letteratura hanno lo scopo di capire a che categoria appartiene un certo veicolo tracciato. Il problema viene affrontato attraverso diversi approcci, ma il più diffuso è sicuramente quello in cui vengono confrontati i modelli delle diverse categorie con l'immagine. Considerando questo approccio, nonostante la classificazione sia un problema diverso dal tracking 3D, in molti casi i sistemi che fanno classificazione fanno implicitamente anche tracking 3D. Infatti, per decidere a quale categoria appartiene un veicolo attraverso il confronto tra modelli, si stima implicitamente anche la pose, e alcuni sistemi di classificazione fanno anche tracking [42, 1, 10, 62] ottenendo perciò una traiettoria 3D.

Ad esempio in [42] viene effettuato tracking in 2D region-based. Per rendere più rapida la computazione viene utilizzato questo tracking ogni ΔT e nel resto del tempo viene fatto tracking feature-based. Successivamente viene confrontata la regione occupata dal veicolo con la proiezione di sei modelli di veicoli diversi, di un modello di motocicletta e di pedone con diverse pose. Considerando la proiezione del modello la cui forma è più simile al veicolo tracciato permette di fare la classificazione. Non è scopo dell'articolo, ma se

considerassimo la pose con cui si ha la maggior somiglianza, si avrà anche l'indicazione della pose 3D del veicolo nel mondo.

In [1] viene utilizzata la background subtraction per individuare i veicoli in movimento nella scena. Viene eseguito tracking 2D sui blob. Successivamente, viene calcolato l'optical flow relativo alle regioni estratte. Per ogni regione l'optical flow medio stima l'orientamento del veicolo; la posizione è data dal centro della regione. Infine viene classificato il veicolo confrontando gli edge del blob con gli edge di quattro modelli di veicoli considerati con la pose stimata. Anche in questo caso non si ha un esplicito tracking 3D ma lo si può stimare dalla pose utilizzata per il confronto con i modelli.

Invece in [10] viene eseguito tracking con il filtro di Kalman nella cui variabile di stato si trova la pose 3D. Quindi il tracking eseguito è effettivamente 3D. La pose viene stimata durante la classificazione fatta con un approccio analogo al region-based.

2.4 Altri approcci non Model-based al tracking 3D

Nonostante le tecniche di tracking 3D usate in letteratura siano per la maggior parte model-based, esistono alcuni approcci che evitano l'utilizzo di un modello per stimare la posizione degli oggetti nel mondo. Tra queste tecniche le più interessanti utilizzano più telecamere per inferire con la triangolazione le informazioni 3D necessarie al tracking [26].

Un approccio consiste nell'eseguire tracking 2D indipendentemente sulle immagini delle due telecamere, per poi inferire quali traiettorie corrispondono allo stesso oggetto. Diversi articoli presentano questo approccio con tipi differenti di tracking e di associazione tra i dati delle camere, mettendo in evidenza come il secondo risulti il compito più complesso.

Ad esempio in [6] vengono associati i punti 2D delle traiettorie estratte nella fase di tracking 2D. Le associazioni vengono stimate trovando le coppie che minimizzano la somma degli errori di retroproiezione per ogni telecamera. Così vengono trovati i punti che appartengono allo stesso oggetto. La posizione 3D viene inferita con triangolazione e quindi con il filtro di Kalman viene fatto tracking sui punti nello spazio. In modo analogo in [43] viene fatto tracking 2D di persone in una scena in uno spazio chiuso. L'associazione tra i dati delle telecamere viene impostato come un problema bayesiano, affrontato con il particle filter. Quindi l'articolo si sofferma soprattutto su come associare dati tra diverse camere da cui poi inferire le posizioni 3D.

Un altro tipo di approccio si trova in [64] in cui viene implementata un'evoluzione del meanshift in forma tridimensionale. Il tracking meanshift [12] utilizza una combinazione di funzioni kernel per approssimare la distribuzione dello stato dell'oggetto da tracciare. Solitamente viene utilizzato l'istogramma colore come distribuzione usata per descrivere l'oggetto. Il tracking avviene calcolando il gradiente della densità stimata e seguendo

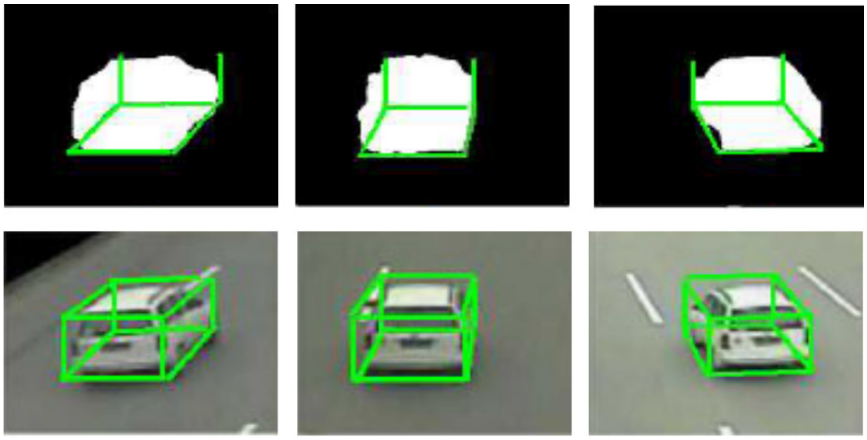


Figura 2.12: Esempio dei risultati dell'algoritmo in [28]. Immagine tratta da [28]

frame per frame la distribuzione che più assomiglia a questa stima. Questo algoritmo è stato ideato per il caso bi-dimensionale, ma in [64] si trova una versione tridimensionale. In questo caso viene utilizzato un altro approccio al tracking con due telecamere: in un primo momento viene individuata la corrispondenza degli oggetti tra le varie telecamere; successivamente viene descritto l'oggetto utilizzando le informazioni derivanti dalle differenti viste; infine viene fatto tracking 3D con meanshift utilizzando questa descrizione dell'oggetto e la triangolazione per inferire la posizione. Secondo gli autori, in questo modo il tracking risulta più robusto dato che è invariante a cambi di vista dell'oggetto: se viene tracciata la testa di un uomo e questa immediatamente ruota, il colore cambia per le singole camere quindi il tracking potrebbe fallire dato che cambia repentinamente l'istogramma dell'oggetto in ogni immagine. Con un istogramma che tenga conto dei diversi punti di vista, invece, la descrizione non cambia e quindi il tracking ha successo.

Un lavoro che si concentra maggiormente sulla stima della pose con telecamere multiple è [28]. In questo articolo viene presentato un metodo per stimare la pose di un veicolo senza utilizzare il modello. In primo luogo, vengono trovati i blob dei veicoli con background subtraction per ogni telecamera. L'assunzione da cui si parte per la stima della pose, è che, fondendo le immagini di un veicolo ripreso da diverse telecamere, la parte comune alle diverse viste, in prima approssimazione è il fondo del veicolo. Ciò permette di stimare un rettangolo che, proiettato sull'immagine, stima la base del veicolo. Stimando l'altezza come l'ultimo pixel del blob lungo la perpendicolare al lato più lontano del rettangolo, viene creato il cuboide, o bounding box 3D, attraverso cui stimare la pose (Figura 2.12).

2.5 Problemi tipici dei sistemi di tracking

Dopo aver delineato un panorama sulle tecniche di tracking 3D, in questo capitolo vengono presentati alcuni tra i più importanti problemi che sorgono nei sistemi di tracking in particolare riferimento ai veicoli.

2.5.1 Inizializzazione del sistema

L'inizializzazione del sistema è un primo problema da affrontare quando si crea un algoritmo di tracking. Con inizializzazione si intendono tutti quei passi necessari per dare inizio al tracking di uno o più oggetti. Il filtro di Kalman e altri strumenti come il particle filter sono iterativi e come tale necessitano di uno stato iniziale da cui partire. Lo stato, come detto in precedenza, è solitamente costituito dalla pose dell'oggetto e dalla velocità. Anche per alcuni approcci model-based, è necessaria una pose iniziale da cui partire, come quelli basati sull'optical flow.

Il problema dell'inizializzazione coincide spesso quindi con la stima della pose iniziale degli oggetti nella scena. Nel caso più semplice l'inizializzazione viene fatta manualmente, ovvero l'utente inizializza la pose tramite un'interfaccia. Questo accade ad esempio in [12], [63], [33], [13] e [65]. Oppure esistono sistemi di inizializzazione semi-automatica, in cui l'utente deve selezionare le zone in cui procedere con l'inizializzazione fatta attraverso l'utilizzo di optical flow e background subtraction [22]. Nella maggior parte dei rimanenti casi di sistemi di tracciamento di veicoli ogni autore adotta un sistema automatico ad hoc per inizializzare il proprio algoritmo di tracking [32, 21, 60, 64, 69, 70].

É significativo portare infine un esempio di uno dei pochi articoli che si concentrano soprattutto sull'inizializzazione nel tracking model-based [47]. Gli autori propongono un sistema che si basa sull'utilizzo congiunto di optical flow e confronto degli edge modello-immagine. Prima di tutto viene calcolato l'optical flow; le regioni adiacenti con optical flow simile vengono raggruppate formando così una mappa di possibili oggetti in movimento. Queste regioni vengono analizzate tra frame adiacenti e studiando l'andamento nel tempo delle regioni, viene determinato se sono state originate da rumore o se rappresentano effettivamente il movimento di un veicolo. In queste regioni vengono eliminati i veicoli che sono già stati inizializzati, e quindi per cui non si vuole procedere con l'inizializzazione. Questo passo che utilizza l'optical flow permette di dare una stima piuttosto precisa sull'orientamento e la velocità dei veicoli. Per effettuare la localizzazione, si ricercano i veicoli nelle aree appena selezionate con l'optical flow e si procede al confronto tra edge dell'immagine e edge del modello preso con l'orientamento stimato.

2.5.2 Ombre e cambiamenti di luminosità

Il tracking dei veicoli soffre di problemi relativi alla presenza di ombre. L'ombra di un veicolo è un elemento di disturbo per il tracking perché a volte non è semplice discernere tra veicolo e ombra stessa. Questo ovviamente comporta una difficolta nella stima della pose del veicolo dato che, non ricostruendo in modo preciso la posizione del veicolo, è anche difficile stimarne l'orientamento se viene utilizzato ad esempio un tracking model-based. Questo problema nasce in particolare quando viene utilizzata la background subtraction all'interno dell'algoritmo di tracking per il riconoscimento del moto.

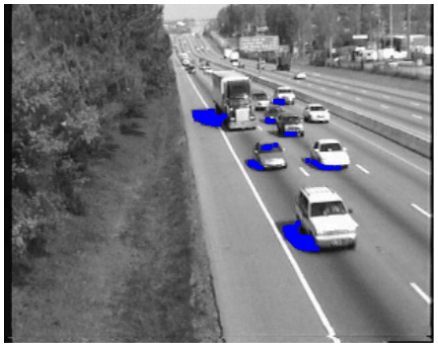
La letteratura è ricca di studi il cui scopo è quello di riconoscere la presenza di ombre per eliminarle, o per poter tener conto dell'esistenza in fase di tracking. A questo fine lo sforzo è quello di individuare le caratteristiche peculiari delle ombre per distinguerle nell'immagine. In [3] viene cercata l'ombra come parte di immagine sotto al veicolo più scura della strade, riconoscendola tramite il calcolo degli edge (Figura 2.13b). In [51] viene utilizzata anche una proprietà di linearità tipica delle regioni corrispondenti alle ombre (Figura 2.13a). In [29] e in [58] viene calcolata l'ombra come la proiezione del bounding box 3D attorno al veicolo considerando l'ora e la posizione nel mondo per determinare l'inclinazione dei raggi solari (Figura 2.13c). Un ragionamento analogo viene fatto in [58] (Figura 2.13d). Questi sono solo alcuni degli approcci per il riconoscimento delle ombre. Non è scopo di questo capitolo nè di questa tesi approfondire ulteriormente questo tema: il tracker 2D già esistente e il sistema di background subtraction permettono infatti di trascurare il problema delle ombre che potrà essere uno dei futuri passi del sistema sviluppato (vedi Capitolo 6).

2.5.3 Occlusioni

Nel tracking di veicoli può capitare che una zona della scena è parzialmente occlusa, e quindi sia difficile riconoscere i veicoli che passano da questo punto. In Figura 2.14 viene mostrato un esempio dell'influenza delle *occlusioni* sulla stima della pose. Questo è un altro tipico problema del tracking. Infatti, un buon algoritmo di tracking è capace di superare il problema delle occlusioni se queste non sono troppo significative.

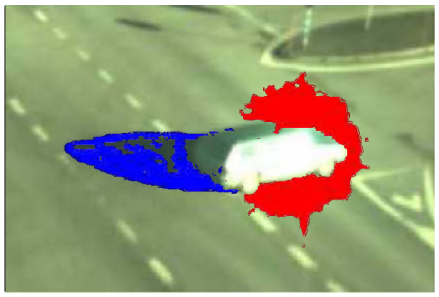
Solitamente il corretto utilizzo del filtraggio alla Kalman o degli strumenti che permettono di predire la pose, è sufficiente nei casi più semplici per superare il problema delle occlusioni [33].

Nei casi di tracking model-based, per valutare la bontà della pose, partecipa al confronto tra modello e immagine solo la parte visibile dell'oggetto [38]. In [23] viene invece modellata la scena in 3D, e vengono tenute in considerazione gli oggetti possibile causa di occlusione per fare ragionamento sulla scena. Questo approccio è difficile da riscontrare in altri sistemi dato

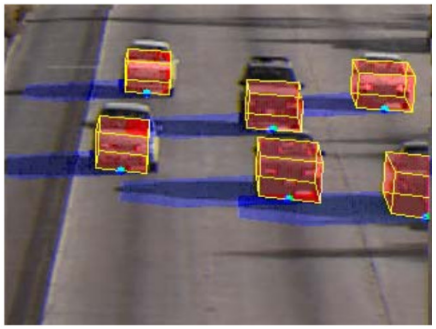


(a) *In blu le parti classificate come ombre.*

(b) *In blu le parti classificate come ombre.*

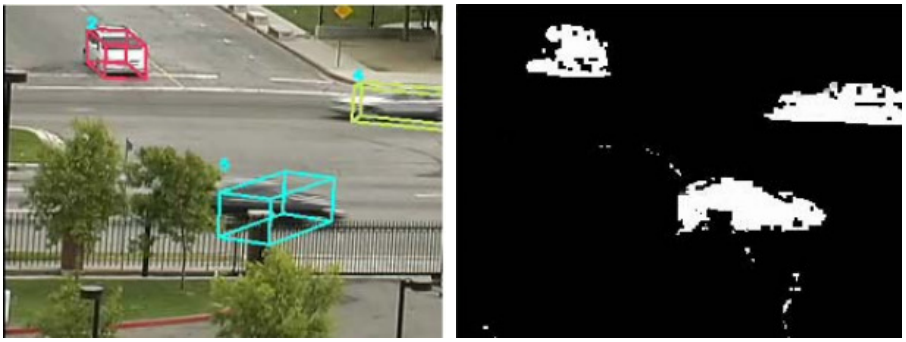


(c) *In blu le parti classificate come ombre, in rosso quelle classificate come alta luminosità.*



(d) *In blu le parti classificate come ombre.*

Figura 2.13: Esempio di riconoscimento di ombre in [51] (Figura 2.13a), in [3] (Figura 2.13b), in [29] (Figura 2.13c) e in [58] (Figura 2.13d).



(a) *Occlusione dovuta alla presenza dell'albero.*

(b) *Foreground corrispondente.*

Figura 2.14: Esempio di problemi dovuti ad occlusioni. Immagini tratte da [58].

che è complesso e quasi mai possibile modellare una scena. Infine [54] è presentato un metodo per gestire le occlusioni nel caso in cui viene utilizzato il tracking con particle filter.

Anche in questo caso, questo problema non riguarda in particolare il lavoro di questa tesi, dato che, come spieghiamo nel capitolo successivo, dal tracker 2D abbiamo già le traiettorie dei veicoli. É sufficiente quanto detto per introdurre il problema, nell'inquadramento generale del tracking.

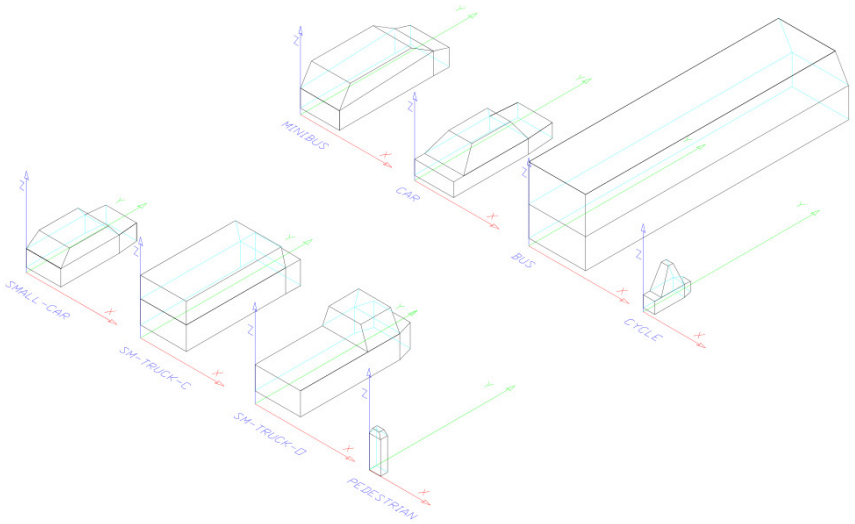
2.5.4 Costruzione del modello

Una questione che sorge nel caso in cui viene compiuto tracking model-based è proprio la costruzione del modello. Non è banale infatti definire un modello adatto per descrivere le grandi varietà di modelli di veicoli presenti in circolazione. Il problema è stato affrontato soprattutto dalla letteratura riguardante la classificazione dei veicoli [18, 17, 31, 35]. Alcuni autori utilizzano modelli fissi di diverse categorie di veicolo (Figura 2.15a), altri utilizzano modelli parametrizzati per cercare di stimare il meglio possibile la classe di appartenenza del veicolo (Figura 2.15b).

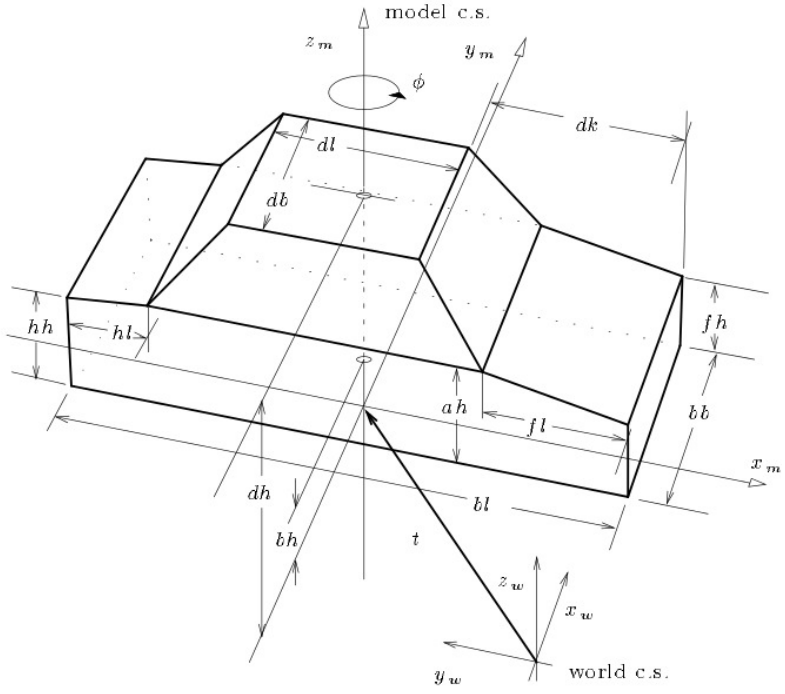
Se, però, lo scopo è quello del tracking 3D, non è necessario avere un modello dettagliato, tanto che in [58] e in [29] viene utilizzato un semplice parallelepipedo per stimare la pose dei veicoli. La maggior parte degli autori utilizza un insieme di modelli ognuno dei quali rappresenta le diverse categorie di tracking.

2.6 L'algoritmo Song-Nevatia

Tra gli algoritmi presentati in letteratura presentati finora, ha suscitato il nostro interesse il lavoro in [58] già citato nel Paragrafo 2.3.2. Questo è uno



(a) *Modelli utilizzati in [42].*



(b) *Modello parametrizzato usato in [31].*

Figura 2.15: Esempio di modelli usati.

dei pochi lavori che utilizzano un approccio Montecarlo per eseguire tracking 3D.

Gli autori utilizzano una tecnica di tracking model-based, in cui il modello utilizzato è un parallelepipedo dalle dimensioni fisse e viene assunto il vincolo GPC. Nell'articolo è rivolta grande attenzione a gestire il problema delle occlusioni tra veicoli e l'algoritmo risultante ha lo scopo di affrontare il problema delle occlusioni tra veicoli, evitando configurazioni in cui i veicoli si compenetrino tra loro. Nei sistemi proposti non ci siamo preoccupati di questo aspetto, ma il nostro scopo è quello di stimare con precisione la storia delle pose dei veicoli di cui già conosciamo una stima della posizione sul piano della rotonda. L'algoritmo in [58] consiste essenzialmente in due fasi: nella prima vengono campionate le pose dei veicoli nel frame per frame, nella successiva viene estratta la successione migliore di pose tra quelle campionate.

2.6.1 Campionamento per ogni frame

Per ogni frame t si conosce il foreground che chiameremo F_t, risultato della background subtraction. A partire da questo si vuole stimare la distribuzione delle possibili configurazioni dei veicoli nella scena, per trovare la configurazione θ_t che massimizza la probabilità a posteriori $p_t(\theta_t|F_t)$. Secondo la regola di Bayes:

$$p_t(\theta_t|F_t) \propto p(\theta_t) \cdot p_t(F_t|\theta_t) \tag{2.25}$$

Per una configurazione il cui numero di veicoli è w, gli autori in [58] definiscono il termine a priori come prodotto di tre fattori:

- $p_w(w|N)$ rappresenta la probabilità di avere w veicoli; N è la stima del numero di veicoli calcolata come rapporto tra l'area totale del foreground e area media di un singolo veicolo.

- $\prod_{i=1}^{w} p_v(M_i)$ è la probabilità del singolo veicolo, nell'articolo questa è una distribuzione uniforme.

- $\prod_{i,j\in[1,k]}^{i<j} p_o(M_i, M_j)$ è la probabilità che due modelli in 3D si sovrappongano; aumentando la dimensione del modello in 3D, questo termine è utilizzato per penalizzare configurazioni in cui le macchine sono troppo vicine.

Il termine di verosimiglianza viene definito a partire dai termini di errore e_1 ed e_2. Il primo rappresenta il rapporto tra pixel del foreground non coperti dalla proiezione del modello e il totale dei pixel di foreground. L'errore e_2 invece è il rapporto tra i pixel del modello proiettato che non appartengono al

foreground e l'area del foreground stesso. Chiamando λ_1 e λ_2 due coefficienti per pesare i due errori si ha:

$$p_t(F_t|\theta_t) = \alpha \cdot e^{-(\lambda_1 \cdot e_1 + \lambda_2 \cdot e_2)} \tag{2.26}$$

Data la complessità dello spazio delle configurazioni che possono assumere i veicoli, non è pensabile utilizzabile un metodo che le valuti tutte. Risulta quindi opportuno un campionamento sparso per stimare la distribuzione della probabilità a posteriori. Il metodo utilizzato degli autori è ispirato al metodo iterativo *Markov Chain MonteCarlo* (*MCMC*), in particolare all'implementazione dell'algoritmo *Metropolis-Hastings*. Lo scopo di questo algoritmo è di partire da un campione qualsiasi, che rappresenta una possibile configurazione, estratto dalla distribuzione a posteriori $p_t(\theta_t|F_t)$ e avvicinarsi iterativamente a una zona a più alta probabilità della distribuzione stessa. Questo avviene sfruttando una distribuzione di probabilità più semplice rispetto a quella di $p(\theta_t|F_t)$ che permette di generare un campione ragionevole θ'_t a partire dal campione precedente θ_t^{k-1}. Questa distribuzione è detta *proposal distribution* e la chiameremo $q(\theta'_t|\theta_t^{k-1})$, con θ'_t indichiamo il *campione proposto* ovvero, la configurazione di veicoli proposta. Con θ_t^{k-1} e θ_t^k indichiamo due campioni successivi nel contesto delle iterazioni dell'algoritmo Metropolis-Hastings.

Con il nuovo campione e il campione precedente viene calcolata la probabilità di passare dal campione vecchio al campione nuovo come:

$$p(\theta_t^{k-1}, \theta'_t) = \min\left\{1, \frac{P(\theta'|I) \cdot q(\theta^{k-1}|\theta')}{P(\theta^{k-1}|I) \cdot q(\theta'|\theta^{k-1})}\right\} \tag{2.27}$$

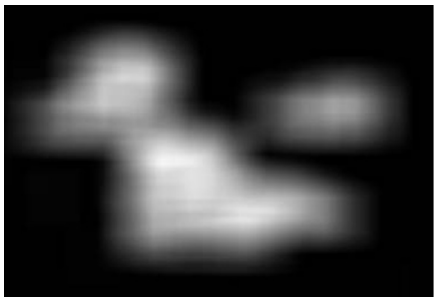
Nell'algoritmo di Song e Nevatia la funzione q non è definita da un'espressione analitica, a differenza di quanto previsto dall'algoritmo originario Metropolis-Hastings. Gli autori preferiscono definire una procedura la cui funzione è la medesima di q: ovvero partendo da una configurazione θ_t^{k-1} per il frame t, viene generata una nuova configurazione proposta θ'_t. La procedura per proporre una nuova configurazione è la seguente. A ogni iterazione dell'algoritmo MCMC, viene scelta casualmente una delle seguenti azioni:

- aggiunta di un nuovo veicolo: viene aggiunto un veicolo a quelli già presenti nel campione campionando la posizione da una distribuzione in cui i valori più alti di probabilità sono concentrati nelle zome con più pixel di foreground; l'orientamento viene campionato dall'istogramma dei vettori dell'optical flow nell'intorno del centro campionato (Figura 2.16).

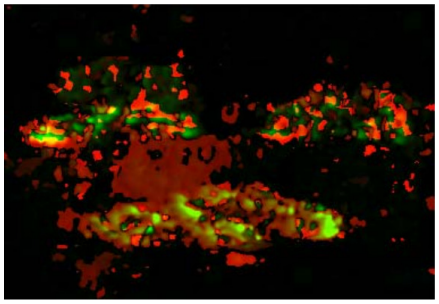
- rimozione di un veicolo a caso.

- sostituzione di un veicolo: viene proposto un veicolo come nel primo caso ma viene successivamente eliminato il veicolo più vicino.

(a) *Foreground.*



(b) *Distribuzione da cui campionare il centro.*



(c) *Optical flow utilizzato (in verde il movimento lungo le x, in rosso lungo le y).*

(d) *Esempio di 14 proposte.*

Figura 2.16: Immagini tratte da [58] che supportano la spiegazione di come vengono proposti nuovi veicoli.

- modifica delle posizioni dei veicoli: la posizione del centro di ogni veicolo viene spostata di una quantità calcolata con un metodo analogo all'algoritmo meanshift, ovvero una quantità che diminuisca l'errore di retroproiezione del modello.

- modifica degli orientamenti: per ogni veicolo viene ricampionato l'orientamento con il metodo visto nel primo punto.

Una volta noto il campione proposto, o nel caso dei [58] la configurazione proposta viene estratto un numero β a caso da una distribuzione uniforme nell'intervallo $[0, 1)$; il nuovo campione θ^k, ovvero la nuova configurazione θ_t^k per la prossima iterazione diventa:

$$\theta^k = \begin{cases} \theta', & \text{se } \beta < p(\theta^{k-1}, \theta') \\ \theta^{k-1}, & \text{altrimenti} \end{cases} . \tag{2.28}$$

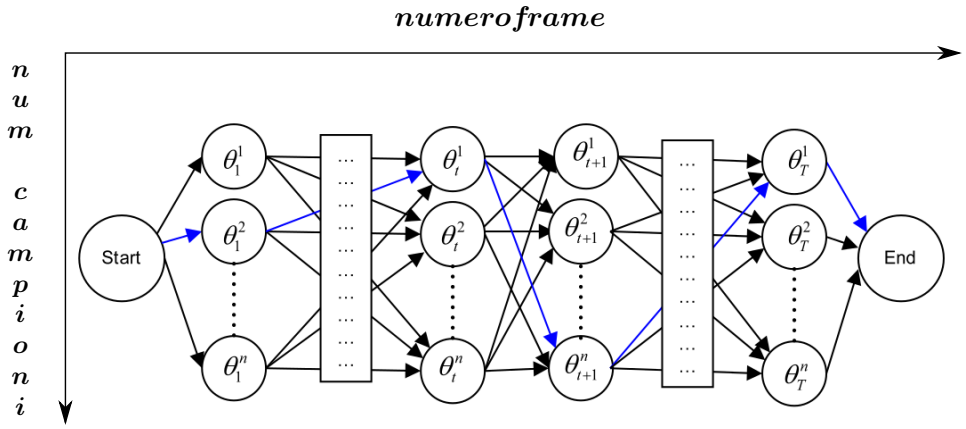


Figura 2.17: Rappresentazione del problema di tracking. In blu la sequenza estratta con l'algoritmo di Viterbi. Immagine tratta da [58].

2.6.2 Tracking con Viterbi

Per ogni frame $1 < t \leq T$, sono stati estratti i campioni $\{\theta_t^1, \ldots, \theta_t^k, \ldots, \theta_t^n\}$ della probabilità $p(\theta_t|F_t)$ dove n è il numero di iterazioni dell'algoritmo Metropolis-Hastings ed per ogni campione è stata calcolata la loro probabilità. Successivamente il problema di tracking viene modellizzato come *Hidden Markov Model* (vedi Paragrafo 3.4.1). Ogni stato-configurazione θ_t^k dipende dallo stato precedente θ_{t-1}^k, e dalla misura, il foreground, all'istante corrente.

I campioni che rappresentano le configurazioni dei veicoli nella scena, formano il grafo in Figura 2.17. Per trovare una soluzione al problema di tracking, viene ricercata la sequenza di configurazioni ottima all'interno di questo grafo che porti dal nodo fittizio *Start* al nodo fittizio *End*. Per sequenza di configurazioni ottima, per un numero T di frame, si intende una sequenza $\{\theta_1^*, \ldots, \theta_T^*\}$ tale che:

$$\{\theta_1^*, \ldots, \theta_T^*\} = \underset{\theta_1^{k_1}, \ldots, \theta_T^{k_T}}{\arg\max} \left\{ \prod_{t=1}^{T} p(\theta_t^{k_t}|F_t) \times \prod_{t=1}^{T-1} a(\theta_t^{k_t}, \theta_{t-1}^{k_{t-1}}) \right\} \tag{2.29}$$

dove con $a(\theta_t^{k_t}, \theta_{t-1}^{k_{t-1}})$ si intende la *funzione di associazione* che valuta la probabilità che la configurazione $\theta_{t-1}^{k_{t-1}}$ sia successiva a $\theta_t^{k_t}$.

Per il calcolo della funzione di associazione, a partire della configurazione di ognuno dei veicoli stimata in $\theta_{t-1}^{k_{t-1}}$ e $\theta_t^{k_t}$ viene proiettato sul piano immagine un modello a forma di parallelepipedo. Le proiezioni dei modelli per l'istante $t-1$ vengono fatte avanzare di una quantità pari a quella stimata nell'optical flow. In questo modo viene compensato il moto compiuto dai veicoli tra due frame. Viene calcolata l'area di sovrapposizione $O(v_{t-1}^i, u_t^j)$ delle proiezioni dei modelli del veicolo v_{t-1}^i e u_t^j. Supponendo che l veicoli

corrispondano tra loro e il resto non corrispondano chiamiamo C l'insieme dei veicoli che corrispondono, $Scomparsi$ l'insieme dei veicoli presenti al frame $t-1$ e non in t e $Apparsi$ l'insieme veicoli presenti al frame t e non in $t-1$. Si ottiene per l'i-esimo campione del frame $t-1$ e il j-esimo campione del frame t:

$$a(\theta_t - 1^i, \theta_t^j) = \prod_{w \in C} O(v_{t-1}^w, u_t^w) \cdot \prod_{w \in Scomparsi} p_{\text{end}}(v_{t-1}^w) \cdot \prod_{w \in Apparsi} p_{\text{new}}(u_t^w) \tag{2.30}$$

Dove con $p_{\text{end}(v_w)}$ si intende la probabilità che un veicolo scompaia, e con $p_{\text{new}(v_w)}$ che appaia. Entrambe hanno un valore di 1 vicino ai bordi del frame e un valore molto basso in altri punti: in questo modo si penalizzano scelte di configurazioni tali che un veicolo scompaia durante il suo percorso, a meno che non stia entrando o uscendo dall'immagine.

Gli autori, risolvono la Equazione (2.29) così calcolata attraverso l'*algoritmo di Viterbi*, ovvero, per ogni frame t ed ogni configurazione j-esima del frame t, e ogni configurazione i-esima del frame $t-1$, viene calcolato il valore di una funzione chiamata equazione di programmazione dinamica:

$$f(\theta_t^j) = \max \left\{ f(\theta_{t-1}^i) \times p(\theta_t^j | I_t) \times a(\theta_t^j, \theta_{t-1}^i) \right\} \tag{2.31}$$

con inizializzazione $f(\theta_0^1) = f(Start) = 1$. Concluso il calcolo, per l'ultimo frame viene scelta la configurazione il cui valore di f associato è massimo. seguendo il percorso che ha portato a questa configurazione viene ricavata la sequenza di configurazioni $\{\theta_1^*, \ldots, \theta_T^*\}$ (ad esempio il cammino blu in Figura 2.17).

Nel prossimo capitolo presentiamo una descrizione più precisa dell'algoritmo di Viterbi e di come il primo dei sistemi proposti si ispira all'algoritmo appena presentato.

Capitolo 3

Dal Tracking 2D al 3D

In questo capitolo presentiamo il percorso che ci ha portato a proporre i nostri algoritmi, descrivendo i dati 2D dai quali partiamo a fare tracking, le motivazioni che ci hanno portato a scegliere il tracking 3D e come quest'ultimo viene eseguito dai nostri algoritmi. Dopo aver presentato il problema del tracking in modo dettagliato, dunque, presenteremo i dati dai quali siamo partiti, descrivendo poi, alla luce di questi dati, gli algoritmi necessari per presentare i due approcci proposti.

3.1 Il tracking 2D

Gli algoritmi sviluppati hanno lo scopo di fare tracking 3D dei veicoli che percorrono un'intersezione rotatoria: la scena è quella mostrata in Figura 3.1. Una telecamera non è sufficiente a riprendere l'intera rotonda; per questo motivo abbiamo tre telecamere sincronizzate che riprendono la stessa rotonda da punti di vista differenti. In prima istanza descriviamo i problemi e i sistemi riferendoci principalmente al caso in cui la telecamera è singola. La gestione del caso multicamera è presentato nel prossimo capitolo come naturale estensione degli algoritmi presentati in questo capitolo.

A differenza dei classici sistemi proposti in letteratura, i nostri algoritmi non processano direttamente i video ma i risultati ottenuti da un tracker 2D che fornisce una stima della traiettoria dei veicoli sul piano immagine.

3.1.1 Il tracker 2D

Il tracker 2D è un insieme di filtri di Kalman estesi (EKF), e idealmente ogni filtro corrisponde ad uno e un solo veicolo. Lo stato del filtro è un vettore $s = [\rho, \theta, v_x, v_y]$: ρ e θ rappresentano la posizione del veicolo in coordinate polari rispetto al centro della rotonda nel piano immagine, v_x e v_y rappresentano la velocità del veicolo. Per posizione si intende il baricentro del veicolo. La misura di un veicolo è il centro del rettangolo, ovvero il *bounding box*,

Figura 3.1: Esempio di scena da analizzare

che circoscrive il blob corrispondente al veicolo, calcolato con la background subtraction: questo punto approssima il baricentro del veicolo. A ogni filtro sono associate due matrici di covarianza che modellano il rumore dello stato e della misura, ovvero di w_t e v_t di (2.14). Le matrici danno un indicazione di quanto siamo confidenti che il veicolo si trovi nello stato del filtro, e di quanto la misura sia affidabile.

L'EKF esegue tracking bayesiano, dopo l'inizializzazione, il suo funzionamento consiste nel passo di predizione e quello di aggiornamento. Quando un veicolo entra nella scena, non ha un filtro associato, ne viene quindi inizializzato uno la cui posizione coincide con il centro del blob del nuovo veicolo. Per ogni frame, durante il passo di predizione viene fatto avanzare lo stato del filtro basando il calcolo sullo stato all'istante precedente. Il nuovo stato ha una posizione prossima a quella del veicolo. Viene associata al filtro la misura del veicolo scegliendo tra le misure di tutti i veicoli presenti nella scena in base alla posizione predetta: questo passo viene detto *Data Association*. Con questa misura viene aggiornato lo stato del filtro e si conclude l'iterazione per il frame corrente.

Per problemi dovuti a occlusioni e cambiamenti di luminosità, è possibile che le misure non siano presenti a ogni istante. In questi casi il filtro procede solo con il passo di predizione. Questo può essere fatto solo per un numero

limitato di volte. Infatti, se non vengono integrate nel filtro misure per troppo tempo e se il veicolo non compie la traiettoria predetta, potrebbe venir associata al filtro una misura di un veicolo diverso da quello tracciato. L'utilizzo del tracking bayesiano sul piano immagine permette dunque non solo di inseguire le traiettorie dei veicoli nella scena, ma anche di superare, almeno parzialmente, problemi di occlusioni e quindi di misure mancanti.

Per ogni veicolo, il risultato del tracking 2D è quindi una traiettoria delineata dalla storia delle posizioni degli stati assunti dal filtri di Kalman. La misura del veicolo per i nostri algoritmi non è più una stima 2D del baricentro calcolata dal blob, ma è il blob stesso. Per questo motivo durante il tracking 2D viene salvato per ogni istante il blob da cui è stato calcolato il centro associato al filtro nella fase di data association.

3.1.2 Perché utilizzare le traiettorie 2D

Gli approcci classici noti in letteratura eseguono direttamente tracking 3D senza passare dalla stima sul piano immagine. L'utilizzo delle traiettorie 2D ci permette di ottenere alcuni importanti benefici nella progettazione del tracker 3D. In primo luogo il tracker 2D implementa un sistema che esegue la data association. Dato che questo processo è già implementato nel tracker 2D, possiamo implementare un sistema 3D che non si preoccupi della data association, ma si concentri esclusivamente sul problema di tracking. Il compito risulterà semplificato, ma il costo da pagare è che gli errori derivati da una data association errata si riflettono nel tracking 3D. Abbiamo deciso di ignorare questa evenienza e focalizzare il nostro sistema sul problema di migliorare la stima della posizione nel mondo dei veicoli. Un ragionamento su questo punto può essere fatto per estendere il lavoro presentato nella tesi e ne discutiamo nel Capitolo 6.

L'utilizzo delle traiettorie 2D permette inoltre di rendere semplice il problema dell'inizializzazione che, come visto nel Paragrafo 2.5.1, è spesso un aspetto critico nei sistemi classici. L'utilizzo dei dati 2D ci permette di avere una inizializzazione sufficientemente precisa della posizione e dell'orientamento. Proiettiamo i primi due centri della traiettoria in 3D: prendendo il primo come posizione iniziale e l'angolo formato dal vettore che va dal primo al secondo punto come orientamento.

Un ultimo benificio che deriva dall'utilizzo delle intere traiettorie 2D è la possibilità di fare smoothing piuttosto che tracking.

Lo smoothing

I nostri algoritmi, a differenza di quanto avviene spesso nei sistemi di tracking che si trovano in letteratura, non eseguono stima tramite un filtro ma tramite uno smoother.

Nel Capitolo precedente abbiamo presentato il tracking bayesiano come stima della distribuzione $p(s_t|z_{1:t})$. Stimare lo stato al tempo t supponendo di conoscere le misure fino al tempo t stesso è un processo che prende il nome di filtraggio. Le applicazioni di tracking, specie quelle real time, devono stimare la posizione dell'oggetto nell'istante in cui è nota la misura, per questo motivo viene tipicamente utilizzato un filtro per affrontare questo problema.

Abbiamo visto però che i dati da cui partiamo per sviluppare i nostri sistemi sono le intere traiettorie 2D, quindi ad ogni istante conosciamo le misure passate presenti e future. Questo ci permette di ottenere una stima *smussata* (*smoothed*) rispetto ai filtri, infatti possiamo tener conto di informazioni aggiuntive date dalle misure future. Il processo di stima dello stato al tempo t date tutte le misure, dall'istante 1 all'istante T, prende dunque il nome di *smoothing*. Analiticamente si può scrivere che il sistema che esegue smoothing stima all'istante t la distribuzione $p(s_t|z_{1:T})$, ovvero $p(s_{1:T}|z_{1:T})$ nell'arco temporale che va dal tempo 1 al tempo T.

3.2 Perché passare al tracking 3D

I dati del tracker 2D offrono una buona approssimazione di quanto accade nella rotatoria. Nel caso però in cui si voglia una stima più precisa, è necessario l'utilizzo del tracker 3D. In particolare i sistemi 3D che proponiamo hanno il compito di superare le limitazioni intrinseche al tracker 2D, riportate qui di seguito.

I blob estratti dalla background subtraction sono una approssimazione del contorno del veicolo e abbiamo visto che nel tracking 2D vengono utilizzati per generare la stima del baricentro del veicolo utilizzata come misura per il filtro di Kalman. Per stimare la posizione del veicolo nel mondo dobbiamo proiettare il baricentro stimato sul piano della rotatoria. Da quest'operazione risulta una stima poco precisa: ne è un esempio la Figura 3.2 che rappresenta schematicamente un camion visto da dietro. I raggi r_1 e r_2 racchiudono la regione del blob sul piano immagine, il raggio r_b indica invece la direzione della retta che passa dal centro della telecamera e per il centroide stimato. Come si vede dall'immagine il centroide proiettato sul piano della rotonda, C_{err} è nettamente differente dalla reale proiezione del centro C sulla rotatoria.

A questo si aggiunge il fatto che, quando un veicolo è parzialmente presente nell'immagine, il tracker 2D propone una stima del centro polarizzata verso la parte di veicolo presente nell'immagine. In Figura 3.3 mostriamo un esempio su un camion, dove questo errore risulta più evidente. É chiaro come il centro, in rosso, sia un'approssimazione del centro del veicolo tanto peggiore quanto più questo si avvicina al bordo. Se, ad esempio, volessimo stimare la posizione del camion in una situazione come in Figura 3.3d il trac-

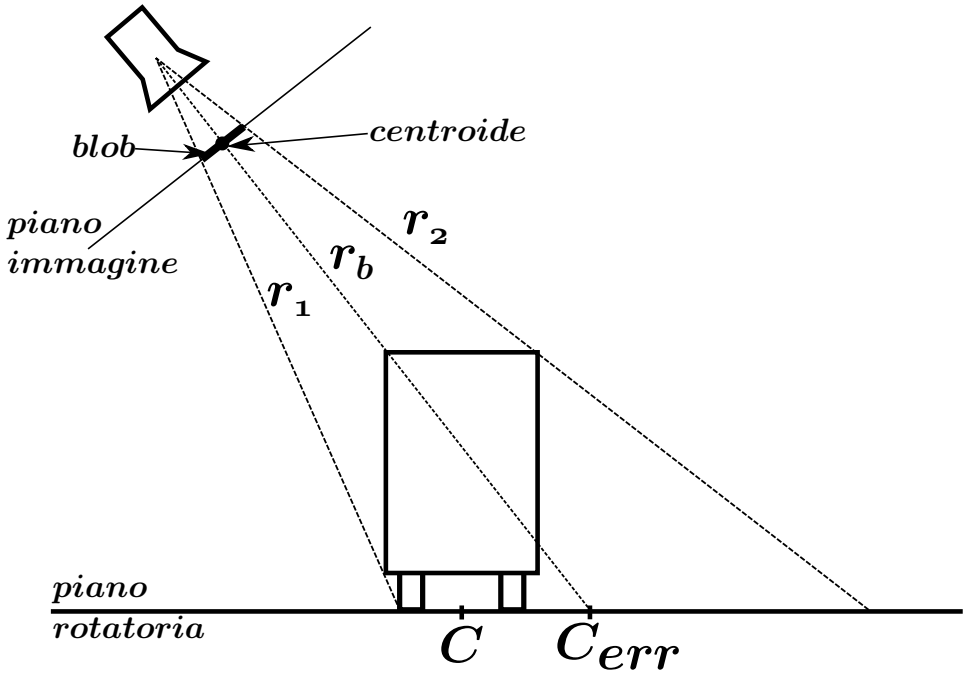
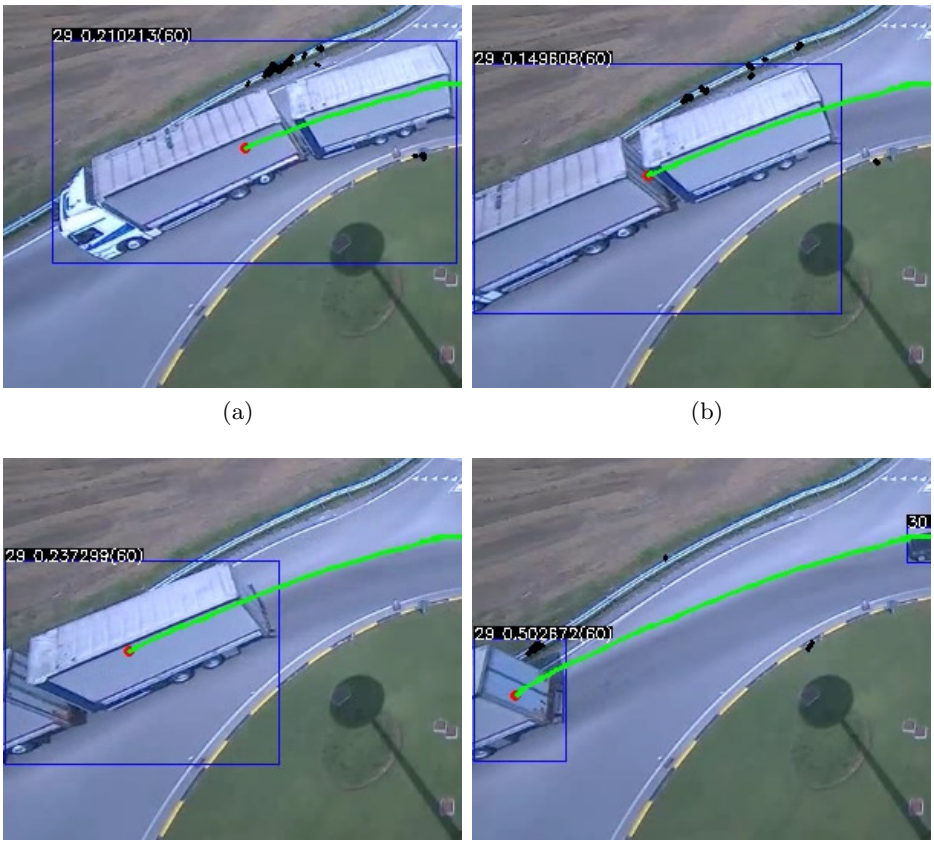


Figura 3.2: Errore di stima del centro del tracker 2D

ker suggerirebbe che la posizione del camion è in corrispondenza del vano posteriore del veicolo. Inoltre la stima errata del centro viene incorporata nel filtro di Kalman senza che esso indichi la poca confidenza da associare a questa misura.

Il tracking 3D model-based permette di superare questi problemi. Tali sistemi, come visto nel capitolo precedente, utilizzano un modello del veicolo, nel nostro caso un parallelepipedo. L'utilizzo di un modello 3D permette di rendere coerente le stime delle pose con il comportamento reale del veicolo. Il modello fornisce una descrizione della forma e della dimensione del veicolo più precisa rispetto al baricentro utilizzato dal tracker 2D. Inoltre, il tracker 2D, utilizza il centro del bounding box come stima del baricentro del veicolo. In questo passaggio vengono utilizzate due approssimazioni. Il centro del bounding box abbiamo visto che è solo una stima, a volte grossolana, del reale baricentro del veicolo. Ma anche il blob da cui questo punto viene calcolato è una stima della regione occupata dal veicolo. Il tracking 3D che proponiamo confronta la pose stimata assunta dal modello nel mondo direttamente con i blob: in questo modo eliminiamo la prima tra le due approssimazioni, riducendo in modo significativo i problemi visti in questo paragrafo.

Inoltre, il tracking nel caso multicamera presenta problemi di stima quando i veicoli si trovano nello stesso istante in entrambe le telecamere. Se una macchina, ad un certo istante, è presente in un punto della scena di una



(a) (b)

(c) (d)

Figura 3.3: La sequenza mette in evidenza il problema del tracking 2D nei pressi dei bordi.

telecamera, può capitare che sia presente anche sull'immagine ripresa da un'altra telecamera. In questo caso il tracker 2D può solo cercare di fondere le traiettorie o i blob, ad esempio sovrapponendo i frame e facendo una media dei centri. Ragionando in 3D invece si può proiettare il modello nell'immagine dell'una e dell'altra telecamera, pesando quindi indipendentemente la probabilità di una certa pose rispetto alle due misure.

3.3 Soluzioni proposte per il tracking 3D

Dall'analisi svolta studiando i lavori presenti in letteratura abbiamo notato che la maggior parte dei sistemi esistenti utilizzano il filtro di Kalman in una delle sue declinazioni per eseguire tracking bayesiano 3D. La gran parte di questi sistemi sono model-based, quindi utilizzano un modello degli oggetti che viene confrontato in diversi modi con le misure disponibili.

I nostri algoritmi sono model-based e, ricalcando la classificazione fatta nel capitolo precedente, sono region-based, ovvero, la bontà di una pose viene calcolata confrontando la regione corrispondente all'area del veicolo con la proiezione del modello dal mondo 3D al piano immagine. Utilizzare un metodo region-based ci permette di dare una valutazione di una pose semplicemente confrontando il blob con la proiezione del modello del veicolo, collocato in accordo con la pose stimata. Questo confronto ci sembra il metodo più naturale per dare una valutazione il più conforme possibile con la misura del veicolo che è esattamente il blob estratto dalla background subtraction.

A differenza di molti sistemi di tracking, i due sistemi proposti adottano un approccio Montecarlo, piuttosto che alla Kalman. Il motivo di questa scelta è conseguenza diretta di quanto appena affermato. Per utilizzare il filtro di Kalman bisogna rendere esplicita l'equazione $\mathbf{z}_t = h(\mathbf{s}_t, \mathbf{v}_t)$ del sistema (2.14), in una forma del tipo $z_t = Hs_t + v_t$. La matrice H descrive come lo stato e la misura sono legate tra loro. Per utilizzare il filtro di Kalman sarebbe necessario descrivere in modo analitico questo legame. Lavorando con i centroidi dei veicoli questo compito è semplice: il valore di z_t, ovvero la misura, è un punto nello spazio 2D ed è quindi semplice trovare una matrice che la leghi alla posizione. Nel nostro caso, però vogliamo confrontare direttamente lo stato con il blob per i motivi appena illustrati. Inoltre, la f dell'equazione $s_t = f(s_{t-1}, w_t)$ del sistema (2.14) per il filtro di Kalman è lineare. La funzione f modella il moto del veicolo che nella realtà non è lineare, dato che sono coinvolte delle rotazioni.

Per risolvere entrambe questi problemi abbiamo scelto un approccio Montecarlo.

3.3.1 Il campionamento Montecarlo

Il campionamento Montecarlo ha lo scopo di stimare una generica densità di probabilità. Abbiamo visto nel Paragrafo 2.2.3 che fare tracking bayesiano coincide spesso con stimare per ogni istante la distribuzione di probabilità dello stato dell'oggetto inseguito. Una delle tecniche che può essere utilizzata per stimare una generica densità di probabilità è nota come campionamento Montecarlo.

Consideriamo una generica densità di probabilità π e N campioni indipendenti $X_i \sim \pi$ per $i = 1, 2, \ldots, N$. Denotando con δ_{X_i} la funzione delta di Dirac centrata in X_i, allora la stima π é:

$$\hat{\pi} = \frac{1}{N} \sum_{i=1}^{N} \delta_{X_i} \tag{3.1}$$

Questo metodo offre una stima non polarizzata, ovvero la media di $\hat{\pi} - \pi$ è nulla con $N \to \infty$ e la cui varianza diminuisce linearmente con N [15]. Questo approccio permette di superare i limiti che il filtro di Kalman evidenzia per il tracking 3D region-based.

Partendo da una distribuzione opportuna, generiamo dei campioni della pose. Ognuno di questi può essere pesato associando un semplice valore numerico che è tanto maggiore quanto è migliore il confronto tra il blob e la proiezione del modello posizionato secondo la pose stessa. Otteniamo in questo modo un'indicazione di quali tra i campioni estratti rappresentano meglio al pose del veicolo, utilizzando il blob come misura per confrontare la stima.

Inoltre, abbiamo visto che nel filtro di Kalman la relazione che modella il moto tra due stati in istanti successivi è la funzione lineare f. Utilizzando un approccio Montecarlo in cui i campioni rappresentano le pose del veicolo, possiamo esprimere la relazione tra stati successivi come una distribuzione di densità qualsiasi.

3.4 Tracking con Viterbi

L'algoritmo basato su Viterbi nasce da un'evoluzione dell'algoritmo presentato in [58] e spiegato in dettaglio nel Paragrafo 2.6.2. Data l'originalità di questo sistema abbiamo deciso di implementarne un'evoluzione alla luce dei dati 2D.

In primo luogo, il nostro algoritmo, a differenza di quello di Song e Nevatia propone l'utilizzo di un modello dei veicoli, sempre a forma di parallelepipedo, ma le cui dimensioni sono variabili. In altre parole, realizziamo il tracking utilizzando in contemporanea modelli a forma di parallelepipedo di differenti dimensioni: l'algoritmo sceglierà poi quale tra questi modelli è il più probabile. Questo permette di modellare in modo più preciso diversi

tipi di veicoli. Un modello a dimensione fissa può causare una stima molto imprecisa della posizione dei veicoli. Per esempio, in Figura 3.4 si vedono tre situazioni che si potrebbero ottenere utilizzando un modello a dimensione fissa. I pixel bianchi rappresentano il blob del veicolo mentre in rosso è disegnata la proiezione del modello sull'immagine. In tutti questi casi la probabilità che definiremo nel prossimo capitolo risulta approssimativamente la medesima, quindi il sistema non peserebbe in modo corretto le pose chiaramente errate senza distinguerle da quelle corrette.

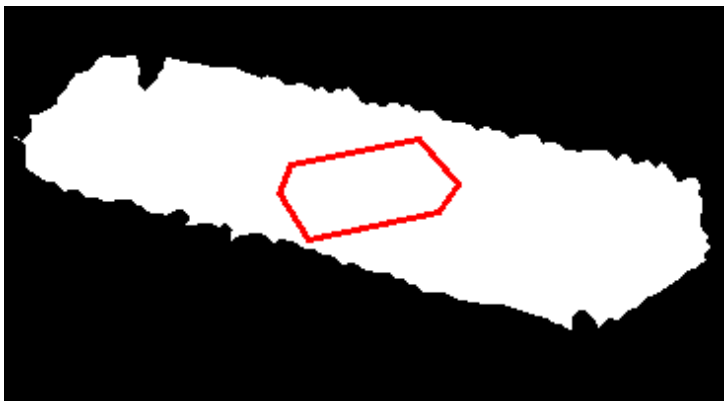
L'utilizzo del modello a dimensioni variabili offre anche una stima delle dimensioni reali del veicolo. Da questa stima il nostro sistema può inferire la classe a cui appartiene il veicolo. Nel Paragrafo 4.2.2 descriviamo in modo dettagliato come scegliamo le dimensioni dei modelli e di conseguenza come riusciamo a fare classificazione.

Rispetto a questo algoritmo trascuriamo le ombre mentre gli autori Song e Nevatia preferiscono proiettare il modello del veicolo sul terreno in base alla direzione del sole e considerare anche questa per valutare la probabilità dello stato. Nelle soluzioni proposte non c'è nessun ragionamento sull'ombra dato che in fase di background subtraction si sono ottenuti buone approssimazioni dei veicoli senza ombre. É comunque possibile che in futuro venga integrato nel sistema un metodo di gestione delle ombre.

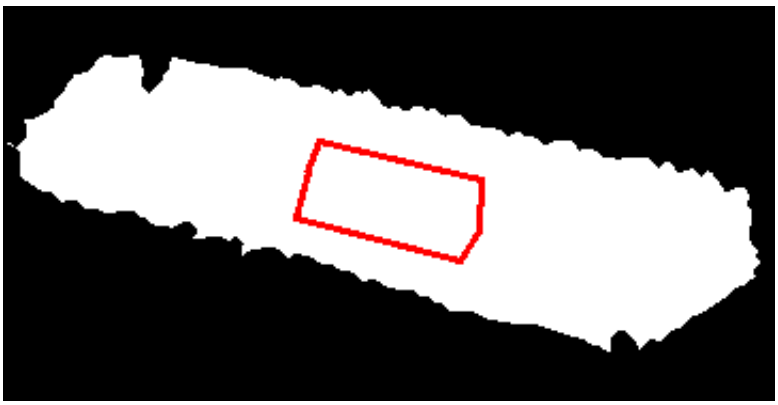
Il sistema Song-Nevatia lavora direttamente sul risultato della background subtraction e dell'optical flow. Nel nostro caso, invece, conosciamo le posizioni assunte dal veicolo nel piano immagine a seguito dell'esecuzione del tracker 2D (Paragrafo 3.2). Questo ci permette di conoscere come il veicolo si muove nel tempo rendendo possibile quindi evitare il calcolo dell'oneroso optical flow eseguito in [58]. Inoltre la traiettoria 2D è riferita ad un singolo veicolo; nel nostro sistema, non è dunque più necessario considerare contemporaneamente la configurazione di tutti i veicoli come avviene invece in [58]. Ogni singolo veicolo viene analizzato singolarmente, partendo dalle posizioni che il tracker 2D estrae dal filmato. Lo stato θ_t che Song e Nevatia utilizzano per indicare le pose dei veicoli al tempo t diventa quindi la pose del singolo veicolo. La distribuzione da cui campionare la pose del singolo veicolo si può ipotizzare come una Gaussiana trivariata centrata nella posizione e con l'orientamento 3D inferiti dai dati 2D [1]. Cade quindi la necessità di utilizzare l'algoritmo Metropolis-Hastings, che è necessario solo con distribuzioni non banali.

L'algoritmo proposto estrae i campioni della pose del singolo veicolo per ogni frame, e utilizza l'algoritmo di Viterbi in modo analogo a quanto fatto da Song e Nevatia per scegliere la sequenza di pose. A questo proposito è utile introdurre gli Hidden Markov Model e, quindi l'algoritmo di Viterbi.

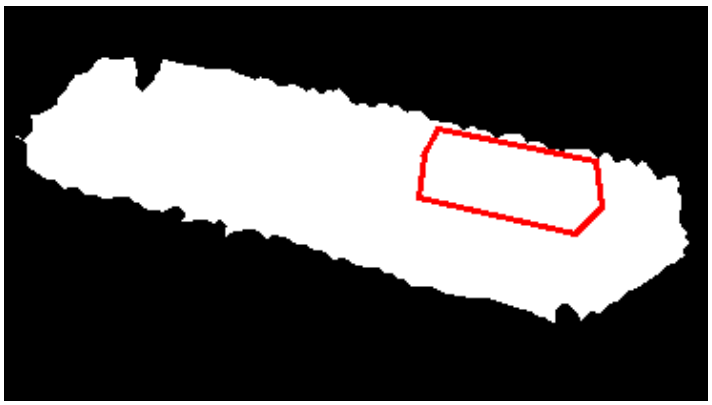
[1] Ricordiamo che gli algoritmi di tracking di veicoli adottano il Ground Plane Constraint quindi, un veicolo nel mondo viene individuato da tre componenti: due coordinate x e y definiscono la posizione nel piano della strada, e θ definisce l'orientamento del veicolo.



(a)



(b)



(c)

Figura 3.4: Errori causati dall'utilizzo di un modello a dimensioni fisse.

3.4.1 Gli Hidden Markov Model

Per parlare di Hidden Markov Model è opportuno introdurre prima i *processi stocastici*, in particolar modo i *processi markoviani*. Un processo stocastico è una sequenza di variabili aleatorie ordinata nel tempo. Una variabile casuale s dipende dall'esito e di un esperimento e dal tempo t, quindi si indica con $s(e,t)$. Generalmente la dipendenza dall'esito viene omessa, quindi viene indicata con $s(t)$ oppure s_t [5]. Quando il tempo è discreto e per ogni istante di tempo t vale

$$p(s_{t+1}|s_t, s_{t-1}, \ldots, s_1) = p(s_{t+1}|s_t) \tag{3.2}$$

si parla di processo Markoviano. Un processo Markoviano ha dunque la proprietà che il valore di ogni variabile dipende solo dal valore assunto nell'istante precedente. La variabile casuale S può essere vista come lo stato che rappresenta un fenomeno nel mondo. Nel caso del tracking questa variabile coincide con la pose del veicolo.

Nel caso in esame non è possibile ricavare direttamente dalle immagini la pose del veicolo a ogni istante; in tutti i casi in cui la variabile S non può essere osservata direttamente, si può descrivere il problema nell'ambito degli Hidden Markov Model. Un Hidden Markov Model (HMM) è una quintupla $< S, Z, P, A, B >$ in cui:

- $S = s_1, \ldots, s_N$ sono i valori degli stati nascosti, che non possono essere osservati direttamente;
- $Z = z_1, \ldots, z_T$ è la sequenza delle osservazioni per ogni istante;
- P è la distribuzione di probabilità dello stato iniziale;
- A è la matrice che descrive la probabilità che da uno stato si passi a un altro, $p(s_{t+1}|s_t)$;
- B è la matrice che descrive la probabilità che dato uno stato si abbia una certa osservazione, $p(z_{t+1}|s_{t+1})$.

In Figura 3.5 è rappresentato un HMM che da l'idea della situazione con cui abbiamo a che fare: le frecce indicano le dipendenze. Ogni stato dipende solo dallo stato precedente. Ogni osservazione dipende solo dallo stato che ha generato l'osservazione. Possiamo riassumere tutto ciò con l'Equazione 3.2 e con:

$$p(z_{t+1}|z_{1:t}, s_{1:t}) = p(z_{t+1}|s_{t+1}, s_t). \tag{3.3}$$

Nel nostro caso gli stati S rappresentano le pose occupate dal veicolo ad ogni istante di tempo. L'osservazione z_t è il foreground dell'immagine al tempo t; più precisamente, a seconda del caso è il centro o il blob estratto dal tracker 2D.

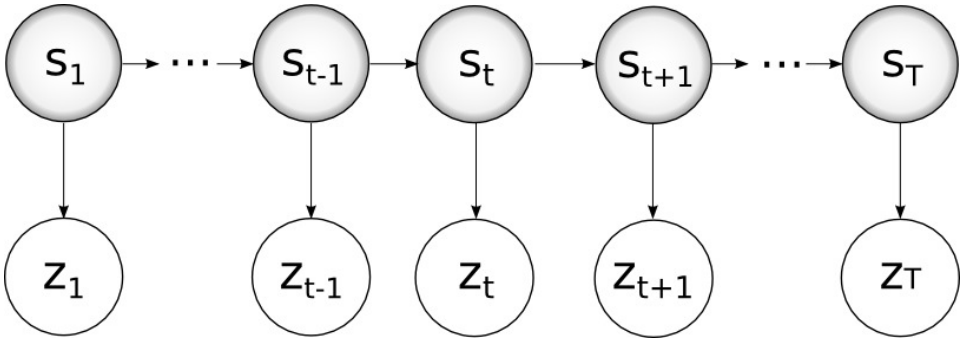


Figura 3.5: Rappresentazione dell'Hidden Markov Model

3.4.2 L'algoritmo di Viterbi

Consideriamo un Hidden Markov Model in cui $s_{1:T}$ e $z_{1:T}$ rappresentano rispettivamente gli stati nascosti e le osservazioni dall'istante 1 all'istante T. L'algoritmo di Viterbi [19] intende stimare la sequenza $\bar{s}_{1:T}$ che massimizza la probabilità a posteriori $p(s_{1:T}|z_{1:T})$:

$$\begin{aligned} \bar{s}_{1:T} &= \arg\max_{s_{1:T}} \{p(s_{1:T}|z_{1:T})\} = \\ &= \arg\max_{s_{1:T}} \left\{ \frac{p(s_{1:T}, z_{1:T})}{p(z_{1:T})} \right\} = \\ &= \arg\max_{s_{1:T}} \{p(s_{1:T}, z_{1:T})\} \end{aligned} \tag{3.4}$$

dato che $p(z_{1:T})$ è costante. Dalla probabilità congiunta $p(s_{1:T}, z_{1:T})$ si ricava la formula iterativa utilizzata dall'algoritmo di Viterbi per selezionare la sequenza $s_{1:T}$ di stati più opportuna:

$$\begin{aligned} p(s_{1:T}, z_{1:T}) &= p(z_{1:T}|s_{1:T}) \cdot p(s_{1:T}) = \\ &= \prod_{t=1}^{T} \{p(s_{t+1}|s_t) \cdot p(z_{t+1}|s_{t+1}, s_t)\}, \end{aligned} \tag{3.5}$$

dove $p(s_{t+1}|s_t)$ è detta probabilità a priori e $p(z_{t+1}|s_{t+1}, s_t)$ verosimiglianza. La soluzione $\bar{s}$ può essere riscritta passando ai logaritmi come:

$$\begin{aligned} \bar{s}_{1:T} &= \arg\min_{s_{1:T}} \{-\log(p(s_{1:T}|z_{1:T}))\} = \\ &= \arg\min_{s_{1:T}} \sum_{t=1}^{T} \{-\log(p(s_{t+1}|s_t)) - \log(p(z_{t+1}|s_{t+1}, s_t))\} \end{aligned}. \tag{3.6}$$

In questo modo è possibile vedere il problema come ricerca di un cammino minimo in un grafo in cui per ogni istante tutti i possibili stati , rappresentati

dai nodi sono collegati ai successivi tramite un arco il cui peso è dato dal valore di $-\log(p(s_{t+1}|s_t)) - \log(p(z_{t+1}|s_{t+1}, s_t))$.

Nel sistema proposto gli stati a ogni istante sono i campioni della pose estratti nella prima fase che chiamiamo. Gli archi sono invece pesati calcolando la verosimiglianza dei campioni rispetto alle immagini, e calcolando il valore di associazione tra un campione e il successivo. In Figura 3.6 mostriamo il grafo creato dove s_t^i è l'i-esimo campione estratto al tempo t: la soluzione dell'algoritmo di Viterbi è il cammino minimo per arrivare dal nodo *start* al nodo *end*.

In riferimento a quanto detto nel Paragrafo 2.6.2 si noti come la forma dell'equazione 2.29 sia analoga a quella ricavata in 3.5. In Song-Nevatia si ha però l'utilizzo del termine a posteriori al posto del termine a priori che si trova nell'equazione di Viterbi; l'algoritmo classico è quello presentato in questo paragrafo. É da notare come una proprietà importante dell'algoritmo di Viterbi è l'ottimalità: la formula risolutiva (3.6), detta anche *equazione dinamica*, è tale che per definizione massimizza il termine $p(s_{1:T}|z_{1:T})$.

Alla fine di questa trattazione si potrebbe credere che l'utilizzo della proprietà di markovianità faccia cadere l'idea di smoothing, dato che si presuppone che lo stato corrente dipenda solo dallo stato precedente e dalla misura corrente. Nonostante ciò possiamo ancora parlare di smoothing perché il risultato della ricerca del cammino di costo minimo è un percorso che minimizza il costo globalmente e non solo a livello locale.

3.4.3 Algoritmo di Dijkstra

La soluzione dell'algoritmo di Viterbi è il cammino di costo minimo tra uno stato iniziale s, da cui parte un arco di peso 0 per ogni nodo corrispondente a un campione del primo frame, e uno stato finale e raggiunto da ogni campione dell'ultimo frame tramite un arco di peso 0.

Dividiamo gli archi in tre insiemi:

- insieme $\mathbb{I}$: archi assegnati all'albero costruito;
- insieme $\mathbb{II}$: archi tra cui viene selezionato il prossimo da aggiungere all'insieme $\mathbb{I}$;
- insieme $\mathbb{III}$: archi rimanenti.

e i nodi in due insiemi:

- insieme $\mathbb{A}$: nodi connessi all'albero;
- insieme $\mathbb{B}$: nodi rimanenti.

Di seguito si suppone che un arco da N_1 a N_2 abbia una lunghezza pari al costo impiegato per arrivare ad N_1 sommato al costo per arrivare da N_1 a N_2.

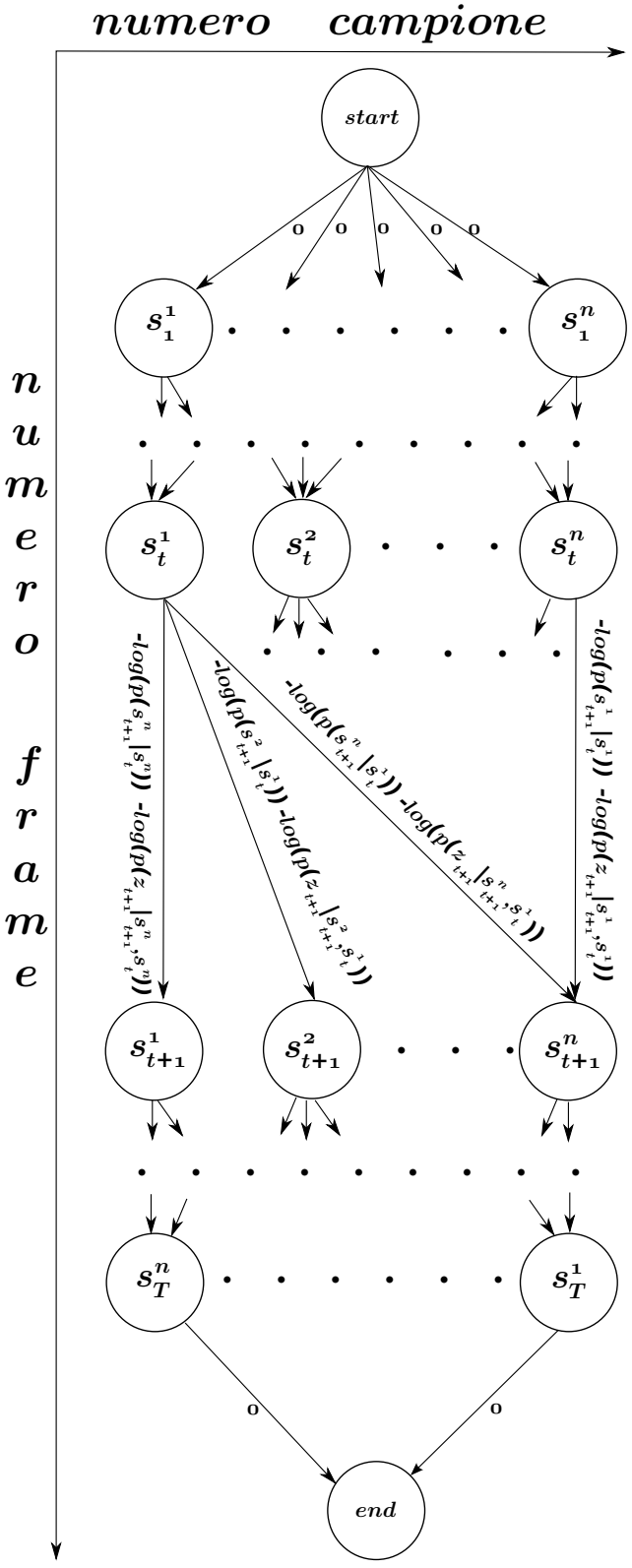


Figura 3.6: Grafo su cui ricerchiamo il cammino minimo.

La ricerca del cammino minimo comincia scegliendo un nodo, nel nostro caso il nodo fittizio s. L'insieme $\mathbb{A}$ contiene solo questo nodo, i rimanenti sono in $\mathbb{B}$. L'insieme $\mathbb{I}$ è vuoto. Tutti gli archi che finiscono in questo nodo scelto devono essere messi in $\mathbb{II}$, che nel nostro caso è vuoto. Gli archi rimanenti popolano $\mathbb{III}$. Se l'insieme $\mathbb{II}$ è vuoto si parte dal secondo punto, altrimenti dal primo, eseguendo iterativamente i due passi fino a che non aggiungiamo e all'insieme $\mathbb{A}$:

- consideriamo l'arco in $\mathbb{II}$ di lunghezza minore, lo rimuoviamo da $\mathbb{II}$e lo aggiungiamo in $\mathbb{I}$. Spostiamo il nodo raggiunto dall'arco da $\mathbb{B}$ ad $\mathbb{A}$.
- consideriamo gli archi che collegano l'ultimo nodo aggiunto N_{last} ad $\mathbb{A}$ a nodi dell'insieme $\mathbb{B}$. Per ogni nodo N_R raggiunto, quindi per ogni arco A_{cur} da N_{last} a N_R:
  - se non esistono altri archi dell'albero entranti in N_R, aggiungiamo A_{cur} in $\mathbb{II}$;
  - se esiste un altro arco dell'albero entrante in N_R , ed è più corto di A_{cur} rifiuto A_{cur} e lo metto in in $\mathbb{III}$, altrimenti rimpiazziamo con A_{cur} l'arco in $\mathbb{II}$.

Questo algoritmo permette di non valutare la lunghezza di alcuni archi, e questo di conseguenza fa in modo che si raggiunga la soluzione in un tempo inferiore rispetto alla ricerca esaustiva.

Lo stesso algoritmo si può vedere ragionando sui nodi più che su gli archi (Algoritmo 1). Questa è l'implementazione usuale ed è quella utilizzata nel nostro sistema: piuttosto che associare la lunghezza agli archi, si pesano i nodi in base a quanto distano dal nodo sorgente, ovvero il nostro s, e viene memorizzato un puntatore al nodo precedente tale da permettre di risalire al nodo s a partire dall'ultimo nodo scelto, lungo il cammino di costo minimo. Per il resto l'algoritmo funziona esattamente come quello presentato originariamente da Dijkstra.

3.4.4 L'algoritmo proposto

L'algoritmo basato su Viterbi è strutturato in due fasi analogamente a [58]: il campionamento frame per frame e la ricerca della traiettoria del veicolo attraverso l'algoritmo di Viterbi. Questi passi vengono eseguiti per ogni modello del veicolo generando quindi una sequenza di pose associata a ogni modello. Solo quando il sistema ha completato i due passi sceglie la traiettoria 3D più probabile: il modello associato a questa traiettoria è quello che rappresenta nel modo migliore il veicolo inseguito. L'Algoritmo 2 rappresenta in pseudo-codice le operazioni svolte per processare i risultati del tracker 2D e ricostruire la traiettoria 3D del un singolo veicolo. La gestione di più traiettorie è una naturale estensione di questo algoritmo che presentaimo nel prossimo capitolo. Il numero n dei campioni nel nostro caso è 500.

Algoritmo 1: Algoritmo di Dijkstra

Input: $\mathbb{G} = \{\mathbb{N}, \mathbb{A}\}$ grafo con nodi N e archi A, s nodo iniziale, e nodo finale

Output: *nodoPrecedente*: vettore dei puntatori che associano un nodo al precedente migliore, nel senso del cammino minimo

Inizializzazione;
foreach n *in* $\mathbb{N}$ **do**
    $lunghezza[n] = infinito$;
    $nodoPrecedente[n] = indefinito$;
$lunghezza[s] = 0$;
$u = s$;
$\mathbb{U} = \mathbb{N}$;
Ciclo Principale;
while $u \neq e$ **do**
    rimuovi u da $\mathbb{U}$;
    foreach *nodo n raggiungibile da u tramite un arco in* $\mathbb{A}$ **do**
        $lunghezzaCorrente = lunghezza[u] + distanza(u, n)$;
        if $lunghezzaCorrente < lunghezza[n]$ **then**
            $lunghezza[n] = lunghezzaCorrente$;
            $nodoPrecedente[n] = u$;
    $u = \min_{elem} \mathbb{U}$;

Le funzioni di verosimiglianza e del termine a priori verranno descritte nel prossimo capitolo.

Campionamento frame per frame Nella prima fase generiamo i campioni tra cui l'algoritmo di Viterbi sceglie quelli che compongono la sequenza che stima in modo migliore la traiettoria compiuta dal veicolo. Per traiettoria intendiamo l'insieme delle pose che il veicolo occupa ad ogni istante. Per ogni veicolo, e per ogni frame conosciamo dal tracker 2D la stima della posizione sul piano immagine. Proiettando sul piano della rotonda questo punto otteniamo una prima stima della posizione del veicolo nel mondo. L'orientamento è stimato come il vettore che va da questo punto alla proiezione sul piano della rotonda della stima della posizione del veicolo all'istante successivo. Questa posizione e questo orientamento sono una stima della pose del veicolo. Il passo di proiezione dal 2D al 3D viene presentato nel Paragrafo 4.2.2: per ora è sufficiente dire che questo passo è dipendente dalla dimensione del modello scelto, quindi nell'algoritmo viene indicato come $proietta2D3D(p_t, m)$ dove p_t è il punto della traiettoria all'istante t. Successivamente estraiamo i campioni della pose campionando da una Gaussiana trivariata con media nella pose stimata sul piano della rotonda. Ognuno di questi campioni rappresenta un possibile stato che può assumere il veicolo. Per ognuno di essi calcoliamo la funzione $p(z_{t+1}|s_{t+1}, s_t)$ di verosimiglianza.

Scelta della miglior sequenza con Viterbi Una volta che sono noti tutti i campioni per ogni istante di tempo utilizziamo l'algoritmo di Viterbi per ricercare il la sequenza di campioni che meglio rappresenta il moto del veicolo. Con i campioni estratti per ogni istante possiamo creare un grafo in cui i nodi sono gli stati-campioni e gli archi associano campioni successivi. Il costo di un arco tra uno stato s_t e s_{t+1} è $-\log(p(s_{t+1}|s_t)) - \log(p(z_{t+1}|s_{t+1}, s_t))$, dove $p(z_{t+1}|s_{t+1}, s_t)$ è la verosimiglianza calcolata nel passo precedente, $p(z_{t+1}|s_{t+1}, s_t)$ è il termine a priori che calcoliamo in in questa fase. La rappresentazione a grafo ci permette di utilizzare l'algoritmo di Dijkstra come abbiamo visto in precedenza. Solo ogni volta che l'algoritmo deve valutare il costo di un arco, viene calcolato il termine a priori. In questo modo, se l'algoritmo non espande un nodo e, di conseguenza, non ne valuta gli archi uscenti, evitiamo di calcolare il termine a priori.

3.5 Tracking con particle smoother

Durante l'implementazione dell'algoritmo appena presentato ci siamo resi conto che questo sistema soffre di un problema: estraendo i campioni istante per istante, in maniera indipendente per ogni frame, dobbiamo utilizzarne un gran numero per generare delle pose sufficientemente rappresentative della pose reale del veicolo. Utilizzando le informazioni sullo stato precedente,

Algoritmo 2: Algoritmo basato su Viterbi

Input: $puntiTraiettoria$: sequenza di posizioni 2D del veicolo sul piano immagine
Input: $\{b_1, \ldots, b_T\} \in blobVeicolo$: sequenza dei blob
Input: $modelDimension$: insieme di triple che definiscono differenti dimensioni del parallelepipedo
Output: $pose3DVeicolo$: sequenza di pose 3D occupate dal veicolo

Campionamento frame per frame;
foreach p_t *in* $puntiTraiettoria$ **do**
    foreach m *in* $modelDimension$ **do**
        $(x_t^{3D}, y_t^{3D}) = proietta2D3D(p_t, m)$;
        $(x_{t+1}^{3D}, y_{t+1}^{3D}) = proietta2D3D(p_{t+1}, m)$;
        $\theta_t = orientamentoVettore((x_t^{3D}, y_t^{3D}), (x_{t+1}^{3D}, y_{t+1}^{3D}))$;
        $\mathbf{s}_{(t,m)} =$
        $\{n$ campioni dalla Gaussiana centrata in $(x_t^{3D}, y_t^{3D}, \theta_t)\}$;
        foreach s_i *in* $\mathbf{s}_{(t,m)}$ **do**
            $verCamp_{(s_i,t,m)} = calcolaVerosimiglianza(s_i, b_t, m)$;
Scelta della miglior sequenza con Viterbi;
foreach m *in* $modelDimension$ **do**
    $[pose3DVeicoloM(m), probCammino(m)] =$
    $Dijkstra2(\{\mathbf{s}_{1,m}, \ldots, \mathbf{s}_{T,m}\}, \{verCamp_{(1,1,m)}, \ldots, verCamp_{(n,T,m)}\})$;
Sceglie il modello migliore;
$M_{best} = \arg\max_m \{probCammino(m)\}$;
$pose3DVeicolo = pose3DVeicoloM(M_{best})$;

Algoritmo 3: Versione dell'algoritmo di Dijkstra utilizzata nel nostro algoritmo (Dijkstra2)

Input: $\{\mathbf{s}_1, \ldots, \mathbf{s}_T\}$: sequenza si tutti i campioni estratti nel passo precedente

Input: $\{verCamp_{(1,1)}, \ldots, verCamp_{(n,T)}\}$: valori della funzione di verosimiglianza associati ai campioni

Output: $sequenzaMigliore, probCammino$: sequenza dei campioni migliori nel tempo e probabilità del cammino

Inizializzazione;
for $t = 1$ **to** T **do**
    foreach s *in* $\mathbf{s}_t$ **do**
        $lunghezza[s,t] = infinito$;
        $nodoPrecedente[s,t] = indefinito$;
        $nodiVisitati[s,t] = 0$;
Nodo start (indice $(s,t) = (0,0)$);
$(s_{cur}, t_{cur}) = (0,0)$;
$lunghezza[s_{cur}, t_{cur}] = 0$;
Ciclo Principale;
while $t_{cur} \neq T$ **do**
    $nodiVisitati[s_{cur}, t_{cur}] = 1$;
    foreach s *in* $\mathbf{s}_{t_{cur}+1}$ **do**
        $lunghezzaCorrente =$
        $lunghezza[s_{cur}, t_{cur}+1] + calcolaPriori(s, s_{cur})$;
        if $lunghezzaCorrente < lunghezza[s, t_{cur}+1]$ **then**
            $lunghezza[s, t_{cur}+1] = lunghezzaCorrente$;
            $nodoPrecedente[s, t_{cur}+1] = s_{cur}$;
    $(s_{cur}, t_{cur}) = \min_{elem}\{nodiVisitati[\cdot,\cdot] = 0\}$;
$probCammino = lunghezza[s_{cur}, T]$;
Ricostruisce il cammino ;
$ultimoNodo = s_{cur}$;
for $t = T$ **to** 1 **do**
    $sequenzaMigliore(t) = nodoPrecedente[ultimoNodo, t_{cur}+1]$;
    $ultimoNodo = sequenzaMigliore(t)$;

senza far cader quindi l'ipotesi di Markov, potremmo invece polarizzare il campionamento nel punto in cui crediamo sia più probabile che il veicolo arrivi, generando quindi i campioni che hanno più probabilità di essere vicini alla pose reale. A questo scopo abbiamo deciso di sviluppare anche un altro algoritmo che permetta però questo tipo di polarizzazione dei campioni: il particle filter, o, più precisamente il particle smoother. Sebbene utilizzato in diversi campi di tracking, per esempio in [7] e [55] nella versione a filtro viene sorprendentemente poco utilizzato nel tracking 3D, in particolare nel tracking di veicoli. Esistono, infatti, solo alcuni lavori recenti che sfruttano questo strumento per il tracking dei veicoli [70] e [54] di cui solo uno [70] offre una soluzione 3D.

Abbiamo già presentato l'algoritmo del particle filter, da cui si parte per creare lo smoother nel Paragrafo 2.2.3. Prima di presentare lo smoother vediamo però una spiegazione più dettagliata del filtro e la sua implementazione.

3.5.1 Particle Filter

Lo scopo del tracking bayesiano è la stima del valore della densità di probabilità a posteriori, $p(s_{1:t}|z_{1:t})$ indicando con $s_{1:t}$ e $z_{1:t}$ gli stati dal tempo 1 e le misure dal tempo 1 al tempo t. Il particle filter approssima questa distribuzione attraverso dei campioni $\{s^i_{1:t}, w_t\}_{i=1}^{N_s}$, dette *particelle* dove $\{s^i_{1:t}\}$ indica l'i-esimo campione il cui peso è w^i_t. Dato che i pesi rappresentano una distribuzione di probabilità, il loro valore è sempre normalizzato in modo tale che $\sum_i w^k_i = 1$. Con un numero N_s di campioni:

$$p(s_{1:t}|z_{1:t}) \approx \sum_{i=1}^{N_s} w^i_t \cdot \delta(s_{1:t} - s^i_{1:t}). \tag{3.7}$$

Non sempre è possibile generare dei campioni direttamente dalla distribuzione $p(\cdot)$. Per questo si utilizza solitamente una distribuzione $q(\cdot)$ chiamata importance distribution che approssima il comportamento della p in modo analogo a quanto accade nell'algoritmo Metropolis-Hastings con la proposal distribution (Paragrafo 2.6.1). Il peso dei campioni viene calcolato come

$$w^i_t \propto \frac{p(s_{1:t}|z_{1:t})}{q(s_{1:t}|z_{1:t})}. \tag{3.8}$$

Utilizzando opportunamente il teorema di Bayes, possiamo riscrivere $p(s_{1:t}|z_{1:t})$ come:

$$\begin{aligned} p(s_{1:t}|z_{1:t}) &= \frac{p(z_t|s_t)p(s_t|s_{t-1})}{p(z_t|z_{1:t-1})} p(s_{1:t-1}|z_{1:t-1}) \\ &\propto p(z_t|s_t)p(s_t|s_{t-1})p(s_{1:t-1}|z_{1:t-1}) \end{aligned} \tag{3.9}$$

e $q(s_{1:t}|z_{1:t})$ come:

$$q(s_{1:t}|z_{1:t}) = q(s_t|s_{1:t-1}, z_{1:t})q(s_{1:t-1}|z_{1:t}). \tag{3.10}$$

L'Equazione (3.8) può essere resa ricorsiva nel seguente modo:

$$\begin{aligned} w_t^i &\propto \frac{p(z_t|s_t)p(s_t|s_{t-1})p(s_{1:t-1}|z_{1:t-1})}{q(s_t|s_{1:t-1}, z_{1:t})q(s_{1:t-1}|z_{1:t})} \\ & w_{t-1}^i \frac{p(z_t|s_t)p(s_t|s_{t-1})}{q(s_t|s_{1:t-1}, z_{1:t})}. \end{aligned} \tag{3.11}$$

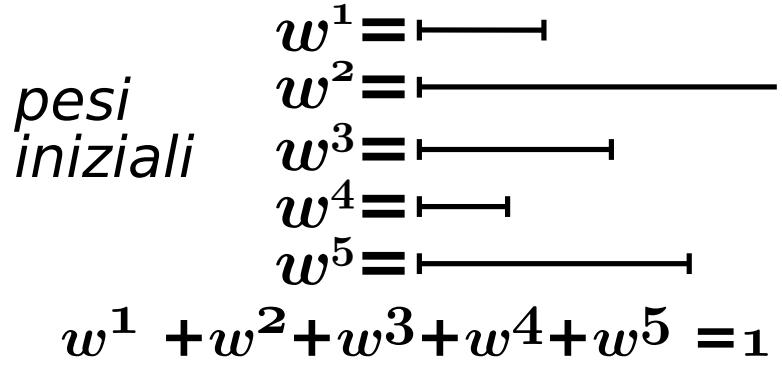
Inoltre, sotto l'ipotesi di Markov vale $q(s_t|s_{1:t-1}, z_{1:t}) = q(s_t|s_{1:t-1}, z_t)$ e possiamo riscrivere (3.11) come:

$$w_t^i \propto w_{t-1}^i \cdot \frac{p(z_t|s_t^i)p(s_t^i|s_{t-1}^i)}{q(s_t^i|s_{t-1}^i, z_t)}, \tag{3.12}$$

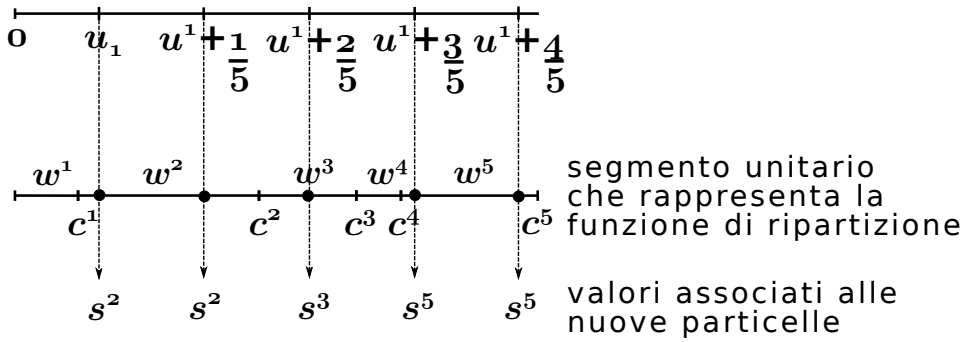
e possiamo scrivere la stima a posteriori dello stato corrente:

$$p(s_t|z_{1:t}) \approx \sum_{i=1}^{N_s} w_t^i \cdot \delta(s_t - s_t^i). \tag{3.13}$$

Il maggiore problema di cui soffre il particle filter è la degenerazione delle particelle, che avviene quando queste ultime tendono ad avere tutte pesi trascurabili a parte una. Per questo motivo si utilizza il ricampionamento [2], il quale permette di modificare la concentrazione delle particelle dove la probabilità $p(s_t|z_{1:t})$ è maggiore. Partendo da un insieme $\{s_t^i\}_{i=1}^{N_s}$ di particelle a cui è associato un peso w_t^i, lo scopo del ricampionamento è quello di ottenere un nuovo insieme di particelle $\{\bar{s}_t^j\}_{j=1}^{N_s}$ in modo tale che $P(\bar{s}_t^j = s_t^i) = w_t^i$. I pesi delle nuove particelle saranno quindi tutti uguali e pari a $\bar{w}_t^j = \frac{1}{N_s}$. Il risultato del ricampionamento è quindi di eliminare le particelle il cui peso è trascurabile, e concentrarle dove i pesi sono maggiori. Un metodo per far ciò è presentato con l'Algoritmo 4. Viene costruita una funzione di ripartizione in accordo con i pesi w_t^i dalla quale vengono estratti i nuovi campioni. Presupponendo i pesi w_t^i normalizzati, si può pensare ad un segmento di lunghezza unitaria partizionato in modo che l'i-esimo segmento abbia una lunghezza pari al peso w_t^i (secondo segmento in Figura 3.7b). L'algoritmo scorre il segmento così creato estraendo N_s punti equidistanti. Ognuno di questi punti, che rappresenta una particella, cade all'interno di un segmento corrispondente ad un peso w_t^i; il valore della nuova particella è quindi il valore della particella corrispondente al peso w_t^i. Prima di eseguire il passo di ricampionamento i pesi rappresentano le probabilità di ciascuna delle particelle; dopo il ricampionamento, tutte le particelle hanno peso uguale e pari a $w_t^j = N^{-1}$. Questo perché il passo di campionamento ha lo scopo di approssimare la distribuzione non più con i pesi ma proprio



(a) *Pesi dei campioni iniziali prima del ricampionamento.*



(b) *scelta degli stati da associare alle nuove particelle.*

Figura 3.7: Schematizzazione del ricampionamento.

Algoritmo 4: Ricampiona

Input: $\{s_t{}^i, w_t^i\}_{i=1}^{N_s}$ campioni su cui eseguire il ricampionamento
Output: $\{\bar{s}_t^j, w_t^j\}_{j=1}^{N_s}$ nuovi campioni ricampionati dagli input

crea la funzione di ripartizione CDF da cui campionare;
inizializza CDF: $c_1 = 0$;
for $i = 2$ **to** N_s **do**
  $c_i = c_{i-1} + w_t^i$;
comincia dall'inizio della CDF: $i = 1$;
campiona il primo punto: $u_1 \sim \mathbb{U}[0, N_s^{-1}]$;
for $j = 1$ **to** N_s **do**
  scorri lungo la CDF: $u_j = u_1 + N_s^{-1}(j-1)$;
  while $u_j > c_i$ **do**
    $i = i + 1$;
  $\bar{s}_t^j = s_t{}^i$;
  $w_t^j = N^{-1}$;
$\{s_{1:t}{}^i, w_t\}_{i=1}^{N_s} = \text{ricampiona}(\{s_{1:t}{}^i, w_t\}_{i=1}^{N_s})$;

attraverso una redistribuzione delle particelle coerente con i pesi in ingresso alla funzione.

Il particle filter che utilizziamo per l'algoritmo di tracking è detto *Sampling Importance Resampling* (SIR). Questa versione sceglie come importance density il termine a priori, ovvero $q(s_t|s_{t-1}^i, z_t) = p(s_t|s_{t-1}^i)$, e utilizza il ricampionamento ad ogni iterazione (Algoritmo 5).

Algoritmo 5: SIR Particle Filter

Input: $\{{s_{1:t}}^i, {w_t}^i\}_{i=1}^{N_s}$
particelle e pesi associati all'istante t **Output**: $\{{s_{1:t+1}}^i, w_{t+1}^i\}_{i=1}^{N_s}, z_t$
particelle e pesi associati all'istante $t+1$

for $i = 1$ **to** N_s **do**
  campiona $sx_t^i \sim p(s_t^i|s_{t-1}^i)$;
  calcola $w_t^i = p(z_t|s_t^i)$;
for $i = 1$ **to** N_s **do**
  normalizza w_t^i;
$\{{s_{1:t}}^i, w_t\}_{i=1}^{N_s} = ricampiona(\{{s_t}^i, w_t^i\}_{i=1}^{N_s})$;

3.5.2 Particle Smoother

A partire dal particle filter presentato, sviluppiamo un metodo di smoothing che tenga conto delle misure $z_{1:T}$ per ogni istante.

L'idea sulla quale si basa l'algoritmo è che, partendo dalla densità $p(s_{t+1}|s_t)$, possiamo facilmente costruire in modo speculare la densità di probabilità $p(s_t|s_{t+1})$. La probabilità $p(s_{t+1}|s_t)$ rappresenta il modello del moto dei veicoli: dire che il veicolo passa da uno stato s_t a uno stato s_{t+1} con alta probabilità significa infatti che il movimento per passare dal primo al secondo è sufficientemente realistico, viceversa, una bassa probabilità indica uno spostamento che un veicolo difficilmente può compiere. Possiamo quindi considerare la densità $p(s_t|s_{t+1})$ uguale a $p(s_{t+1}|s_t)$ a meno di un segno negativo nel termine dell'orientamento. In altre parole, il moto inverso da s_{t+1} a s_t viene modellato cambiando il segno all'orientamento della pose in s_{t+1} e utilizzando la stessa densità del moto diretto.

Tenendo conto di questa osservazione, il nostro algoritmo, per ogni dimensione del modello a forma di parallelepipedo, esegue lo smoothing tramite due particle filter che agiscono in serie. Inizializziamo il primo filtro partendo dal primo punto della traiettoria 2D generando campioni come per il passo di campionamento per il singolo frame visto per l'algoritmo basato su Viterbi. Il risultato di questa fase sono un numero n di campioni per ogni dimensione del modello dei veicoli e la verosimiglianza associata. A partire da essi, eseguiamo l'Algoritmo 5 per ogni istante di tempo $2 < t \leq T$.

L'algoritmo di ricampionamento che utilizziamo è una versione leggermente modificata dell'Algoritmo 4. Per creare la funzione di ripartizione da cui ricampionare, non utilizziamo tutte le particelle campionate, ma scegliamo solo quelle che hanno probabilità maggiore. Scegliamo una particella j con peso w_t^j solo se $w_t^j > \mu w_t^{\text{medio}}$ dove w_t^{medio} è il valore di probabilità medio delle particelle al tempo t. Deve valere $\mu < 1$ per aver la certezza di mantenere almeno una particella, nel caso pessimo in cui tutte abbiano un valore di probabilità pari al valore medio; tanto più μ è grande, quanto meno particelle si considerano. Questo ci permette di concentrare le particelle attorno al punto in cui è maggiore la verosimiglianza come in [30]. Questo filtro esegue quella che chiamiamo la ricorsione in avanti.

Raggiunta la fine della traiettoria al tempo T, utilizziamo le ultime particelle risultanti come punto di partenza per inizializzare il secondo filtro che insegue il veicolo facendo regredire il tempo. Per ogni j-esima particella s_T^j, creiamo una nuova particella $\bar{s}_1^j$ con la medesima posizione e orientamento cambiato di segno. Da queste particelle viene eseguito l'algoritmo di particle filter analogo a quello appena visto, ma utilizzando le immagine dal tempo T a 1. Le medie delle posizioni delle particelle e le medie dell'orientamento corretto per ogni istante sono il risultato dell'algoritmo di tracking, ovvero la sequenza di pose assunte dal veicolo nel mondo. Quest'ultima operazione di tracking con particle filter, che chiamiamo ricorsione all'indietro, non sarebbe possibile se avessimo ad un istante t solo le misure degli istanti precedenti. Mostriamo nell'Algoritmo 6 lo pseudocodice del particle smoother appena descritto.

Sia per l'algoritmo basato su Viterbi che per l'algoritmo appena presentato, vedremo nel prossimo capitolo una descrizione dettagliata dei moduli implementati, e vedremo anche che l'implementazione proposta differisce leggermente dagli algoritmo presentati in questo capitolo, perchè per questioni di efficienza abbiamo utilizzato delle euristiche per anticipare al scelta delle dimensioni del modello e per gestire il problema del tracking ai bordi dell'immagine descritto nel Paragrafo 3.2.

3.6 Confronto tra approccio alla Kalman, con Viterbi e filtro a particelle

Concludiamo il capitolo con un riassunto delle caratteristiche teoriche salienti dei metodi scelti confrontando l'approccio alla Kalman, il particle smoother e l'algoritmo di Viterbi.

Il filtro e lo smoother di Kalman, ricercano la soluzione del problema di tracking bayesiano a patto di avere un modello del sistema in questa forma:

$$\begin{cases} x_t &= F_t x_{t-1} + B_t u_t + w_t \\ z_t &= H_t x_t + v_t \end{cases} \tag{3.14}$$

Algoritmo 6: Algoritmo di Particle Smoother proposto

Input: $\{p_1, \ldots, p_T\} \in puntiTraiettoria$: sequenza di posizioni 2D del veicolo sul piano immagine
Input: $\{b_1, \ldots, b_T\} \in blobVeicolo$: sequenza dei blob
Input: $modelDimension$: insieme di triple che definiscono differenti dimensioni del parallelepipedo
Output: $pose3DVeicolo$: sequenza di pose 3D occupate dal veicolo

Ricorsione in avanti: inizializzazione;
foreach m *in* $modelDimension$ **do**
  $(x_1^{3D}, y_1^{3D}) = proietta2D3D(p_1, m)$;
  $(x_2^{3D}, y_2^{3D}) = proietta2D3D(p_1, m)$;
  $\theta_1 = orientamentoVettore((x_1^{3D}, y_1^{3D}), (x_2^{3D}, y_2^{3D}))$;
  $\mathbf{s}_{(1,m)} = \{n$ campioni dalla Gaussiana centrata in $(x_1^{3D}, y_1^{3D}, \theta_1)\}$;
  foreach s_i *in* $\mathbf{s}_{(1,m)}$ **do**
    $verCamp_{(s_i,1,m)} = calcolaVerosimiglianza(s_i, b_1, m)$;

Ricorsione in avanti: ricorsione;
for $t = 2$ **to** T **do**
  foreach m *in* $modelDimension$ **do**
    $\mathbf{s}_{(t,m)} = particleFilter(\mathbf{s}_{(t-1,m)}, verCamp_{(\cdot,t-1,m)})$ **foreach** s_i *in* $\mathbf{s}_{(t,m)}$ **do**
      $verCamp_{(s_i,t,m)} = calcolaVerosimiglianza(s_i, b_t, m)$;

Ricorsione all'indietro: inizializzazione con le ultime particelle;
foreach m *in* $modelDimension$ **do**
  foreach s_i *in* $\mathbf{s}_{(T,m)}$ **do**
    $s_i.\theta = -s_i.\theta$;

Ricorsione all'indietro;
for $t = T - 1$ **to** 1 **do**
  foreach m *in* $modelDimension$ **do**
    $\mathbf{s}_{(t,m)} = particleFilter(\mathbf{s}_{(t-1,m)}, verCamp_{(\cdot,t-1,m)})$;
    foreach s_i *in* $\mathbf{s}_{(t,m)}$ **do**
      $verCamp_{(s_i,t,m)} = calcolaVerosimiglianza(s_i, b_t, m)$;

$correggiOrientamenti(\mathbf{s}_{(:,m)})$;
foreach m *in* $modelDimension$ **do**
  for $t = 1$ **to** T **do**
    $storiePose(m, t) = poseMedia(\mathbf{s}_{(t,m)})$;
    $mediaTemp(m, t) = media(verCamp_{(:,t,m)})$;
  $mediaPose(m) = media(mediaTemp(m, :))$;
Sceglie il modello migliore;
$M_{best} = \arg\max_m \{mediaPose(m)\}$;
$pose3DVeicolo = storiePose(M_{best}, :)$;

dove x_t è lo stato del sistema al tempo t, per esempio la posizione del veicolo, z_t è la misura e u_t la variabile di controllo che nel caso tipico del tracking è nulla, w_t e v_t sono due rumori Gaussiani che modellano gli errori di misura e del moto. Il sistema su cui fare tracking deve poter quindi essere modellato in modo lineare, la distribuzione dello stato deve essere Gaussiana, e in particolare si deve riuscire ad esprimere un legame tra lo stato e la misura attraverso una matrice H. La soluzione trovata dal filtro di Kalman è in forma chiusa e ottima sotto queste ipotesi.

Abbiamo accennato al fatto che Viterbi è un algoritmo ottimo, ma è ottimo in un senso diverso rispetto a Kalman. Infatti Viterbi ricerca una cammino tra i campioni estratti nella fase di campionamento frame per frame, ed è in questo senso che trova la soluzione ottima: tra i campioni estratti trova la soluzione che massimizza la densità $p(s_{1:T}|z_{1:T})$. Kalman, invece ricerca la soluzione migliore in tutto lo spazio degli stati. Per un discorso analogo non si può parlare di ottimalità del particle filter nè del particle smoother. Più in generale l'ottimalità viene meno utilizzando un metodo Montecarlo, diventando comunque asintotica ottimalità: quanto più il numero di particelle è elevato, tanto più la distribuzione dello stato viene approssimata bene.

La non ottimalità è compensata dal fatto che i metodi Montecarlo permettono di approssimare qualsiasi sistema, senza dover ipotizzare la linearità, la gaussianità dello stato , l'unimodalità, e soprattutto senza aver la necessità di rendere in forma esplicita la relazione tra stima dello stato e misura. Quest'ultima ipotesi risulta un grosso limite per un sistema di tracking 3D in cui la pose viene valutata confrontando la proiezione del modello sull'immagine con il blob del veicolo. Se usassimo il filtro di Kalman dovremmo dare una definizione opportuna di H che permetta di legare la pose stimata con la proiezione del modello e la proiezione del modello con il blob. Questo passo non è banale e il campionamento Montecarlo, permette di evitarlo. Inoltre ipotizziamo che il moto dei veicoli non possa essere adeguatamente modellato da un sistema lineare. Pur sottolineando il fatto che esiste la versione estesa del filtro di Kalman che linearizza un sistema non lineare per rendere possibile eseguire il filtro di Kalman, preferiamo un approccio Montecarlo che gestisce in modo naturale la non linearità

Infine, chiamando n_f il numero dei frame, il filtro di Kalman ha complessità $\mathbb{O}(n_f m^3)$ in cui m è la dimensione dello stato, una quantità "piccola" e costante, quindi si può scrivere $\mathbb{O}(n_f)$. Considerando i tempi di campionamento trascurabili, Viterbi ha complessità dipendente dall'algoritmo di ricerca del cammino minimo usato, la versione dell'algoritmo di Dijkstra utilizzata costa $\mathbb{O}((n_f n_s)^2)$ dove n_s indica il numero di campioni. Il particle filter ha complessità $\mathbb{O}(n_f n_s)$. In Tabella 3.1 vengono riassunte le proprietà presentate. Nonostante l'algoritmo di Viterbi e il particle smoother abbiano una complessità maggiore, li abbiamo scelti per i benefici legati all'utilizzo di un metodo Montecarlo.

Tabella 3.1: Tabella riassuntiva delle proprietà dei tre approcci

	Kalman	Particle	Viterbi
sistema	lineare	qualsiasi	qualsiasi
distribuzione	unimodale gaussiana	multimodale qualsiasi	multimodale qualsiasi
relazione stato-misura	matrice H	qualsiasi	qualsiasi
ottimalità	ottimo	asintoticamente ottimo	asintoticamente ottimo
complessità	$\mathbb{O}(n_f)$	$\mathbb{O}(n_f n_s)$	$\mathbb{O}((n_f n_s)^2)$

Capitolo 4

Architettura dei due sistemi

Nel capitolo precedente abbiamo visto i due algoritmi che utilizziamo per fare tracking concentrando la nostra attenzione sugli aspetti prettamente legati al tracking bayesiano. Diamo ora una descrizione dettagliata dell'architettura dei due algoritmi. Spieghiamo come abbiamo implementato i moduli software necessari ad eseguire gli algoritmi.

I due sistemi sono schematizzati in Figura 4.1 e 4.2. Ogni rettangolo rappresenta un modulo software di cui viene specificato il nome e la funzione. Per semplicità, illustriamo gli algoritmi per tracciare un singolo veicolo. La gestione di più veicoli contemporaneamente è una naturale estensione di quello che presentiamo in questo capitolo: è sufficiente infatti eseguire un istanza dell'algoritmo per ogni veicolo di cui si vuole ricostruire la traiettoria 3D.

Partendo dai dati in ingresso (IN), entrambi gli algoritmi eseguono alcuni passi di inizializzazione. Eseguono `findFirsAndLastContour` che trova il frame da cui iniziare e il frame in cui finire il tracking. Successivamente eseguono i moduli `findFrameWithEntireVehicle` e `correctTraj` per Viterbi e il modulo `findFrameWithEntireVehicle` per il particle smoother, che permettono di gestire il problema dei bordi presentato nel Paragrafo 3.2.

L'algoritmo basato su Viterbi esegue le fasi di campionamento per ogni frame (`singleFrameSingleVehicle`) e di ricerca della sequenza migliore (`viterbiDijkstra`). Come abbiamo visto nell'Algoritmo 2 e 3 del capitolo precedente, in queste due fasi deve essere calcolata la verosimiglianza tra campioni della pose e immagine e il termine a priori che valuta la probabilità di un campione della pose noto il campione precedente. Queste sono implementate rispettivamente per mezzo di `likelihoodProbability`, illustrata nel Paragrafo 4.3 e `computeViterbiPrior` illustrata nel Paragrafo 4.5.1. Il calcolo della probabilità coinvolge anche altri due moduli, `modelProjection`, il cui funzionamento viene descritto nel Paragrafo 4.2, dove discutiamo anche in modo dettagliato del modello dei veicoli utilizzato.

Il particle smoother (Algoritmo 6), a seguito dell'inizializzazione del

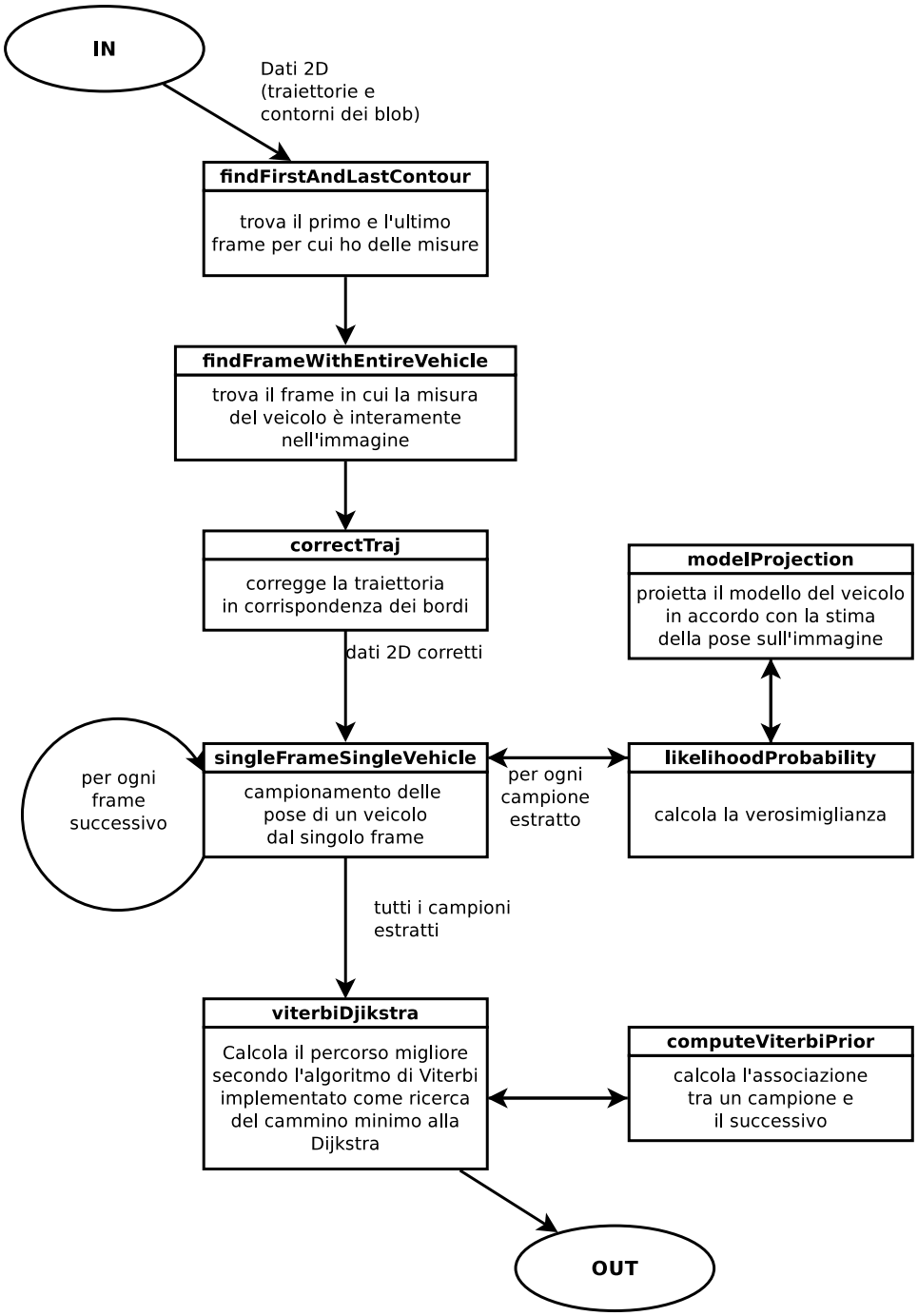


Figura 4.1: Diagramma del sistema di tracking con Viterbi.

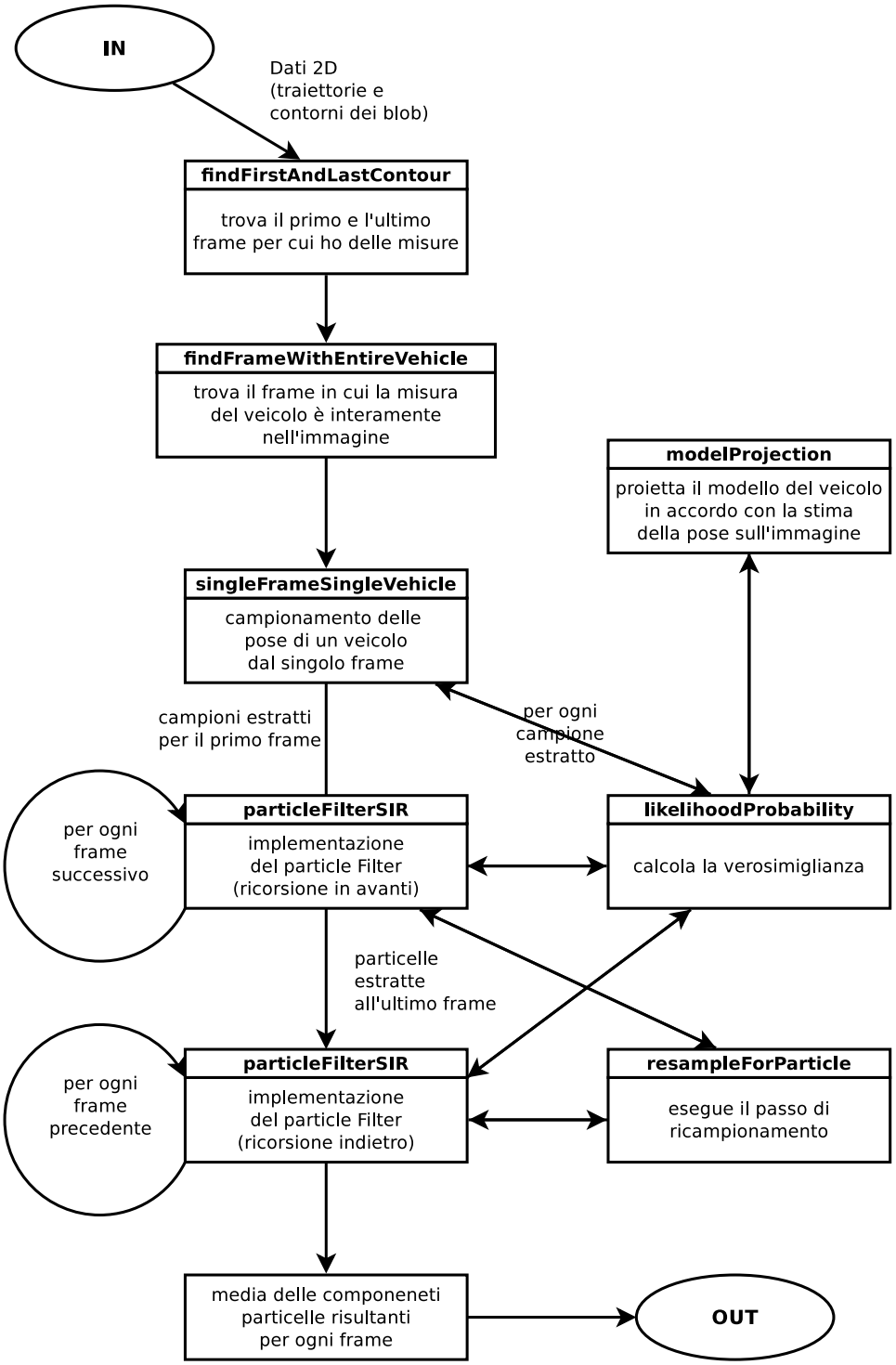


Figura 4.2: Diagramma del sistema di tracking con il Particle smoother.

sistema, esegue un passo per inizializzare il filtro che esegue la ricorsione in avanti utilizzando il medesimo modulo di campionamento dal singolo frame (`singleFrameSingleVehicle`) che l'algoritmo basato di Viterbi richiama per ogni frame. A seguito di questa inizializzazione, viene eseguita la ricorsione in avanti e all'indietro sfruttando lo stesso modulo (`particleFilterSIR`), il quale implementa l'Algoritmo 5 il quale a sua volta richiama l'Algoritmo 4 di ricampionamento che abbiamo implementato in `resampleForParticle`. Infine viene estratto il risultato del tracking come descritto nell'Algoritmo 6.

La funzione di verosimiglianza è calcolata dallo stesso modulo visto per l'algoritmo basato su Viterbi (`likelihoodProbability`). Il termine a priori non viene in questo caso calcolato esplicitamente, perché, per come funziona il particle smoother, questo termine viene utilizzato nel momento in cui generiamo le nuove particelle. Nel Paragrafo 4.5.2 spiegheremo dunque come abbiamo modellato il termine a priori utilizzato nel modulo `particleFilterSIR`.

Abbiamo scelto di implementare i due sistemi in Matlab. Nonostante questo linguaggio non permetta di ottenere algoritmi computazionalmente efficienti, esso ci ha permesso di concentrare i nostri sforzi sull'aspetto algoritmico rispetto a quello implementativo. Vedremo comunque che, dove possibile, abbiamo adottato delle tecniche per rendere il codice sviluppato più efficiente, adottando per esempio delle euristiche (4.6) e le funzioni MEX (4.3.2).

4.1 La gestione dei bordi

Nel capitolo precedente abbiamo trattato il problema di gestione delle misure che si trovano nei pressi del bordo dell'immagine (Paragrafo 3.2). Quando i veicoli si trovano parzialmente fuori dall'immagine, il tracker 2D genera delle stime imprecise del centroide del veicolo. In questi casi il blob corrispondente al veicolo rappresenta solo una parte di esso, ma il tracker 2D considera il blob come il veicolo intero; quindi il centroide stimato non è il centroide di tutto il veicolo, ma solo della porzione visibile nell'immagine. Affinché i nostri sistemi di tracking 3D rendano più precisa questa stima, dobbiamo tener presente che le stime delle posizioni del veicolo sul piano immagine ottenute dal tracker 2D soffrono di questo problema. Solitamente i punti vicini al bordo dell'immagine corrispondono ai punti iniziali e finali della traiettorie, che corrispondono a quando il veicolo entra nell'immagine o ne esce.

4.1.1 Algoritmo basato su Viterbi

Per il sistema di tracking con Viterbi abbiamo bisogno di una stima 2D del centroide sul piano immagine per ogni istante. Per gestire il problema

dei bordi dobbiamo quindi correggere i punti della traiettoria ottenuta dal tracker 2D in corrispondenza dei margini del piano immagine.

Nel caso in cui il veicolo entra nell'immagine, ovvero, per i punti iniziali della traiettoria 2D, la porzione di traiettoria da correggere va dal punto relativo al primo frame, al punto precedente l'istante in cui il veicolo è completamente nella scena. Questo istante, che chiamiamo t_{vi}, corrisponde al momento in cui il contorno del blob è sufficientemente distante dal bordo dell'immagine e viene calcolato dal modulo `findFrameWithEntireVehicle`.

Calcoliamo ora la direzione ϕ con cui il veicolo entra nella scena. Una buona approssimazione è data dalla direzione del vettore che va dal punto al tempo $t_{vi} + 1$ a quello al tempo t_{vi} (Figura 4.3a). Per correggere la traiettoria trasliamo il blob dell'istante t_{vi} lungo ϕ. Al tempo 0, cioè un istante prima del frame iniziale, il veicolo deve essere fuori dall'immagine, mentre nell'istante successivo deve entrare. Il valore della traslazione lungo ϕ è dunque pari alla lunghezza del veicolo. La distanza lungo ϕ del punto al tempo t_{vi} dal contorno dell'immagine è approssimativamente pari al valore della semilunghezza del veicolo e la chiamiamo d.

Il generico punto all'istante t tale che $1 \leq t < t_{vi}$ è quindi:

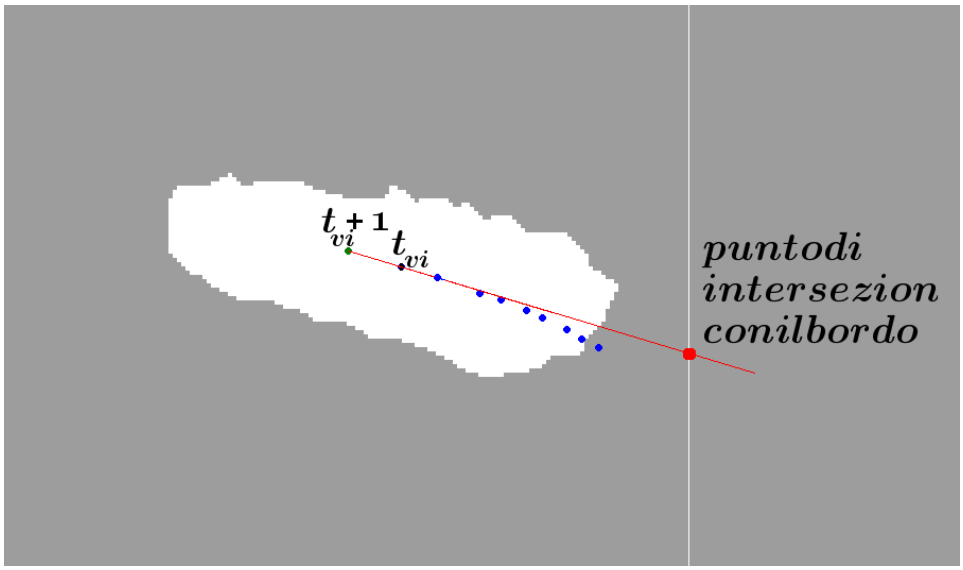
$$\begin{aligned} x_t &= x_{t+1} + \frac{2d}{n_p}(t_{vi} - t)\cos(\phi) \\ y_t &= y_{t+1} + \frac{2d}{n_p}(t_{vi} - t)\cos(\phi) \end{aligned} \tag{4.1}$$

dove n_p è il numero di punti della traiettoria da correggere, ovvero il numero di frame in cui il veicolo è parzialmente fuori dall'immagine. In Figura 4.3 mostriamo un esempio del risultato dell'algoritmo: in Figura 4.3a i centri blu sono quelli che in Figura 4.3b sono stati corretti. Inoltre, per dare l'idea di come entrerebbe il veicolo nella scena con i centri corretti, abbiamo traslato il contorno del blob nel primo centro della nuova traiettoria.

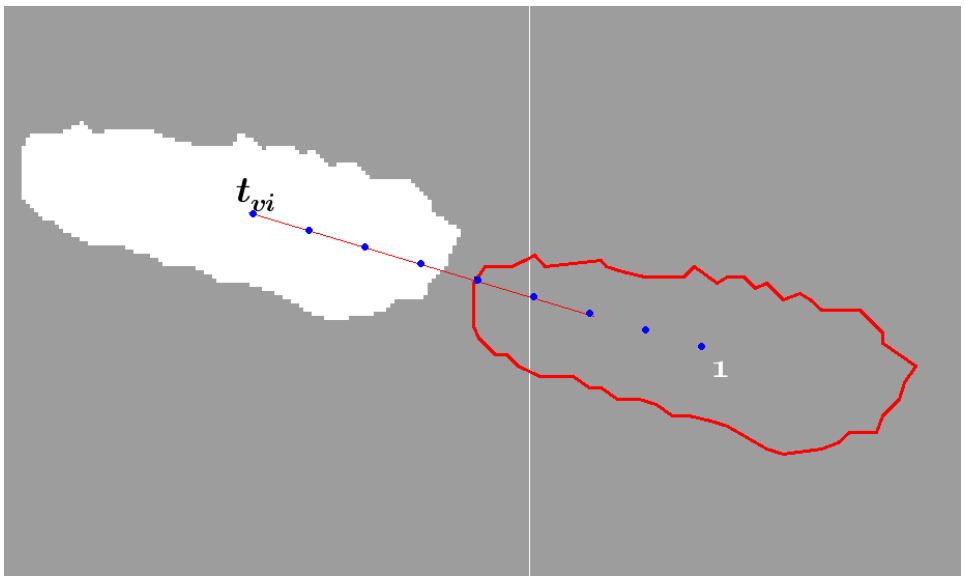
In modo analogo troviamo i punti corretti per l'ultima porzione di traiettoria. I punti corretti offrono una approssimazione migliore di quelli non corretti, ma meno precisa rispetto agli altri punti della traiettoria trovati con il tracker 2D. Per questo motivo, nel modulo di campionamento aumentiamo la varianza delle componenti della Gaussiana quando partiamo dai punti corretti con il metodo presentato.

4.1.2 Particle smoother

Il particle smoother non utilizza invece tutte le stime 2D della posizione del centroide, ma utilizza solo la prima come inizializzazione, per il resto dell'algoritmo utilizza solo i blob. L'idea è quindi quella di utilizzare la stima 2D opportuna per inizializzare il sistema. Come per il caso precedente, la prima stima che riteniamo affidabile è quella per cui il veicolo è interamente nell'immagine, quindi quella all'istante t_{vi}. A partire da t_{vi} eseguiamo il



(a) *Punti della traiettoria prima di applicare la funzione di correzione.*



(b) *Punti corretti.*

Figura 4.3: Esempio di correzione dei punti della traiettoria vicini al bordo.

filtro che esegue la ricorsione in avanti fino all'ultimo istante T in cui è disponibile la misura.

In teoria potremmo partire da queste ultime particelle per eseguire la ricorsione all'indietro. In realtà, le particelle generate dal particle filter quando il veicolo esce dalla traiettoria, forniscono una stima meno precisa della pose rispetto a quella data negli istanti in cui tutto il veicolo è presente nell'immagine. Questo è dovuto al fatto che il veicolo è rappresentato solo in parte dalla misura. Per questo motivo, invece di inizializzare il filtro per la ricorsione all'indietro con le particelle al tempo T, le inizializziamo con le particelle trovate durante la ricorsione in avanti al tempo t_{vf}, calcolato come nel paragrafo precedente. Eseguiamo questa ricorsione all'indietro fino al tempo $t = 1$. otteniamo così la stima anche per quegli istanti che inizialmente non avevamo considerato nella ricorsione in avanti. Si noti che questo metodo rende l'algoritmo uno smoother solo per gli istanti da t_{vi} a t_{vf}, per i rimanenti istanti eseguiamo infatti solo una delle due ricorsioni. Questo non pregiudica comunque le prestazioni del nostro algoritmo dato che solitamente il numero di frame per cui il veicolo è fuori dall'immagine è basso relativamente al numero totale di frame in cui è presente il veicolo. Fanno eccezione i camion che, per la loro grande mole sono parzialmente fuori dall'immagine per un numero non trascurabile di frame. Il beneficio ottenuto dalla stima con i tracker 3D compensa il fatto che non compiamo smoothing per un numero significativo di frame.

Per il caso a telecamera multipla il ragionamento è analogo: calcoliamo t^i_{vi} e t^i_{vf} per ogni telecamera. Partiamo ad eseguire tracking dal frame t^i_{vi} della prima telecamera in cui appare il veicolo eseguiamo la ricorsione in avanti, e inizializziamo il filtro all'indietro con le particelle all'istante t^i_{vf} calcolato per l'ultima telecamera in cui appare il veicolo.

4.2 Proiezione del modello

Il primo modulo che descriviamo è quello che calcola la proiezione di un modello sull'immagine, ma prima di fare ciò, è necessario chiarire precisamente come abbiamo scelto il modello del veicolo.

4.2.1 Il modello dei veicoli

Un veicolo può avere differenti forme e dimensioni: nel nostro caso si vedono automobili, camion e motocicli molto differenti tra loro. Il nostro scopo è trovare un modello, o un gruppo di modelli che approssimi con sufficiente precisione questa moltitudine di veicoli. La scelta deve essere quindi un buon compromesso tra semplicità del modello e buona rappresentazione dei veicoli. Un'alta complessità del modello può riuscire ad approssimare molto bene i veicoli, ma per tenere in considerazione la varietà di forme si devono

utilizzare numerosi modelli, e ciò rischia di avere un elevato onere computazionale. D'altra parte, adottando un modello semplice, si corre il rischio di non dare una descrizione sufficientemente informativa dei veicoli nella scena.

Un primo modo per decidere quale modello o quali modelli da utilizzare dipende dallo scopo dell'algoritmo: se il compito è quello di classificare i veicoli si tende a scegliere un modello complesso, se, invece, si deve fare tracking 3D, è spesso sufficiente un modello semplice.

La scelta del modello dipende anche dal metodo con cui viene calcolata la verosimiglianza. Quando si utilizza un algoritmo basato sull'estrazione e il confronto del contorno (edge-based) è necessario un modello fedele all'oggetto tracciato. Infatti, confrontando i segmenti del modello proiettato sull'immagine con i contorni del veicolo, la forma del primo deve essere quanto più simile a quella del secondo per ottenere una stima ragionevole della verosimiglianza. Utilizzando un metodo region-based, come nel nostro caso, il modello può essere meno dettagliato: per una buona stima è sufficiente che la sua proiezione abbia una sagoma simile a quella della regione occupata dalla vettura. Tenendo in considerazione il rumore che influenza l'estrazione della regione con background subtraction, un modello a forma di parallelepipedo riesce a dare una ragionevole approssimazione della forma del veicolo sull'immagine.

Un altro fattore che determina la scelta del modello è la conoscenza di quale tipo di veicoli transitano nella scena. In un ambiente cittadino transitano spesso vetture e meno frequentemente mezzi pesanti, escludendo saltuari passaggi di autobus, come nel caso del video analizzato nell'articolo di Song e Nevatia in [58]. In questa situazione l'uso di un modello a dimensione fissa può essere una buona approssimazione, dato che le autovetture hanno dimensioni e forme simili. In generale, però, per rappresentare diversi tipi di veicoli, non può essere sufficiente un modello a dimensione fissa per rappresentare in modo soddisfacente tutti i veicoli.

La nostra scelta è quindi ricaduta su un modello di forma semplice, un parallelepipedo, ma di dimensione variabile. L'idea è quella di circoscrivere il veicolo tramite un parallelepipedo a ogni istante in cui questo appare nella scena. Per questo motivo le tre dimensioni del parallelepipedo, ovvero lunghezza altezza e profondità (l, a, p), devono essere quanto più possibile simili a quelle del veicolo (Figura 4.4).

Rappresentiamo i veicoli discriminandoli per *classe* di appartenenza: camion, automobile e motociclo. Ognuna di esse si differenzia per le dimensioni degli spigoli del parallelepipedo. Per esempio, un camion ha in generale le tre dimensioni maggiori rispetto ad una autovettura e quest'ultima rispetto al motociclo. Inoltre, tenendo in considerazione che il rapporto tra le lunghezze definisce la forma del parallelepipedo, questo è simile all'interno della stessa classe. Un mezzo pesante ha altezza e profondità simili tra loro mentre la lunghezza è di gran lunga maggiore, anche tre quattro volte le altre dimensioni. Un'automobile ha anch'essa a e p simili, ma la lunghezza

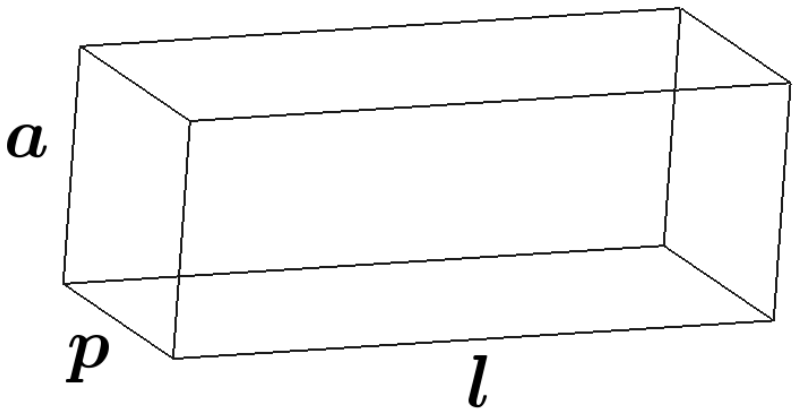


Figura 4.4: Modello utilizzato.

è poco più del doppio di a o p. Ogni classe c è definita quindi da un valore di riferimento L^c, e da un rapporto r^c_{la} tra lunghezza e altezza e r^c_{lp} tra lunghezza e profondità. La lunghezza L^c rappresenta una stima della lunghezza media dei veicoli che appartengono alla classe c.

Una volta definite la lunghezza L^c e i valori di r^c_{la} e r^c_{lp} per ogni classe c di veicoli, generiamo un insieme di triple $< l^c_i, a^c_i, p^c_i >$ che definiscono le dimensioni dei modelli utilizzati nell'algoritmo di tracking, e sono l'ingresso *modelDimension* dell'Algoritmo 2 e 6. Per ogni classe generiamo n triple nel seguente modo. Campioniamo n valori da una Gaussiana centrata sul valore L^c di riferimento della classe (Figura 4.5). Per ogni classe la varianza σ^2_c della Gaussiana da cui campionare è differente ed è proporzionale al valore di L^c: la dimensione di un'automobile varia in misura inferiore rispetto a quella di un camion e in misura maggiore rispetto ad un motociclo. Ad esempio è probabile trovare camion di 7, 10 o 13 metri, ma, se è facile trovare auto lunghe 4 metri, è improbabile vedere auto di 1 o 7 metri. I valori campionati rappresentano n lunghezze l^c_s con $i = 1, \ldots, n$. Da ognuna delle l^c_i ricaviamo l'altezza e la profondità corrispondenti, ottenendo la tripla $d^c_i = \{l^c_i, a^c_i = l^c_i \cdot r^c_{la}, p^c_i = l^c_i \cdot r^c_{lp}\}$. L'insieme *modelDimension* sarà quindi composto da tutte le triple $d^c_i, \forall c, \forall i$ t.c. $1 < i \leq n$. Nel seguito ci riferiremo ad ognuno di queste triple come i *campioni classe-scala*.

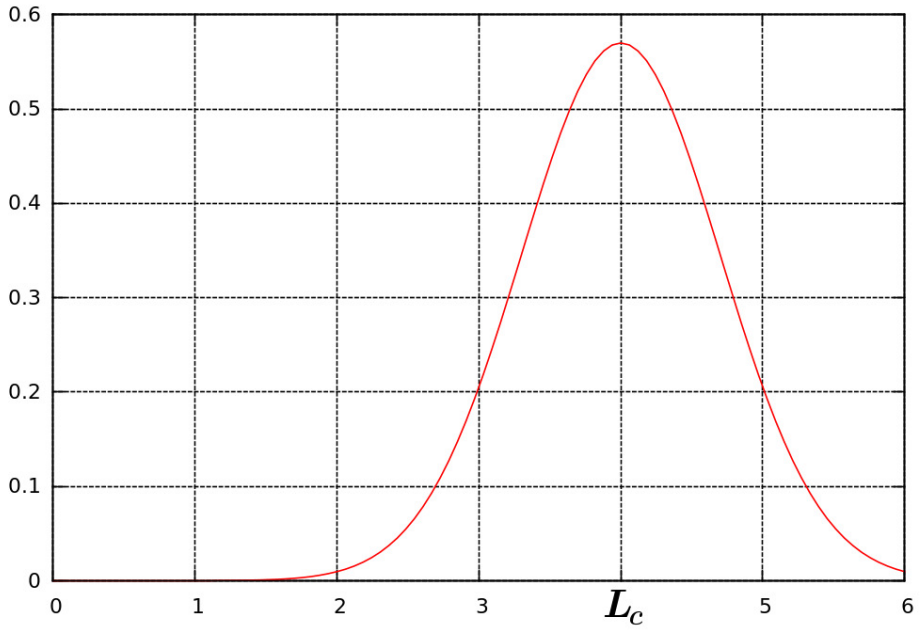


Figura 4.5: Collocazione della Gaussiana da cui campioniamo la dimensione di riferimento.

4.2.2 Implementazione del modulo per la proiezione del modello

Nel Capitolo precedente abbiamo parlato spesso di proiettare il modello del veicolo sull'immagine. Questa operazione consente di confrontare un modello nel mondo con la misura del veicolo, ovvero il blob ottenuto con la background subtraction. Questa operazione permette di valutare la bontà, ovvero la verosimiglianza della pose. La Figura 4.6 offre un esempio della situazione in cui ci troviamo quando dobbiamo proiettare il modello nell'immagine. In blu è rappresentato il modello che viene proiettato attraverso le linee sul piano immagine della telecamera (in rosso). Per proiettare il modello sull'immagine è necessario conoscere la pose, la dimensione del modello e la matrice di calibrazione della telecamera. Dal valore della pose ricaviamo il centro $(x_{3D}^{curr}, y_{3D}^{curr})$ attorno cui costruire il modello e l'orientamento θ_{3D}^{curr} che deve assumere nel mondo. Chiamiamo l^{curr}, a^{curr} e p^{curr} la lunghezza, l'altezza e la profondità del campione corrente di classe-scala visto in precedenza. Allora i vertici del modello che ha centro nell'origine e orientamento

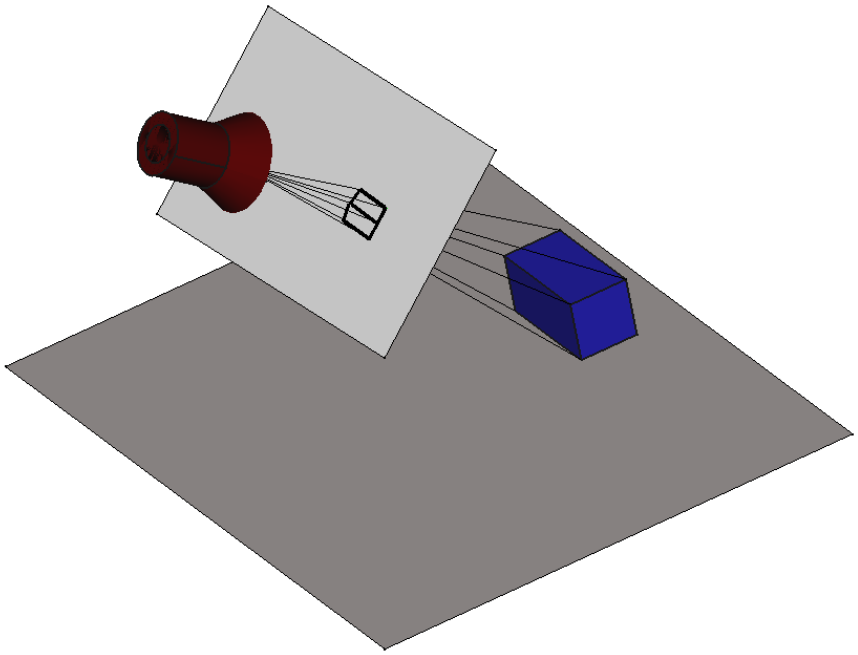


Figura 4.6: Proiezione del modello dul piano immagine.

nullo sono:

$$\begin{aligned}\bar{\mathbb{V}}_{3D} = \Bigg\{ &\left\{-\frac{l^{curr}}{2}, -\frac{p^{curr}}{2}, 0\right\}, \left\{\frac{l^{curr}}{2}, -\frac{p^{curr}}{2}, 0\right\}, \\ &\left\{\frac{l^{curr}}{2}, \frac{p^{curr}}{2}, 0\right\}, \left\{-\frac{l^{curr}}{2}, \frac{p^{curr}}{2}, 0\right\}, \\ &\left\{-\frac{l^{curr}}{2}, -\frac{p^{curr}}{2}, a^{curr}\right\}, \left\{\frac{l^{curr}}{2}, -\frac{p^{curr}}{2}, a^{curr}\right\}, \\ &\left\{\frac{l^{curr}}{2}, \frac{p^{curr}}{2}, a^{curr}\right\}, \left\{-\frac{l^{curr}}{2}, \frac{p^{curr}}{2}, a^{curr}\right\}\Bigg\}\end{aligned} \tag{4.2}$$

Indichiamo con $\bar{v}^i_{3D}$ l'i-esimo vertice e $\bar{v}^i_{3D-H}$ lo stesso espresso in coordinate omogenee. Per ottenere i vertici del modello che ha pose $(x^{curr}_{3D}, y^{curr}_{3D}, \theta^{curr}_{3D})$ è sufficiente applicare una rototraslazione R:

$$R = \begin{bmatrix} \cos(\theta^{curr}_{3D}) & -\sin(\theta^{curr}_{3D}) & 0 & x^{curr}_{3D} \\ \sin(\theta^{curr}_{3D}) & \cos(\theta^{curr}_{3D}) & 0 & y^{curr}_{3D} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

ad ognuno dei punti $\bar{v}^i_{3D_H}$:

$$v^i_{3D_H} = R \cdot \bar{v}^i_{3D_H} \tag{4.4}$$

La proiezione del modello è quindi calcolata proiettando ogni vertice $v^i_{3D_H}$ sull'immagine attraverso la matrice P di calibrazione:

$$v^i_{2D_H} = P \cdot v^i_{3D_H} \tag{4.5}$$

Consideriamo infine la distorsione causata dalla telecamera: il punto $v^i_{2D_H}$ è il risultato della proiezione tramite la matrice P, quindi è un punto non distorto. Le misure dei veicoli sono però calcolate sull'immagine reale che soffre di una distorsione modellata da una funzione $L(r)$, dove r è il centro della distorsione. Quindi dobbiamo trasformare ogni $v^i_{2D_H}$ nel corrispondente punto distorto. Il punto che utilizziamo è quindi:

$$\hat{v}^i_{2D_H} = L(r) \cdot v^i_{2D_H} \tag{4.6}$$

La funzione $L(r)$ è stata calcolata in una fase precedente a questo lavoro. Riassumendo, per ogni vertice $\bar{v}^i_{3D}$ del modello definiti nell'insieme $\bar{\mathbb{V}}_{3D}$:

$$\bar{v}^i_{2D} = L(r) \cdot P \cdot \bar{v}^i_{3D} \tag{4.7}$$

La regione che occupa la proiezione del modello sull'immagine è il *convex hull* dei vertici $\bar{v}^i_{2D}$, ovvero il poligono minimo che racchiude tutti i vertici. Questo è possibile perché assumiamo che la proiezione del modello sia convessa. Quest'ipotesi non sarebbe vera se per esempio considerassimo anche l'ombra del modello sul piano immagine come in [58] . In quest'ultimo caso infatti la regione occupata dal veicolo sarebbe l'unione dei due convex hull dei vertici proiettati del modello e dell'ombra.

4.3 La verosimiglianza

Descriviamo ora il metodo usato per valutare la verosimiglianza della pose di un campione rispetto alla misura del veicolo, ovvero il valore di $p(z_t|s^i_t)$. Per semplicità trattiamo prima il caso di una singola telecamera, poi spieghiamo come adattare il calcolo quando dobbiamo fare tracking su più telecamere.

Data la pose, proiettiamo il modello sull'immagine, ottenendo così il poligono da confrontare con il poligono che racchiude la regione estratta dalla background subtraction. Calcoliamo l'intersezione tra i due poligoni e l'area, che chiamiamo $A_{overlap}$, del poligono risultante. Indichiamo con A_{mv} l'area visibile del modello proiettato (Figura 4.7a); questa si trova calcolando l'area dell'intersezione tra la proiezione del modello e il rettangolo che racchiude i bordi dell'immagine. Se infatti un poligono si proietta in parte all'esterno dell'immagine, come avviene in Figura 4.7, non vogliamo che l'area che non si proietta nella scena contribuisca al calcolo della verisimiglianza il cui valore risulterebbe falsato. Infine A_v è l'area della regione occupata dal veicolo ovvero l'area del blob. Per valutare la bontà della pose procediamo in modo

(a) *Area* A_{mv}. (b) *Area* A_m.

Figura 4.7: Esempio di proiezione del modello in parte fuori dall'area visibile in cui vengono evidenziate le grandezze A_{mv} e A_m.

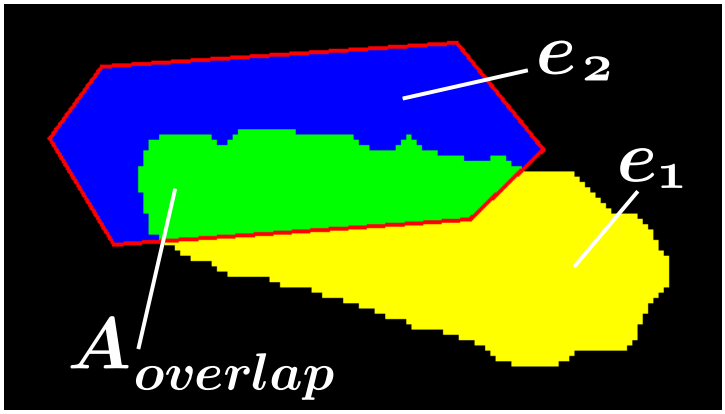


Figura 4.8: Grandezze utilizzate nel calcolo della verosimiglianza.

analogo a quanto fatto da Song e Nevatia in [58]. Definiamo due grandezze, e_1 ed e_2 che rappresentano due differenti errori:

$$e_1 = A_v - A_{overlap} \tag{4.8}$$
$$e_2 = A_{mv} - A_{overlap} \tag{4.9}$$

Per esempio, in Figura 4.8 la zona gialla corrisponde a e_1, la zona blu a e_2 e infine l'area della regione verde è $A_{overlap}$. Un errore e_1 elevato indica che il modello lascia scoperta una grande area di veicolo. Se invece è e_2 ad avere un valore alto, significa che gran parte del modello non copre la regione occupata nel veicolo.

Si noti che nel caso peggiore, ovvero quando non c'è sovrapposizione, i due errori valgono $e_1 = A_v - A_{overlap}$ e $e_2 = A_{mv} - A_{overlap}$ mentre nel caso

migliore, ovvero con sovrapposizione massima:

$$\left.\begin{aligned} e_1 &= A_v - A_{mv} \\ e_2 &= 0 \end{aligned}\right\} \quad \text{se } A_v > A_{mv}$$
$$\left.\begin{aligned} e_1 &= 0 \\ e_2 &= A_{mv} - A_v \end{aligned}\right\} \quad \text{se } A_{mv} > A_v \tag{4.10}$$

Valutiamo l'errore complessivo e come:

$$e = \frac{\lambda_1 e_1 + \lambda_2 e_2}{A_v} \quad \lambda_1 + \lambda_2 = 1 \tag{4.11}$$

I coefficienti λ_1 e λ_2 sono scelti in base all'esperienza. Servono a pesare i due errori distintamente. Quando $\lambda_1 > \lambda_2$ viene pesato maggiormente l'errore e_1, in caso contrario, se $\lambda_1 < \lambda_2$ viene dato più peso a e_2.

A differenza dell'algoritmo di Song e Nevatia la probabilità non viene calcolata tramite una funzione esponenziale. Chiamiamo $\phi = 1 - e$, la funzione di probabilità è:

$$p(z_t|s_t^i) = \begin{cases} \phi, & \text{se } \phi > 0, \\ 0, & \text{se } \phi < 0. \end{cases} \tag{4.12}$$

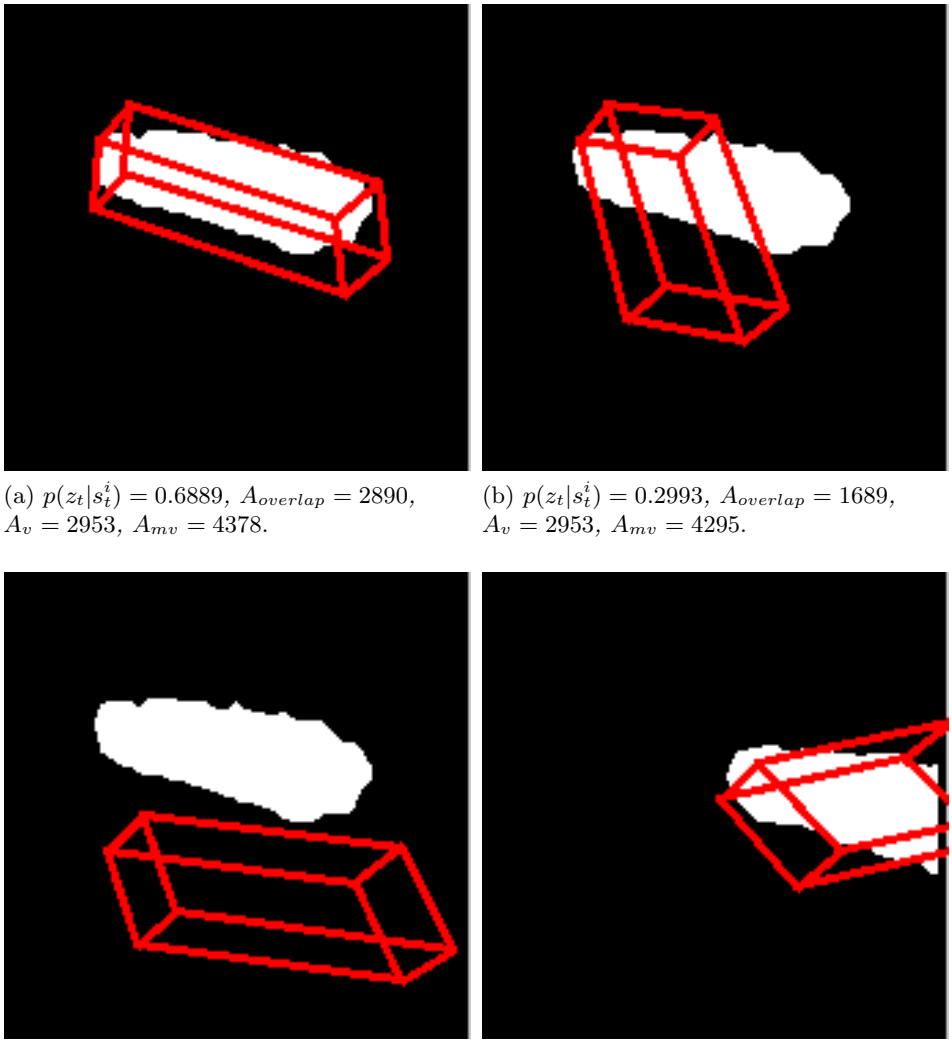
Abbiamo preferito questa funzione perché ha dato dei risultati migliori a livello sperimentale rispetto alla funzione esponenziale di Song e Nevatia. In Figura 4.9 presentiamo degli esempi di comportamento della funzione di verosimiglianza.

4.3.1 Calcolo per il caso multicamera

Quando abbiamo a che fare con più di una telecamera, l'Equazione (4.12) non è sufficiente per definire la probabilità di verosimiglianza, ma dobbiamo considerare come viene ripresa la scena: un veicolo può essere presente contemporaneamente in più telecamere e non comparire in altre.

Dobbiamo pesare in modo opportuno le probabilità di una pose per ogni telecamera. L'idea è quella di dare maggior valore alle probabilità calcolate per un punto di vista da cui si vede interamente il modello proiettato, e una probabilità bassa se si vede parte del modello e nulla se non si vede. A_m è l'area totale del modello proiettato senza considerare quanto se ne vede nell'immagine (Figura 4.7b). Il valore $P(C_i) = \frac{A_{mv}}{A_m}$ è il peso dell'i-esima camera e corrisponde alla percentuale di modello vista nell'immagine. La probabilità di verosimiglianza per n_c telecamere è:

$$p(z_t|s_t^i) = \frac{1}{p(C_1) \cdot \cdots \cdot p(C_{n_c})} \big(P(C_1)p_1(z_t|s_t^i) + \cdots + P(C_{n_c})p_{n_c}(z_t|s_t^i)\big) \tag{4.13}$$



(a) $p(z_t|s_t^i) = 0.6889$, $A_{overlap} = 2890$, $A_v = 2953$, $A_{mv} = 4378$.

(b) $p(z_t|s_t^i) = 0.2993$, $A_{overlap} = 1689$, $A_v = 2953$, $A_{mv} = 4295$.

(c) $p(z_t|s_t^i) = 0$, $A_{overlap} = 0$, $A_v = 2953$, $A_{mv} = 5287$.

(d) $p(z_t|s_t^i) = 0.6754$, $A_{overlap} = 2148$, $A_v = 2357$, $A_{mv} = 3284$.

Figura 4.9: Esempi di calcolo della verosimiglianza.

Tabella 4.1: Tabella riassuntiva dei tempi impiegati nel calcolo della verosimiglianza per 1000 campioni.

	contorno semplice	**contorno complesso**
maschere binarie	39.070200 s	38.813850 s
poligoni	18.128734 s	23.429599 s
poligoni MEX	1.622065 s	2.619747 s

dove $p_j(z_t|s_t^i)$ con $j = 1, 2, 3$ è la probabilità calcolata in (4.12) per la telecamera j, e il coefficiente $\frac{1}{p(C_1)\cdots\cdot p(C_{n_c})}$ ha la funzione di normalizzare la probabilità in modo che vari tra 0 e 1.

4.3.2 Una funzione efficiente

La verosimiglianza viene calcolata numerose volte, di conseguenza la funzione che la calcola deve essere efficiente. Le due operazioni base utilizzate sono il calcolo dell'intersezione tra due due regioni e dell'area di una regione. Una regione può essere descritta in due modi: con un'immagine binaria attraverso i pixel pari a 1 oppure con un poligono.

Abbiamo provato ad utilizzare le funzioni offerte da Matlab per le due operazioni nei due casi. Per valutare l'efficienza abbiamo calcolato 1000 volte la probabilità in una singola camera per un campione per cui c'è sovrapposizione tra la proiezione del modello e l'area del veicolo. Questo equivale a calcolare la verosimiglianza di 1000 campioni differenti. Valutiamo per completezza due casi: in uno il contorno della regione del veicolo è semplice, ovvero il poligono ha pochi vertici, nell'altro la regione è molto più complessa. La funzione che usa i poligoni è meno efficiente nel secondo caso, mentre quella che utilizza le maschere è invariante nei due casi. Infatti il calcolo con le immagini binarie impiega 39.070200 e 38.813850 secondi, rispettivamente per il caso semplice e per quello complesso, quindi pressoché lo stesso tempo, l'utilizzo dei poligoni permette il calcolo in 18.128734 e 23.429599 secondi (Tabella 4.1). Vediamo comunque che, in media, l'uso dei poligoni permette di calcolare la stessa quantità in metà tempo.

Valutare 1000 campioni in circa 20 secondi è comunque poco efficiente dato che ad ogni frame dobbiamo valutarne oltre 3000 (3 classi, per 10 scale, per > 100 campioni). Abbiamo dunque utilizzato una funzione MEX esistente (PoligonClip) per calcolare l'intersezione e ne abbiamo implementata una per computare l'area dei polgoni[1]. Questo ci ha permesso di valutare i 1000 campioni in 1.622065 secondi nel caso semplice e in 2.619747 secondi nel caso complesso. In Tabella 4.1 riassumiamo i risultati.

[1]Le funzioni MEX sono scritte in C e sono eseguibili all'interno del codice Matlab.

4.4 Il campionamento per un singolo frame

L'altro modulo comune ai due sistemi è quello che estrae i campioni della pose per un singolo frame. Per l'algoritmo a particelle questo ha la funzione di inizializzare il filtro; con Viterbi invece questo modulo viene utilizzato ad ogni frame per ottenere i campioni tra i quali l'algoritmo di Viterbi sceglie quelli che costituiscono il percorso migliore.

4.4.1 Retroproiezione del punto 2D in 3D (Backprojection)

In questo modulo partiamo dai dati 2D per ottenere campioni 3D, per fare questo è necessario trasformare un punto nel piano immagine in un punto nel mondo. L'operazione di proiezione dei punti 2D in 3D si chiama *retroproiezione* o *backprojection*. Nel Paragrafo 2.2.1 abbiamo visto come calcolare un punto 2D attraverso la matrice P dato il punto nel sistema di riferimento del mondo. L'operazione inversa è meno banale. Infatti, un punto del piano immagine è la proiezione di uno degli infiniti punti che giace sulla retta passante per il punto stesso e il centro di proiezione della telecamera. Per definire quale tra gli infiniti punti è la corretta retroproiezione 3D intersechiamo la retta con un piano parallelo al piano della rotonda ad una altezza a_π. In Figura 4.10 si nota come varia il punto proiettato sul piano al variare del piano a cui la retta viene intersecata. Siccome stiamo retroproiettando il centro del veicolo, il valore di a_π dipende dalla dimensione del modello che stiamo considerando. Per un campione classe-scala di altezza a, a_π assume il valore $a_\pi = \frac{a}{2}$.

4.4.2 Implementazione del modulo di campionamento per un singolo frame

I dati in ingresso a questo modulo sono i risultati del tracker 2D: le traiettorie 2D e i blob relativi al veicolo tracciato. Per un generico istante t sono noti: il blob corrente del veicolo e il centro 2D del veicolo per t e $t+1$. Il risultato di questo modulo è un insieme di campioni della pose del veicolo e il valore della verosimiglianza ad essi associato. Nel caso in qui questo modulo viene eseguito per Viterbi, salviamo inoltre la stima dello spostamento del veicolo dal frame corrente al successivo nella variabile ρ_t^i, differente per ogni campione classe-scala i: questo valore viene nel computo del termine a priori nel Paragrafo 4.5.1.

Indichiamo con $\mathbf{c}_t^{2D} = \left\{x_t^{2D}, y_t^{2D}\right\}$ il centro 2D del veicolo stimato dal tracker 2D all'istante t. Per ogni campione i classe-scala che definisce le possibili dimensioni del modello calcoliamo la retroproiezione del punto al tempo t e $t+1$ che chiamiamo $\mathbf{c}_t^{3D^i} = \left\{x_t^{3D^i}, y_t^{3D^i}, a_\pi^i\right\}$ e $\mathbf{c}_{t+1}^{3D^i} = \left\{x_{t+1}^{3D^i}, y_{t+1}^{3D^i}, a_\pi^i\right\}$. Trasliamo poi i punti retroproiettati sul piano della rotonda, ovvero annulliamo la componente lungo l'asse z mantenendo invariate le componenti lungo

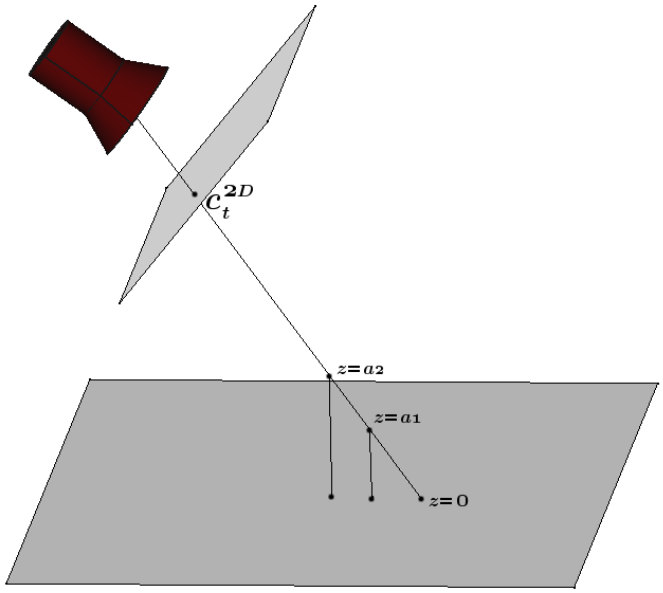


Figura 4.10: Retroproiezione del punto c_t^{2D} sul piano immagine per differenti altezze del piano di intersezione.

gli assi x e y. Quindi:

$$\begin{aligned} \bar{\mathbf{c}}_t^{3D^i} &= \left\{ x_t^{3D^i}, y_t^{3D^i}, 0 \right\} \\ \bar{\mathbf{c}}_{t+1}^{3D^i} &= \left\{ x_{t+1}^{3D^i}, y_{t+1}^{3D^i}, 0 \right\} \end{aligned} \tag{4.14}$$

I punti $x_t^{3D^i}$ e $y_t^{3D^i}$ sono una buona approssimazione iniziale per le componenti x e y della pose. Approssimiamo l'orientamento θ con il vettore spostamento $\bar{\mathbf{c}}_t^{3D^i} \rightarrow \bar{\mathbf{c}}_{t+1}^{3D^i}$, dato che il veicolo è solitamente orientato nella direzione in cui viaggia. Inoltre ρ_t^i è pari al modulo di questo vettore:

$$\rho_t^i = \sqrt{(x_{t+1}^{3D^i} - x_t^{3D^i})^2 + (y_{t+1}^{3D^i} - y_t^{3D^i})^2}. \tag{4.15}$$

Chiamiamo $\hat{\pi}^i = \left\{ \hat{x}^i, \hat{y}^i, \hat{\theta}^i \right\}$ la stima della pose; x, y, θ sono assunte variabili Gaussiane aleatorie indipendenti di media $\bar{x}^i$, $\bar{y}^i$, e $\bar{\theta}^i$ e varianza σ_x^2, σ_x^2 e σ_θ^2. Otteniamo i campioni della pose, da una Gaussiana trivariata con media $\hat{\pi^i}$ e matrice di covarianza:

$$\begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{bmatrix} \tag{4.16}$$

Per ogni campione valutiamo la probabilità di verosimiglianza attraverso il modulo descritto in precedenza. L'utilizzo della Gaussiana trivariata rende l'algoritmo più flessibile. Se in futuro si volesse esprimere una correlazione tra alcune componenti è sufficiente modificare la matrice (4.16).

Il caso multicamera

Finora abbiamo supposto la presenza di un'unica telecamera dalla cui traiettoria calcolare le retroproiezioni 3D. Se lo stesso veicolo viene ripreso da n_t telecamere procediamo in questo modo. Se n_c è il numero di campioni totali che vogliamo per ogni frame, definiamo una grandezza $\bar{n}_t = \lfloor \frac{n_c}{n_t} \rfloor$. Per ogni telecamera estraiamo un numero $\bar{n}_t$ di campioni, per ognuno di essi calcoliamo la probabilità in accordo con (4.13). Il valore salvato in ρ_t^i è invece la media dei valori calcolati a ogni istante per ogni camera.

4.5 Calcolo del termine a priori

Presentiamo ora il metodo che i due algoritmi implementano per modellare il termine a priori $p(s_{t+1}|s_t)$, dove s_t è lo stato del veicolo al tempo t, ovvero la pose stimata (x, y, θ). Il termine a priori esprime la probabilità di s_{t+1} se la pose assunta dal veicolo in precedenza è s_t. In altre parole, dunque, il termine a priori modella il moto del veicolo.

Le due assunzioni su cui basiamo il calcolo del termine a priori sono:

- tra un frame e l'altro un veicolo non subisce accelerazioni significative, quindi presupponiamo che la velocità all'istante t sia uguale alla velocità all'istante $t-1$ più un termine aleatorio, modellato dalle distribuzioni di seguito descritte, che assumiamo essere l'accelerazione;
- distinguiamo tra il caso in cui il veicolo sia fermo e il caso in cui abbia una velocità non trascurabile.

Al fine di distinguere i due casi, utilizziamo una stima del modulo della velocità, che chiamiamo $\dot{\rho}_t$, la quale viene calcolata in modo differente per i due algoritmi. Fissato un valore di soglia τ_{fermo}:

- se $\dot{\rho}_t < \tau_{fermo}$ il veicolo è considerato fermo;
- se $\dot{\rho}_t > \tau_{fermo}$ il veicolo è considerato in moto.

Non poniamo $\tau_{\text{fermo}} = 0$ perché, a causa del rumore, anche se il veicolo è fermo la stima della velocità $\dot{\rho}_t$ difficilmente sarà nulla.

Di seguito parliamo di spostamento e velocità come sinonimi. Lo spostamento infatti si intende tra due frame successivi, t e $t+1$; la velocità del veicolo è lo spostamento compiuto nell'unità di tempo. Se per unità di tempo consideriamo un frame, allora velocità e spostamento coincidono. In

questo caso, se lo spostamento è indicato in metri, l'unità di misura della velocità è $\frac{m}{frame}$. Conoscendo il frame-rate, ovvero quanti frame sono ripresi dalla telecamera per ogni secondo, la velocità in $\frac{m}{s}$ è il prodotto del frame-rate per la velocità in $\frac{m}{frame}$.

4.5.1 Algoritmo basato su Viterbi

Per il calcolo del termine a priori, ricordiamo che per ogni istante t e per ogni campione classe-scala i, abbiamo calcolato nel modulo di campionamento dal singolo frame (Equazione (4.15)) una grandezza ρ_t^i che stima il modulo dello spostamento effettuato dal veicolo tra due frame. Dato che come abbiamo detto velocità e spostamento coincidono ρ_t^i è quindi la stima della velocità del veicolo al tempo t nel caso in cui il campione classe-scala analizzato sia i. Utilizziamo questo termine per distinguere quindi i due tipi di moto del veicolo ponendo $\dot{\rho}_t = \rho_t^i$. Definito questa grandezza, possiamo ora descrivere il calcolo di $p(s_{t+1}|s_t)$.

Nel calcolo del termine a priori che segue facciamo un'ulteriore assunzione oltre alle due già presentate: la probabilità della posizione e dell'orientamento sono indipendenti tra loro; il termine a priori diventa quindi:

$$p(s_{t+1}|s_t) = p(x_{t+1}, y_{t+1}|s_t)p(\theta_{t+1}|s_t). \tag{4.17}$$

Per il calcolo di $p(x_{t+1}, y_{t+1}|s_t)$ definiamo due grandezze:

$$\begin{cases} \rho_{xy} = \sqrt{(x_{t+1} - x_t)^2 + (y_{t+1} - y_t)^2} \\ \theta_{xy} = \text{atan2}(y_{t+1} - y_t, x_{t+1} - x_t) \end{cases}. \tag{4.18}$$

Queste non sono altro che le coordinate polari che esprimono la posizione (x_{t+1}, y_{t+1}) nel sistema di riferimento con origine in (x_t, y_t). Se il veicolo è fermo $p(x_{t+1}, y_{t+1}|s_t)$ dovrebbe avere un valore elevato quando la posizione all'istante $t+1$ è molto vicina a quella al tempo t, indipendentemente da θ_{xy}. Quando invece il veicolo è dotato di moto, $p(x_{t+1}, y_{t+1}|s_t)$ è tanto più probabile quanto più ρ_{xy} è vicino al valore ρ_t^i e θ_{xy} è vicino a θ_t ovvero l'orientamento dello stato al tempo t. Per chiarezza rappresentiamo in Figura 4.11 due generici campioni in due istanti successivi.

Scrivendo $p(x_{t+1}, y_{t+1}|s_t)$ come:

$$p((x_{t+1}, y_{t+1})|s_t) = p(\rho_{xy})p(\theta_{xy}) \tag{4.19}$$

esprimiamo le osservazioni in questo modo:

$$\begin{cases} \rho_{xy} & \sim \mathcal{E}\left(\lambda_{\text{fermo}}\right) \\ p(\theta_{xy}) & = 1 \end{cases}, \tag{4.20}$$

se il veicolo è fermo, mentre

$$\begin{cases} \rho_{xy} & \sim \log\mathcal{N}\left(\log(\rho_t^i - \sigma_\rho^2), \sigma_\rho\right) \\ (\theta_{xy} - \theta_t) & \sim \mathcal{N}\left(0, \sigma_{\theta_{xy}}\right) \end{cases}, \tag{4.21}$$

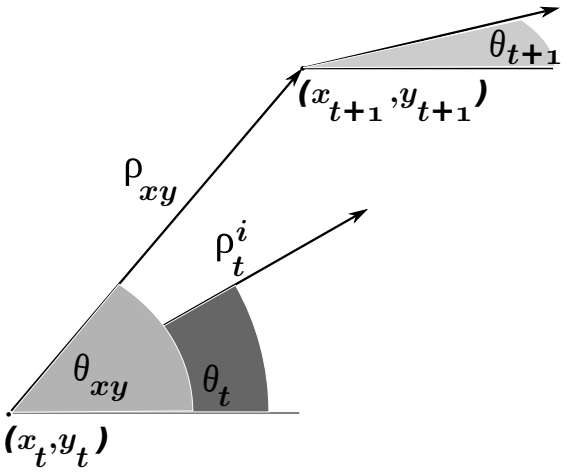


Figura 4.11: Grandezze che concorrono nel computo del termine a priori.

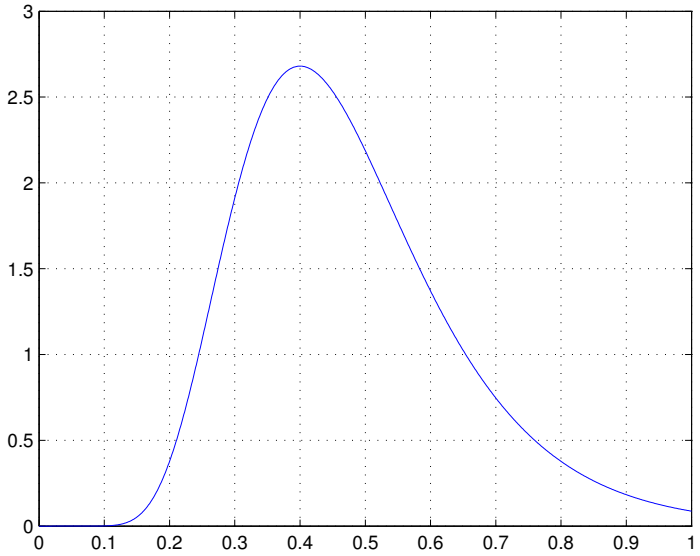
se il veicolo è in movimento.

L'indipendenza del termine a priori dal valore di θ_{xy} nel caso di veicolo fermo viene espressa ponendo $P(\theta_{xy}) = 1$. Nel caso di veicolo in moto, abbiamo preferito esprimere il valore $\theta_{xy} - \theta_t$ come Gaussiana con media nulla invece che esprimere direttamente θ_{xy} come Gaussiana con media θ_t. L'operazione di sottrazione tra angoli non è lineare ed in questo modo viene gestita più semplicemente. La differenza di due angoli α e β non è semplicemente $\alpha - \beta$ dato che $\alpha = \alpha + k2\pi$ con $k \in \mathbb{R}$. Per esempio se $\alpha = 2\pi$ e $\beta = \frac{\pi}{2}$, $\alpha - \beta = \frac{3\pi}{2}$ ma in realtà il valore che ci interessa è $\frac{\pi}{2} = \frac{3\pi}{2} - 2\pi$. Esprimendo la distribuzione di $\theta_{xy} - \theta_t$ è sufficiente controllare che il valore della differenza sia corretto, aggiungendo o sottraendo 2π fino ad ottenere un valore tra $-\pi$ e π. Dal valore delle probabilità che hanno le distribuzioni così definite calcoliamo $p(x_{t+1}, y_{t+1}|s_t)$. Nel caso di veicolo fermo viene utilizzata una distribuzione esponenziale con $\lambda_{\text{fermo}} = \frac{1}{0.5\tau_{\text{fermo}}}$ (Figura 4.12b).

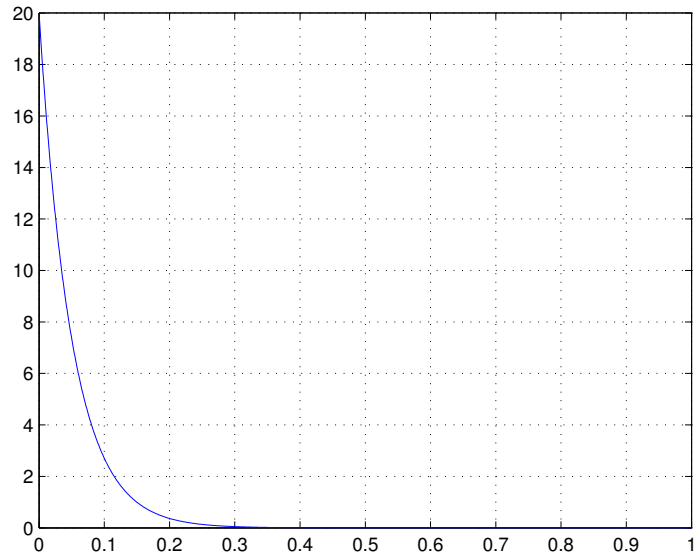
Nel caso in cui il veicolo è in moto, la scelta della distribuzione lognormale permette di penalizzare campioni che si discostano eccessivamente dal valore della velocità stimata, modellando allo stesso tempo il fatto che il veicolo non retrocede a differenza di quanto accadrebbe utilizzando una distribuzione Gaussiana. Si noti come la media utilizzata nella distribuzione di ρ_{xy} non sia semplicemente $\log(\rho_t^i)$ ma $\log(\rho_t^i - \sigma_\rho^2)$. Questo è dovuto al fatto che vogliamo che la moda della distribuzione coincida con la velocità stimata (vedi Figura 4.12a, e il Paragrafo 4.5.3). L'utilizzo di una distribuzione lognormale per il modulo dello spostamento e una Gaussiana per l'orientamento porta ad ottenere una distribuzione come in Figura 4.13, 4.14 e 4.15.

Il termine $p(\theta_{t+1}|s_t)$ dell'Equazione (4.17) viene calcolato invece con le seguenti distribuzioni:

$$\theta_{t+1} - \theta_t \sim \mathcal{N}(0, \sigma_\theta), \tag{4.22}$$



(a) *Con veicolo in moto,* $\rho_t^i = 0.4$, $\sigma_\rho = 0.35$.



(b) *Con veicolo fermo,* $\lambda_{fermo} = \frac{1}{0.05}$.

Figura 4.12: Distribuzione di ρ_{xy}.

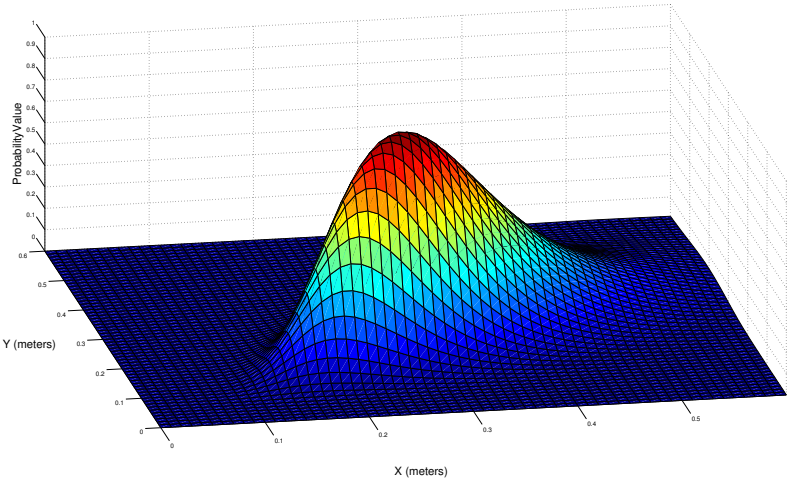


Figura 4.13: Superficie che rappresenta la distribuzione utilizzata per pesare la posizione dello stato s_{t+1} noto s_t.

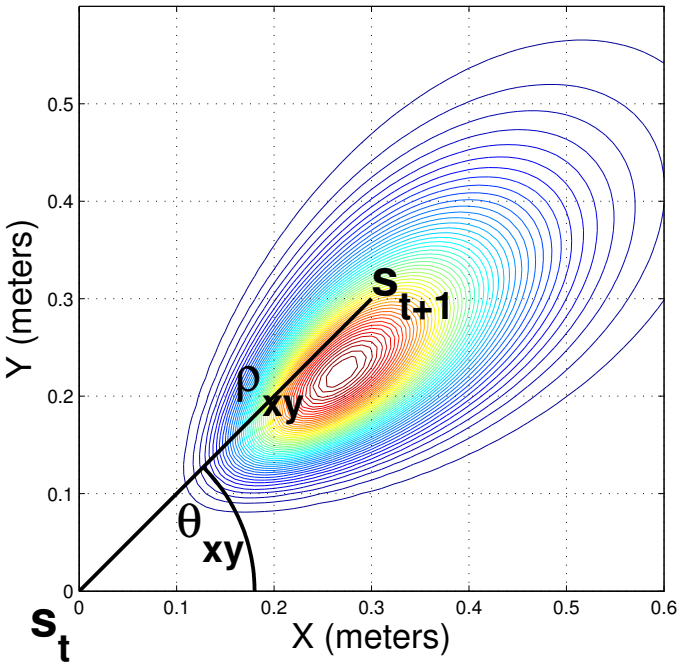


Figura 4.14: Linee di livello che rappresentano la distribuzione utilizzata per pesare la posizione dello stato s_{t+1} noto s_t.

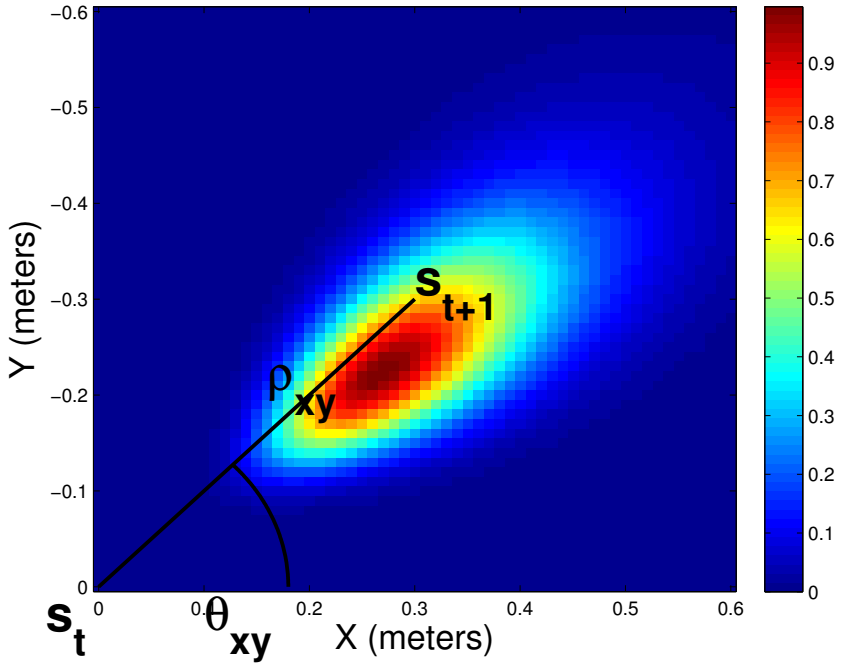


Figura 4.15: Immagine che rappresenta distribuzione utilizzata per pesare la posizione dello stato s_{t+1} noto s_t.

sia con veicolo fermo che in moto. Per il motivo presentato in precedenza abbiamo preferito esprimere la distribuzione della differenza degli angoli piuttosto che dell'angolo θ_{t+1}. Sia quando il veicolo è in moto che quando il veicolo è fermo, l'orientamento del veicolo varia di pochi decimi di grado o rimane costante tra un frame e l'altro, per questo motivo sono probabili i campioni la cui differenza tra θ_{t+1} e θ_t è bassa.

Abbiamo quindi tutti gli elementi per calcolare il valore di $p(s_{t+1}|s_t)$ dati due stati, uno successivo all'altro e nota la stima della velocità ρ_t^i per distinguere il caso di veicolo fermo o in movimento.

4.5.2 Particle smoother

Per il particle smoother il termine a priori viene utilizzato per generare le particelle correnti a partire dalle particelle precedenti. Per definire il termine a priori $p(s_{t+1}|s_t)$ non dobbiamo quindi definire un metodo di calcolo come per Viterbi, ma le distribuzioni opportune da cui vengono generate le particelle al tempo $t+1$ a partire da quelle al tempo t. Infatti, quando il particle filter genera una particella, campiona i valori x, y e θ dalla distribuzione del termine a priori. Lo stato del veicolo, come nel caso precedente, è composto dalle componenti della pose. Per questo algoritmo, abbiamo scelto inoltre di salvare nello stato il valore del modulo della velocità della particella, che coincide con $\dot{\rho}_t$. Questo valore non è campionato come i valori della pose, ma viene solo calcolato e salvato nello stato perché in questo modo è più comodo implementare le funzioni necessarie per il filtro. Un generico stato s_t al tempo t è quindi:

$$s_t = \begin{pmatrix} x_t^{3D} \\ y_t^{3D} \\ \theta_t^{3D} \\ \\ \left[\dot{\rho}_t\right] \end{pmatrix}. \tag{4.23}$$

In Figura 4.16 è rappresentata una generica particella al tempo t. A partire dallo stato s_t noto definiamo la distribuzione di $p(s_{t+1}|s_t)$, da cui l'algoritmo campiona le particelle al tempo t. A questo scopo esprimiamo le coordinate della posizione al tempo $t+1$ in coordinate polari rispetto ad un sistema di riferimento centrato nelle coordinate della posizione al tempo t in questo modo:

$$\begin{cases} x_{t+1}^{3D} = x_t^{3D} + \rho_{\text{camp}} \cos(\theta_{\text{camp}}) \\ y_{t+1}^{3D} = x_t^{3D} + \rho_{\text{camp}} \sin(\theta_{\text{camp}}) \end{cases} \tag{4.24}$$

in cui ρ_{camp} e θ_{camp} sono due variabili aleatorie indipendenti che rappresentano lo spostamento del veicolo tra t e $t+1$. Definiamo le distribuzioni da

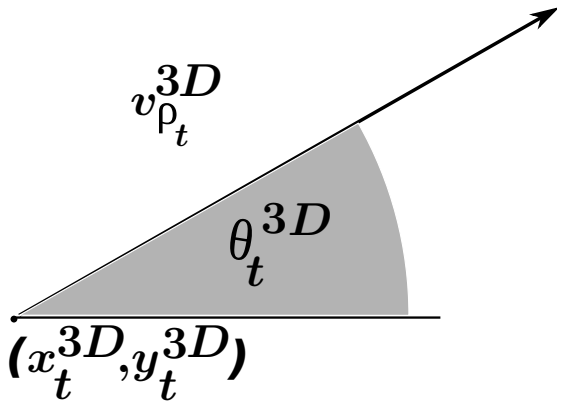


Figura 4.16: Generica particella al tempo t.

cui campionare la prossima particella:

$$\begin{aligned} \rho_{\text{camp}} &\sim \log\mathcal{N}\left(\log(\dot{\rho}_t), \sigma_\rho\right) \\ \theta_{\text{camp}} &\sim \mathcal{N}\left(\theta_t^{3D}, \sigma_\theta\right) \end{aligned}. \tag{4.25}$$

Utilizziamo la lognormale per modellare ρ_{camp} per questioni analoghe a quelle esposte nel Paragrafo 4.5.1. Il valore medio dello spostamento della nuova particella deve essere pari al modulo della velocità stimata $\dot{\rho}_t$ per la prima assunzione fatta in precedenza, a differenza di quanto fatto con il sistema Viterbi (Figura 4.17). A questo proposito approfondiamo le differenze tra i termini a priori usati nei due sistemi nel Paragrafo 4.5.3.

Se il veicolo è fermo non modelliamo ρ_{camp} con una distribuzione esponenziale come in Viterbi per i motivi che presentiamo nel Paragrafo 4.5.3. Inoltre non possiamo modellare ρ_{camp} con la lognormale in (4.25). Vogliamo infatti un campione che con alta probabilità si attesti attorno a 0. D'altra parte vogliamo che la prossima particella abbia qualche opportunità di modellare una possibile ripartenza del veicolo. Quindi abbiamo scelto di utilizzare ancora una distribuzione lognormale traslata di $-\tau_{fermo}$ in modo che la media sia nulla. Otteniamo dunque:

$$(\rho_{\text{camp}} + \tau_{fermo}) \sim \log\mathcal{N}\left(\log(\dot{\rho}_t), \sigma_{\rho_{\text{fermo}}}\right), \quad \text{con } \sigma_{\rho_{\text{fermo}}} > \sigma_\rho. \tag{4.26}$$

In questo modo si ottiene la distribuzione in Figura 4.18. L'angolo θ_{camp} viene modellato come in (4.25).

Noti ρ_{camp} e θ_{camp} possiamo calcolare quindi la pose della nuova particella:

$$s_{t+1} \begin{cases} x_{t+1}^{3D} = x_t^{3D} + \rho_{\text{camp}} \cos(\theta_{\text{camp}}) \\ y_{t+1}^{3D} = y_t^{3D} + \rho_{\text{camp}} \sin(\theta_{\text{camp}}) \\ \theta_{t+1}^{3D} = \theta_{\text{camp}} \\ \dot{\rho_{t+1}} = \rho_{\text{camp}} \cos(\theta_{\text{camp}}) \end{cases}. \tag{4.27}$$

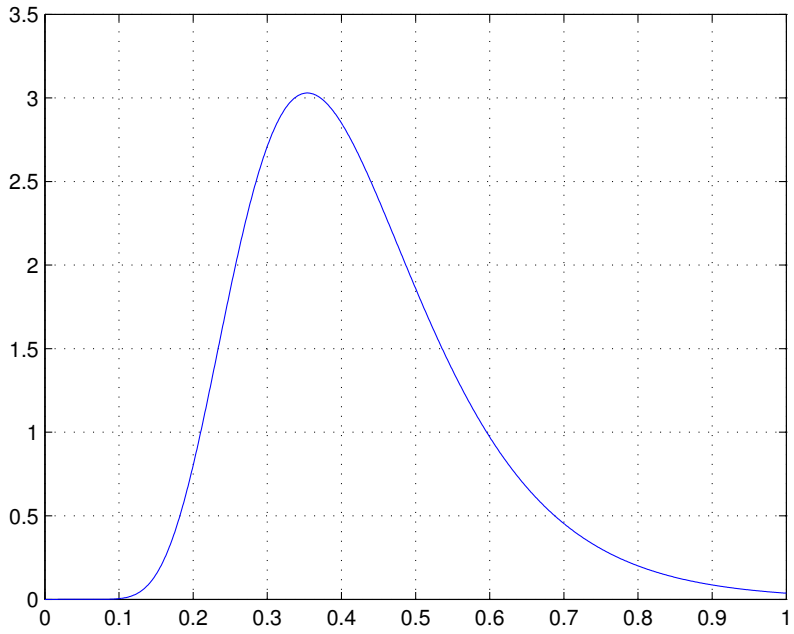


Figura 4.17: Distribuzione di ρ_{xy} con veicolo in moto, $\rho_t^i = 0.4$, $\sigma_\rho = 0.35$.

La presenza di una o più camere non modifica il comportamento della funzione: a differenza della verosimiglianza che dipende dalla misura attuale con cui confrontare lo stato, ovvero dipende dall'immagine relativa ad ogni telecamera; il calcolo del termine a priori avviene nel mondo, sul piano della rotonda.

4.5.3 Confronto tra i termini a priori

I due algoritmi non adottano le medesime distribuzioni per definire il termine a priori $p(s_{t+1}|s_t)$. Questo potrebbe apparire incoerente con il fatto che in entrambi i casi stiamo modellando il moto di un veicolo. La differenza di scelte è dovuta a due fattori: il diverso ruolo svolto dal termine a priori per i due algoritmi, i diversi metodi di stima della velocità. Inoltre, sia per Viterbi sia per il particle smoother abbiamo testato varie combinazioni di distribuzioni e quelle presentate danno sperimentalmente risultati migliori.

Nell'algoritmo basato su Viterbi devono essere valutati due campioni successivi che rappresentano gli stati s_t e s_{t+1}. Con il particle smoother invece per ogni particella che rappresenta lo stato s_t vogliamo generare la prossima particella che rappresenta lo stato s_{t+1} in accordo con la distribuzione del termine a priori. Nei due casi quindi il termine a priori svolge due funzioni differenti: per Viterbi viene utilizzato per valutare l'associazione

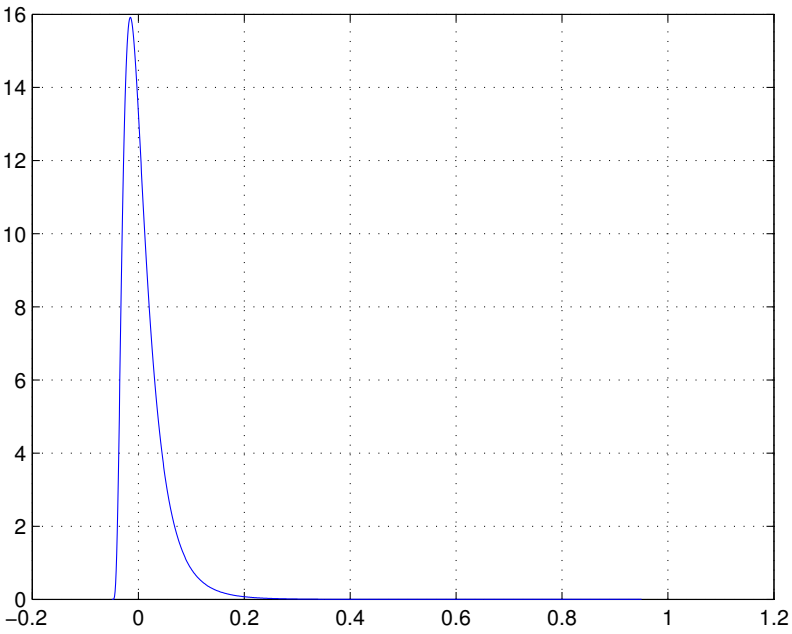


Figura 4.18: Distribuzione di ρ_{xy} con veicolo fermo, $\rho_t^i = \tau_{fermo}$, $\sigma_\rho = 0.6$.

tra due campioni, per il particle smoother viene utilizzato per generare una particella a partire dalla precedente. Consideriamo il caso in cui il veicolo è in moto per illustrare cosa comporta ciò nella scelta delle distribuzioni utilizzate. Supponiamo che all'istante t, la stima della velocità $\dot{\rho}_t$ sia pari a un valore $\bar{\dot{\rho}}_t$ per entrambi gli algoritmi. Per il sistema basato su Viterbi vogliamo che la probabilità $p(s_{t+1}|s_t)$ sia tanto più elevata quanto più il veicolo si sia spostato di una quantità pari a $\bar{\dot{\rho}}_t$, ovvero vogliamo che la moda della distribuzione di $p(s_{t+1}|s_t)$ coincida con $\bar{\dot{\rho}}_t$. Per il particle smoother, invece, vogliamo generare le particelle al tempo $t+1$, tramite il termine a priori, in modo che lo spostamento medio rispetto alla posizione s_t sia pari a $\bar{\dot{\rho}}_t$. Dato che per la distribuzione lognormale media e moda non coincidono, le lognormali definite per i due algoritmi hanno il valore della media differente. Per il sistema basato su Viterbi la media è tale che la moda sia $\bar{\dot{\rho}}_t$; nel particle smoother la media è pari a $\bar{\dot{\rho}}_t$. In Figura 4.19 vediamo come variano le due distribuzioni con un medesimo valore di $\bar{\dot{\rho}}_t = 0.4$ per i due algoritmi.

In realtà non sussiste solo questa differenza pratica tra i due casi.Le stime dello spostamento-velocità all'istante t da cui partiamo per calcolare il termine a priori per Viterbi e per generare le particelle per il particle smoother sono relative ad istanti differenti. Viterbi utilizza una stima della

velocità che indica quanto il veicolo si sposta tra t e $t+1$, il particle smoother invece utilizza la stima salvata nello stato s_t di ogni particella e calcolata nell'iterazione precedente del filtro. Le distribuzioni utilizzate per Viterbi per calcolare il modulo dello spostamento nel caso di veicolo fermo o in moto, modellano quindi l'incertezza della stima del modulo della velocità. Inoltre la velocità stimata con Viterbi dipende dalle misure del tracker 2D, rendendo dunque il termine a priori dipendente dalle misure. Per il particle smoother, invece, le distribuzioni dello spostamento modellano un termine aleatorio che si somma alla stima della velocità dello stato precedente e che consideriamo come accelerazione.

Alla luce di queste osservazioni risulta chiaro come i due termini a priori non possono essere modellati con le stesse distribuzioni. Inoltre, le osservazioni mettono in luce che il metodo per calcolare il termine a priori per l'algoritmo basato su Viterbi non è completamente corretto a livello teorico. In futuro si potrebbe valutare la possibilità di stimare la velocità per questo algoritmo in modo differente. Per esempio, mentre viene eseguito l'algoritmo di Dijkstra, per ogni j-esimo campione s_t^j si potrebbe salvare il modulo dello spostamento dall'i-esimo campione s_{t-1}^i a s_t^j dove s_{t-1}^i è il campione che precede s_t^i nel cammino del grafo generato da Dijkstra.

Per questi motivi, anche nel caso di veicolo fermo le distribuzioni che modellano lo spostamento per i due sistemi sono differenti.

4.6 Euristiche

Presentiamo ora due euristiche, una per ognuno dei due sistemi, che sono basate sullo stesso principio e permettono di ottenere un significativo miglioramento dell'efficienza dei due algoritmi. L'idea è, per entrambe le euristiche, quella di anticipare la scelta del campione classe-scala in $modelDimension$ per evitare di eseguire interamente l'algoritmo per tutti i campioni.

4.6.1 Algoritmo basato su Viterbi

Nel caso dell'algoritmo basato su Viterbi, rendiamo più efficiente l'implementazione dell'algoritmo di tracking diminuendo lo spazio di ricerca dell'algoritmo di Dijkstra. Dopo la fase di campionamento per ogni frame calcoliamo per ogni classe-scala la media delle probabilità di tutti i campioni in tutti gli istante. Il valore risultante è un indice di quanto il campione classe-scala stimi bene la dimensione reale del veicolo. Eseguiamo quindi l'algoritmo di Dijkstra solo per il grafo corrispondente alla classe-scala con probabilità media maggiore.

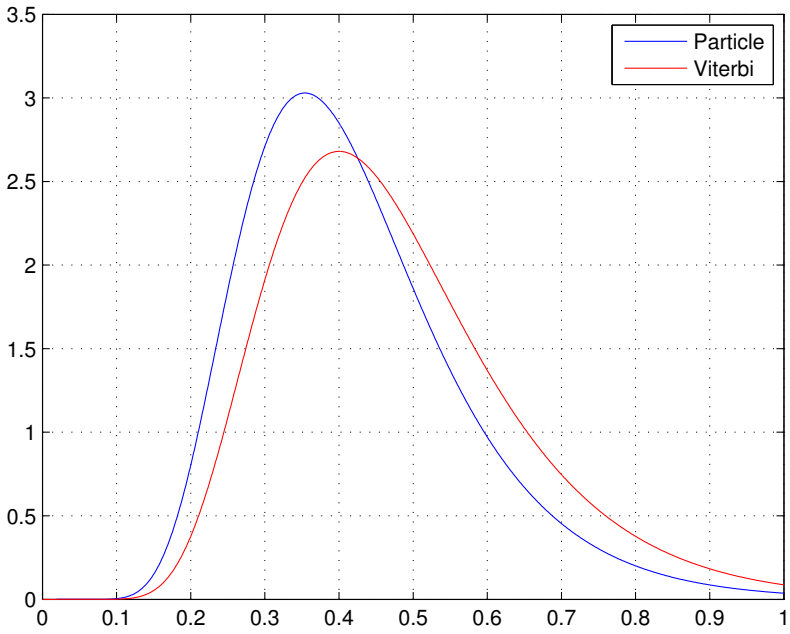


Figura 4.19: Confronto tra le due distribuzioni di ρ_{xy}.

Quindi,

$\ldots$;
Scelta della miglior sequenza con Viterbi;
foreach m *in* $modelDimension$ **do**
    $[pose3DVeicoloM(m), probCammino(m)] =$
    $Dijkstra2(\{\mathbf{s}_{1,m}, \ldots, \mathbf{s}_{T,m}\}, \{verCamp_{(1,1,m)}, \ldots, verCamp_{(n,T,m)}\})$;
Sceglie il modello migliore;
$M_{best} = \arg\max_m \{probCammino(m)\}$;
$pose3DVeicolo = pose3DVeicoloM(M_{best})$;

---

diventa:

$\ldots$;
Scelta della miglior sequenza con Viterbi;
Sceglie il modello migliore;
foreach m *in* $modelDimension$ **do**
    $mediaPose(m,t) = media(verCamp_{(:,:,m)})$;
$M_{best} = \arg\max_m \{mediaPose(m)\}$;
$pose3DVeicolo =$
$Dijkstra2(\{\mathbf{s}_{1,M_{best}}, \ldots, \mathbf{s}_{T,M_{best}}\}, \{verCamp_{(1,1,M_{best})}, \ldots, verCamp_{(n,T,M_{best})}\})$;

4.6.2 Particle smoother

L'algoritmo del particle smoothing presentato svolge la ricorsione avanti e indietro utilizzando tutti i campioni classe-scala. Un modo per ridurre il tempo di esecuzione dell'algoritmo è quello di eliminare alcuni campioni classe-scala e non fare tracking su di essi. Presupponiamo per fare ciò che le particelle estratte dal filtro relativo al campione classe-scala con le dimensioni migliori abbiano in media probabilità maggiore rispetto alle altre. Calcoliamo quindi la media delle probabilità delle particelle ad ogni istante per ogni classe-scala fino alla fine della ricorsione in avanti; successivamente scegliamo la classe-scala con la somma di queste medie maggiore ed eseguiamo la ricorsione all'indietro solo su questa classe-scala.

In teoria avremmo potuto fare questa scelta in qualsiasi istante, da cui poi avremmo ridotto il tracking alla sola classe-scala estratta. Se decidessimo di applicare l'euristica troppo presto, saremmo poco confidenti sulla classe-scala trovati, se troppo tardi vanificheremmo lo scopo di ridurre i tempi di computazione. Per questo motivo abbiamo scelto di applicarla alla fine della ricorsione in avanti in modo da considerare almeno una volta tutti gli istanti in cui appare il veicolo.

Quindi,

```
...;
Ricorsione all'indietro: inizializzazione con le ultime particelle;
foreach s_i in s_(T,m) do
    s_i.θ = −s_i.θ ;
Ricorsione all'indietro;
for t = T − 1 to 1 do
    foreach m in modelDimension do
        s_(t,m) = particleFilter(s_(t−1,m), verCamp_(·,t−1,m));
        foreach s_i in s_(t,m) do
            verCamp_(s_i,t,m) = calcolaVerosimiglianza(s_i, b_t, m);
...;
correggiOrientamenti(s_(:,m));
foreach m in modelDimension do
    for t = 1 to T do
        storiePose(m, t) = poseMedia(s_(t,m));
        mediaTemp(m, t) = media(verCamp_(:,t,m));
    mediaPose(m) = media(mediaTemp(m, :));
Sceglie il modello migliore;
M_best = arg max_m{mediaPose(m)};
pose3DVeicolo = storiePose(M_best, :);
```

---

diventa:

```
...;
foreach m in modelDimension do
    for t = 1 to T do
        storiePose(m, t) = poseMedia(s_(t,m));
        mediaTemp(m, t) = media(verCamp_(:,t,m));
    mediaPose(m) = media(mediaTemp(m, :));
Sceglie il modello migliore;
M_best = arg max_m {mediaPose(m)};
Ricorsione all'indietro: inizializzazione con le ultime particelle;
foreach s_i in s_(T,M_best) do
    s_i.θ = −s_i.θ ;
Ricorsione all'indietro;
for t = T − 1 to 1 do
    s_(t,M_best) = particleFilter(s_(t−1,M_best), verCamp_(·,t−1,M_best));
    foreach s_i in s_(t,M_best) do
        verCamp_(s_i,t,M_best) = calcolaVerosimiglianza(s_i, b_t, M_best);
...;
correggiOrientamenti(s_(:,M_best));
pose3DVeicolo = storiePose(M_best, :);
```

Capitolo 5

Risultati Sperimentali

In questo capitolo mostriamo i risultati sperimentali ottenuti con i due algoritmi di tracking proposti.

L'obiettivo degli algoritmi è quello di migliorare i dati estratti dal tracker 2D, rendendo la stima della pose del veicolo più precisa possibile. Per dare una valutazione significativa, non vogliamo limitarci a osservare qualitativamente quanto il tracker insegue bene i veicoli e quanto il modello proiettato sull'immagine rappresenti fedelmente il moto del veicolo. Concentriamo la nostra attenzione soprattutto a confrontare le stime delle pose con le pose reali. In generale questo non è possibile perché nella realtà non conosciamo la pose dei veicoli, anche perché se fosse altrimenti sarebbero inutili i sistemi proposti. Agiamo quindi eseguendo due differenti esperimenti: testiamo i tracker su dati simulati in cui il veicolo assume pose note, e tracciamo uno specifico veicolo che è stato dotato di strumenti di misura che ne calcolino la pose. In quest'ultimo caso presentiamo anche i risultati del tracker 2D per verificare se effettivamente i nostri algoritmi ne migliorano le stime.

5.1 Dati Simulati

Un primo modo di valutare con precisione i risultati degli algoritmi di tracking è quello di creare un filmato in cui viene ripreso un veicolo che assume pose note, inseguirlo e confrontare le pose stimate con quelle note. Testiamo qui i sistemi di tracking su una singola telecamera, mentre i risultati dei sistemi nel caso multicamera vengono discussi nel prossimo paragrafo.

Il veicolo che utilizziamo per generare le misure è il modello a forma di parallelepipedo con dimensioni $\{l, a, p\} = \{4, 1.4, 1.7\}$, che coincidono con le dimensioni di riferimento del campione che rappresenta le automobili. Abbiamo fatto due esperimenti. Nel primo scegliamo due serie di pose in modo che vengano rappresentate due situazioni differenti: in una il veicolo ha moto circolare uniforme, nell'altra ha moto rettilineo, durante il quale frena, staziona e accelera. Il primo approssima il moto di un veicolo che ha impe-

gnato la rotatoria, il secondo rappresenta invece il comportamento durante l'immissione in rotonda. Per generare i filmati proiettiamo sull'immagine il modello collocato in accordo con le pose scelte. Per la proiezione adottiamo tre differenti matrici di calibrazione per considerare diversi punti di vista. In Figura 5.3 mostriamo per ogni colonna una sequenza di frame ottenuti proiettando il modello adottando tre prospettive differenti, ad ogni riga corrispondono le proiezioni della medesima pose. Nel secondo esperimento generiamo il filmato adottando la prospettiva di una delle telecamere reali e, utilizzando i dati noti sul raggio della rotatoria reale, simuliamo il passaggio di un veicolo ideale per il centro della corsia della rotatoria. Otteniamo dunque sette filmati sui quali eseguiamo il tracker 2D. I blob e le traiettorie risultanti, sono gli ingressi dei due sistemi proposti. Confrontiamo infine il risultato con le pose a partire dalle quali abbiamo creato i filmati.

Ciò che otteniamo da questo confronto è un'indicazione sulle prestazioni massime degli algoritmi. Utilizzando il filmato simulato non ci sono infatti disturbi sulle misure e il veicolo tracciato coincide proprio con il modello utilizzato negli algoritmi. Siamo quindi in un caso ideale e dei buoni risultati con questo setup sono una condizione necessaria per ottenere un buon tracking dei veicoli nel caso reale, in cui le misure dei veicoli sono affette da rumore, e i veicoli stessi non hanno una forma identica al modello utilizzato. In particolare, nel secondo esperimento, dato che utilizziamo i dati che modellano la rotonda reale, otteniamo un'indicazione di quanto preciso può risultare il tracking su questa rotatoria, con la prospettiva adottata.

Nei test, per ogni campione classe-scala, il tracker con il particle smoother utilizza 250 particelle; per ottenere risultati paragonabili, il sistema che utilizza Viterbi lavora con 500 campioni per frame.

In Appendice B mostriamo gli errori di stima nel tempo di ogni componente della pose rispetto alle pose da cui abbiamo generato i filmati. In Figura B.1, B.2 e B.3 mostriamo gli errori per i filmati che riprendono il moto circolare uniforme rispettivamente dalle tre differenti prospettive; in Figura B.4, B.5 e B.6 mostriamo invece i risultati per il caso di moto rettilineo. In Figura B.7 e B.8 riassumiamo i risultati per i due tipi di moto sotto forma di istogramma. Infine in Figura B.9 e B.10 mostriamo i grafici e l'istogrammi degli errori per il secondo esperimento . Riportiamo come esempio due grafici, uno che rappresenta l'andamento dell'errore (Figura 5.1) e uno che rappresenta l'istogramma dell'errore (Figura 5.2). Il grafico in Figura 5.1 rappresenta l'errore di stima delle x nel tempo compiuto dal particle smoother il veicolo in moto circolare uniforme ripreso dalla prima prospettiva. In ascisse è indicato il numero di frame dell'errore e in ordinate il valore dell'errre. I risultati, come si può immaginare, sono tanto migliori quanto più i valori si attestano nell'intorno dello zero. L'istogramma in Figura 5.2 rappresenta invece la distribuzione degli errori di stima compiuti dal particle smoother per il moto circolare uniforme considerando gli errori compiuti su tutte le prospettive. Questo istogramma indica la qualità della

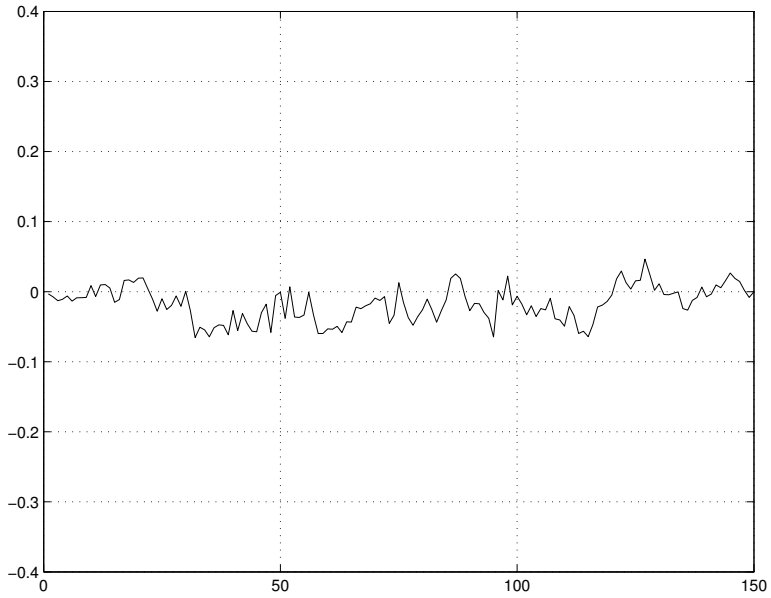


Figura 5.1: Errore sulle x del tracking del veicolo in moto circolare uniforme dalla prima prospettiva con il particle smoother.

stima ottenuta rispetto ai dati veri. Un istogramma che rappresenta una buona stima è centrato attorno allo 0, il che indica che l'errore non è polarizzato. inoltre, quanto più l'istogramma approssima bene una Gaussiana, e quanto più la varianza della Gaussiana è bassa, tanto più le stime sono migliori.

La Tabella 5.1 per il primo esperimento, e la Tabella 5.2 riassume i valori della media del valore assoluto (media ABS), della media, della mediana, della MAD, della deviazione standard e dell'intervallo interquantile (IQR) dell'errore compiuto dal particle filter (P) e dall'algoritmo Viterbi-based (V) per il moto rettilineo (R) e circolare (C). Per MAD intendiamo il Median Absolute Deviation, ovvero il valore calcolato come $mediana(|x_i - mediana(\mathbf{x}|_{\forall i}))$ per un qualsiasi vettore $\mathbf{x}$ di valori. Questa cifra dà un'indicazione della dispersione dei valori: valori minori sono preferibili perché indicano una minore dispersione dell'errore. Anche l'IQR (interquartile range) è una cifra che valuta la dispersione degli errori, e indica l'ampiezza dell'intervallo che contiene la metà centrale degli errori, e è calcolato come $IQR = Q_{\frac{3}{4}} - Q_{\frac{1}{4}}$ dove, se con F indichiamo la funzione di ripartizione della distribuzione dell'errore, $Q_{\frac{3}{4}} = F(\frac{3}{4})$ e $Q_{\frac{1}{4}} = F(\frac{1}{4})$ sono rispettivamente il terzo e il primo quartile della distribuzione dell'errore.

Il particle smoother dà risultati notevolmente migliori in tutte le prove effettuate sui sette filmati simulati: gli errori medi sono nell'ordine di pochi centimetri per le componenti x e y e di pochi decimi di grado per θ in entrambe le condizioni di moto; inoltre l'errore è limitato nell'intorno dello

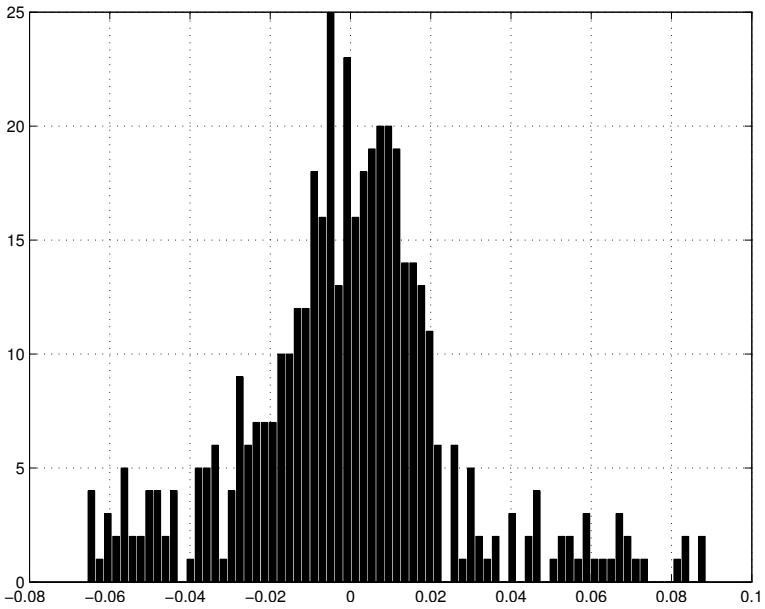


Figura 5.2: Istogramma degli errori sulle x per il tracking per il moto circolare uniforme.

Tabella 5.1: Tabella riassuntiva degli errori sui dati simulati del primo esperimento

		media ABS	media	SDev	mediana	MAD	IQR
VR	$x(m)$	0.048	−0.008	0.063	−0.006	0.037	0.069
	$y(m)$	0.036	−0.012	0.045	−0.010	0.030	0.058
	$\theta(deg)$	1.557	−0.525	1.960	−0.181	1.303	2.814
PR	$x(m)$	0.021	0.009	0.028	0.008	0.014	0.029
	$y(m)$	0.014	0.004	0.018	0.002	0.012	0.023
	$\theta(deg)$	0.550	0.097	0.676	0.063	0.470	0.944
VC	$x(m)$	0.068	−0.050	0.073	−0.037	0.051	0.102
	$y(m)$	0.064	−0.027	0.083	−0.008	0.047	0.111
	$\theta(deg)$	3.051	−2.590	2.724	−2.389	1.812	3.736
PC	$x(m)$	0.019	−0.001	0.027	0.000	0.013	0.026
	$y(m)$	0.027	−0.001	0.035	−0.001	0.022	0.044
	$\theta(deg)$	0.642	0.189	0.808	0.194	0.482	0.982

Tabella 5.2: Tabella riassuntiva degli errori sui dati simulati del secondo esperimento

		media ABS	media	SDev	mediana	MAD	IQR
V	$x(m)$	0.064	0.026	0.080	0.036	0.042	0.090
	$y(m)$	0.079	-0.029	0.114	-0.039	0.055	0.113
	$\theta(deg)$	4.983	-4.019	4.124	-4.137	1.281	2.478
P	$x(m)$	0.054	-0.018	0.081	-0.027	0.022	0.050
	$y(m)$	0.036	0.005	0.045	0.004	0.023	0.046
	$\theta(deg)$	3.583	-0.879	16.471	0.803	1.176	2.459

zero. Questo significa che il particle smoother, indipendentemente dalla prospettiva utilizzata e dal moto da stimare offre delle ottime prestazioni sul filmato simulato.

Il sistema basato su Viterbi, pur ottenendo degli errori medi bassi, nell'ordine dei centimetri per le coordinate x e y, e di pochi gradi per θ, produce stime spesso polarizzate e più rumorose rispetto a quelle ottenute con il particle smoother. Le stime dell'orientamento soffrono di errori maggiori, e la loro distribuzione ha una varianza molto elevata.

Inoltre, nelle Tabelle 5.1 e 5.2, si può notare una significativa differenza di valori, soprattutto sull'orientamento. Questo è probabilmente legato al fatto che la velocità di percorrenza del veicolo è differente nei due casi. Nel primo esperimento abbiamo scelto una velocità di 0.3 $\frac{m}{frame}$, ovvero approssimativamente di 26 $\frac{km}{h}$; per il secondo, ci siamo basati sulle stime delle velocità dei tracker eseguiti sui veicoli reali, e abbiamo scelto invece un valore di 0.5 $\frac{m}{frame}$, ovvero 46 $\frac{km}{h}$. A velocità maggiori i cambi di orientamento e posizione sono più rapidi, e, di conseguenza, i due tracker, in particolar modo quello basato su Viterbi, sono affetti da errori più elevati, soprattutto sull'orientamento.

Alla luce della discussione sull'ottimalità dell'algoritmo di Viterbi del Paragrafo 3.6 ci si sarebbe potuti aspettare dei risultati per lo meno simili tra i due algoritmi, se non migliori per il caso con Viterbi. Il motivo per cui Viterbi offre una stima peggiore è dovuto al fatto che sceglie per un istante un solo campione, che, se da una parte è quello migliore per il computo della probabilità a posteriori, dall'altra rappresenta una singola pose. Il particle smoother invece calcola la stima della pose come media delle particelle generate ad ogni istante. Quindi, gli errori causati dalle approssimazioni del modello del moto si compensano, dando come risultato una serie di pose maggiormente smussata. Inoltre, come abbiamo visto, i campioni generati da Viterbi non sono "guidati" dal moto come per il particle smoother. Questo implica che, per ottenere buoni risultati il sistema basato su Viterbi deve utilizzare un numero maggiore di campioni.

I tempi di esecuzione variano sensibilmente nei due casi: il particle smoother impiega per ogni traiettoria $6\frac{s}{frame}$ a frame, il sistema Viterbi-based $13\frac{s}{frame}$. Ovviamente sono entrambi inefficienti, principalmente a causa del fatto che abbiamo utilizzato Matlab per l'implementazione, ma il particle smoother impiega la metà del tempo, e la complessità è lineare rispetto al numero di campioni, mentre con Viterbi la complessità è quadratica.

5.2 Dati Reali

Dalle indicazioni ottenute con gli esperimenti del paragrafo precedente ci aspettiamo anche per il caso reale che le stime ottenute con il particle smoother siano migliori rispetto a quelle ottenute con il sistema che utilizza Viterbi. Presentiamo l'organizzazione dell'esperimento e successivamente analizziamo i risultati.

5.2.1 Organizzazione dell'esperimento

L'esperimento condotto per valutare il tracking è analogo a quello in [45]. La scena oggetto di tracking è un'intersezione rotatoria con quattro immissioni. Il diametro esterno della rotonda è di 48 metri e la larghezza della corsia della rotonda è di 11 metri. Sono state utilizzate tre telecamere utilizzate per riprendere la rotatoria disposte sopra il centro della rotonda. Le tre telecamere riprendono parzialmente la rotonda e alcune zone della scena sono inquadrate da più di una telecamera.

I dati che utilizzeremo per il confronto sono stati raccolti da un sistema RTK-GPS il quale permette una stima precisa della posizione del veicolo e una IMU dedicata a raccogliere i dati sull'orientamento del veicolo. Il sistema RTK-GPS, la IMU e le tre telecamere sono connesse tra loro tramite un PC che associa un timestamp a ogni misura proveniente dai sensori e a ogni frame catturato dalle telecamere. Questo permette in una fase successiva alle riprese di sincronizzare i dati estratti dalle tre telecamere e dai sensori RTK-GPS e IMU.

Delle tre telecamere, ne consideriamo solo due perché per la terza la calibrazione calcolata non è adeguatamente precisa. Due telecamere sono comunque sufficienti per discutere i risultati dell'algoritmo per il caso multicamera.

5.2.2 Risultati

Il confronto con i dati reali consiste nel confrontare le posizioni RTK-GPS e l'orientamento dato dall'IMU con le pose estratte dal tracker 3D al medesimo istante. Per fare questo quindi telecamera, RTK-GPS e IMU sono stati sincronizzati. Quest'operazione introduce degli errori che sono dovuti a ritardi tra le sincronizzazioni di telecamera e RTK-GPS e IMU e nel caso

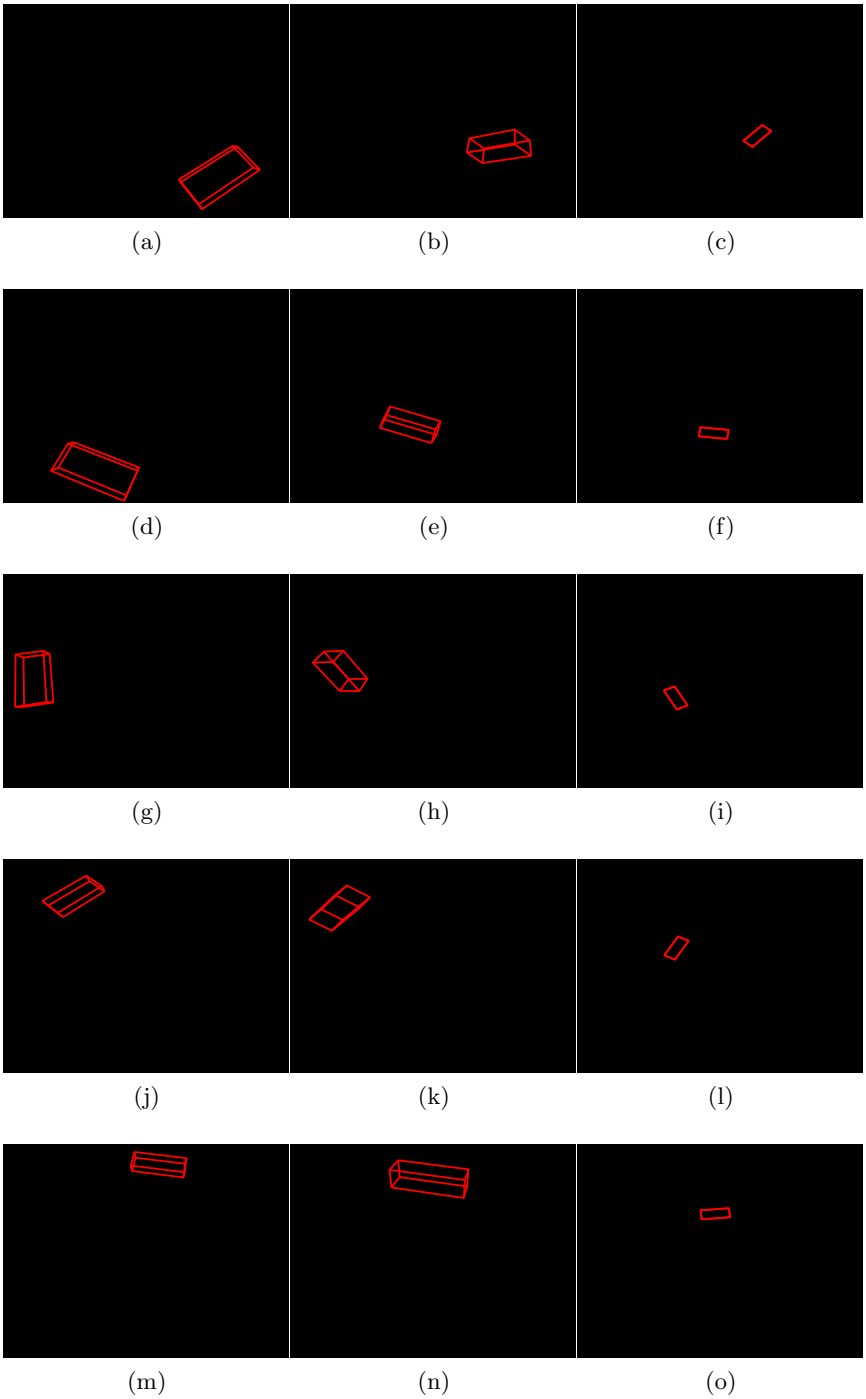


Figura 5.3: Per ogni colonna vengono degli esempi di proiezione del modello per le tre differenti prospettive.

a più camere, anche tra le telecamere stesse. Inoltre, i dati RTK-GPS e, in misura inferiore i dati IMU sono affetti da rumore. Un ultimo problema relativo alla stima delle posizioni deriva dal fatto che in alcune zone dell'immagine gli effetti di distorsione possono non essere compensati a sufficienza dalla funzione vista nel Paragrafo 4.2.2 e quindi i punti 3D corrispondenti a queste zone dell'immagine non sono confrontabili con i dati dell'RTK-GPS che, non soffrono di questo disturbo essendo indipendenti dall'immagine. Le prime problematiche dovute alla sincronizzazione non possono essere risolte, invece, per evitare il problema della stima in zone distorte dell'immagine escludiamo queste ultime dal computo totale degli errori.

Prima di analizzare i dati osserviamo infine che la frequenza con cui vengono elaborati i dati RTK-GPS è minore di quella di funzionamento della telecamera. Questo implica il fatto che il confronto avviene tra la posizione RTK-GPS e la posizione stimata più vicina temporalmente. Viene introdotta anche in questo caso un'approssimazione che disturba il confronto tra i risultati. I dati provenienti dalla IMU hanno invece una frequenza maggiore di quella della telecamera, quindi dobbiamo scegliere tra i dati della IMU quelli più vicino temporalmente alle stime. Anche in questo caso quindi introduciamo un errore, che risulterà comunque inferiore a quello introdotto con l'RTK-GPS, grazie al fatto che la frequenza di campionamento è maggiore. Inoltre, mentre nel caso dell'RTK-GPS dobbiamo scartare una parte consistente delle nostre stime per il confronto, con la IMU scartiamo i dati provenienti dalla IMU stessa, considerando quindi tutte le stime prodotte dai tracker.

In Appendice C mettiamo a confronto i dati ottenuti con entrambe i tracker sia nel caso a singola telecamera e nel caso multicamera attraverso i grafici e gli istogrammi degli errori, analogamente a quanto fatto per il caso simulato. I dati risultano spesso polarizzati come si vede per esempio in Figura 5.4 su entrambe le coordinate e sull'orientamento. La causa di questi è da imputare principalmente ai disturbi sulle misure dei veicoli. I blob che rappresentano la regione dell'auto includono spesso una porzione di ombra, quindi la regione è maggiore di quella occupata realmente dal veicolo e dal modello proiettato sull'immagine. In questi casi, variazioni anche non trascurabili della pose stimata rispetto alla pose reale, non variano significativamente il valore della verosimiglianza e il termine a priori non è sempre sufficiente a compensare questo disturbo, provocando quindi l'errore di polarizzazione notato. Questo problema, quando riguarda le posizioni, è riconducibile anche a quanto affermato riguardo l'imprecisione della sincronizzazione. Quando esiste un ritardo di pochi decimi di secondo tra telecamera e rilevatore RTK-GPS le posizioni sono infatti sfasate tra loro. Questo problema è più evidente nelle zone in cui il veicolo è più veloce, dato che percorre uno spazio maggiore durante lo stesso tempo corrispondente al ritardo di sincronizzazione.

Nelle ultime pagine dell'Appendice C, mostriamo inoltre gli istogrammi

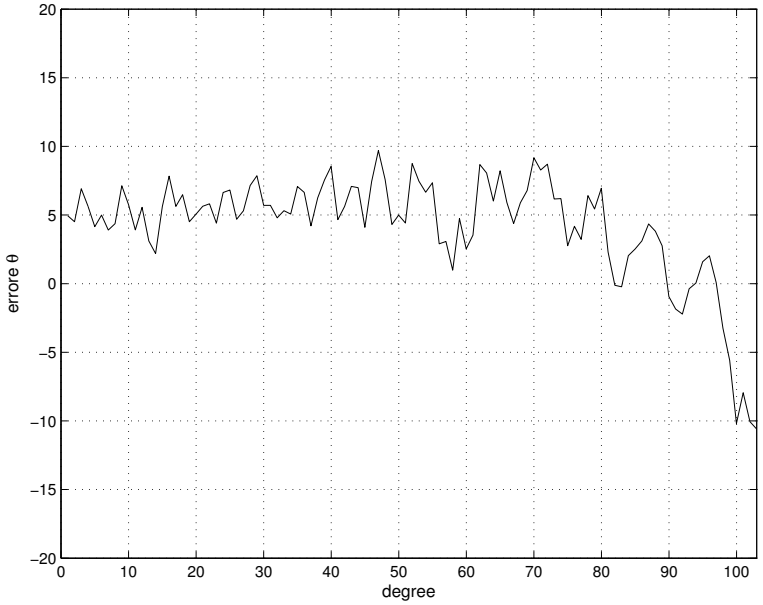


Figura 5.4: Esempio di grafico con errore polarizzato.

degli errori su tutte le traiettorie considerate, sia per il caso a telecamera singola che a telecamera multipla. I dati riassuntivi delle stime su tutte le traiettorie sono riportati in Tabella 5.1 dove indichiamo con PS e PM il particle smoother a singola e multipla telecamera, e con VS e VM il sistema basato su Viterbi nel caso a singola e multipla camera. Riportiamo la media dell'valore assoluto, la media, la mediana, la deviazione standard, la MAD e lo scarto interquartile (IQR) dell'errore. In Tabella 5.4 riportiamo gli errori ottenuti dal tracker 2D dalle cui stime eseguiamo il tracking 3D.

Rispetto al caso simulato abbiamo errori maggiori per tutte le componenti dello stato stimato. Possiamo però notare che gli errori sulle coordinate x e y non superano il metro (caso pessimo) e sull'orientamento non superano i 15 gradi eccetto casi eccezionali per entrambi. Nel caso a singola telecamera si nota come, in corrispondenza dei primi e degli ultimi istanti il valore degli errori è spesso differente rispetto alla zona centrale dei grafici. Questo si verifica in tutti i casi in cui il veicolo passa dal bordi dell'immagine. Il bordo dell'immagine, nonostante le euristiche adottate per gestire i problemi discussi al Paragrafo 4.1, rimane comunque una zona in cui la misura del veicolo è solo parziale quindi la stima della posizione è affetta da un errore maggiore rispetto ad altre posizioni.

Nel caso multicamera i due tracker sono meno precisi nei punti corrispondenti al momento in cui il veicolo presente nella prima telecamera entra nell'immagine della seconda. Ciò è imputabile al fatto che, quando le due telecamere non sono sincronizzate perfettamente tra loro, anche le due mi-

Tabella 5.3: Tabella riassuntiva degli errori della stima prodotta dagli algoritmi proposti rispetto ai dati reali

		media ABS	media	SDev	mediana	MAD	IQR
VS	$x(m)$	0.343	-0.211	0.445	-0.180	0.236	0.468
	$y(m)$	0.330	-0.122	0.414	-0.136	0.316	0.500
	$\theta(deg)$	11.214	-1.928	22.734	-3.664	5.123	10.227
PS	$x(m)$	0.247	-0.125	0.264	-0.171	0.171	0.348
	$y(m)$	0.189	0.072	0.238	0.058	0.135	0.277
	$\theta(deg)$	4.882	3.900	4.942	4.156	2.446	4.875
VM	$x(m)$	0.327	0.115	0.382	0.154	0.305	0.512
	$y(m)$	0.241	-0.011	0.300	-0.041	0.241	0.384
	$\theta(deg)$	6.161	-4.653	6.885	-3.759	3.845	7.718
PM	$x(m)$	0.262	-0.004	0.313	-0.056	0.261	0.475
	$y(m)$	0.200	0.144	0.235	0.094	0.179	0.248
	$\theta(deg)$	7.592	5.322	10.330	2.401	7.629	8.465

Tabella 5.4: Tabella riassuntiva degli errori della stima del tracker 2D rispetto ai dati reali

	media ABS	media	SDev	mediana	MAD	IQR
$x(m)$	0.520	0.286	0.688	0.292	0.338	0.348
$y(m)$	0.789	0.159	2.0133	-0.317	0.284	0.575

sure dei veicoli non corrispondono perfettamente. Il tracker cerca una stima una posizione che aderisca nel miglior modo possibile ad entrambe le misure, ma se rappresentano il veicolo in due pose non perfettamente coincidenti, la stima sarà una pose approssimativamente media tra le due. Questo errore del tracking dovuto al ritardo introdotto dalla sincronizzazione è evidente per esempio nella Figura 5.5, dove in rosso sono disegnate le posizioni RTK-GPS, in blu le posizioni stimate e i segmenti neri esprimono le associazioni tra i dati. Dal momento in cui il veicolo entra nella seconda camera, il ritardo di sincronizzazione causa un ritardo anche nella posizione stimata. Un disturbo che si somma a questo deriva dal fatto che le due telecamere offrono due prospettive differenti quindi anche la porzione di ombra vista è differente. Siccome la background subtraction non elimina del tutto l'ombra, i due blob corrispondenti nelle due camere corrispondono al veicolo insieme a porzioni differenti di ombra. Questo aumenta dunque l'incertezza della stima della pose in questi punti.

In Figura 5.5 possiamo notare però che nei due punti in cui il veicolo esce dalla prima camera il tracker si comporta in modo analogo a quanto avviene nei punti precedenti e successivo. Questo avviene anche per le altre traiettorie e ci permette di dire che, seppur al costo di perdere precisione nella stima di alcune posizioni, il tracking multicamera è robusto rispetto ai problemi di stima della posizione ai bordi.

Per quanto riguarda il confronto tra gli errori delle stime del tracker 2D con quello proposto, possiamo confrontare solo gli errori di posizione, dato che il tracker 2D non stima l'orientamento del veicolo. Si può notare come l'errore medio e la mediana sia nel caso a singola telecamera, che a telecamera multipla per entrambi gli algoritmi proposti, sono inferiori a quelli ottenuti con il tracker 2D. Inoltre, dai valori della deviazione standard, della MAD e dell'IQR possiamo dedurre che, nel caso medio la dispersione degli errori è minore per gli algoritmi proposti. Questo confronto riguarda il tracking su un'automobile. Come abbiamo visto nel Paragrafo 3.2, però, gli errori del tracker 2D più evidenti che i sistemi proposti hanno lo scopo di superare si hanno nella stima della posizione dei camion, soprattutto quando entrano o escono nell'immagine. La stima data dai tracker 3D in corrispondenza dei bordi è nettamente migliore rispetto al caso 2D. In Figura 5.6 mostriamo il confronto tra la stima della posizione del camion nell'immagine effettuata dal tracker 2D (cerchio in Figura 5.6a), e la stima data dal tracker 3D (Figura 5.6b).

Per concludere il discorso sui risultati del tracking riportiamo in Tabella 5.5 tutte le misure degli errori viste finora. Riportiamo in tabella gli errori. Per ogni componente della pose riportiamo gli errori per il tracking dell'algoritmo basato su Viterbi (V) e del particle smoother (P) sui filmati simulati creati con condizioni analoghe a quelle reali (V Sim 3D e P Sim 3D), per il tracking 2D e per il tracking sui dati reali nel caso a singola telecamera (V SC 3D e P SC 3D) e a telecamera multipla(V MC 3D e P MC 3D). Abbia-

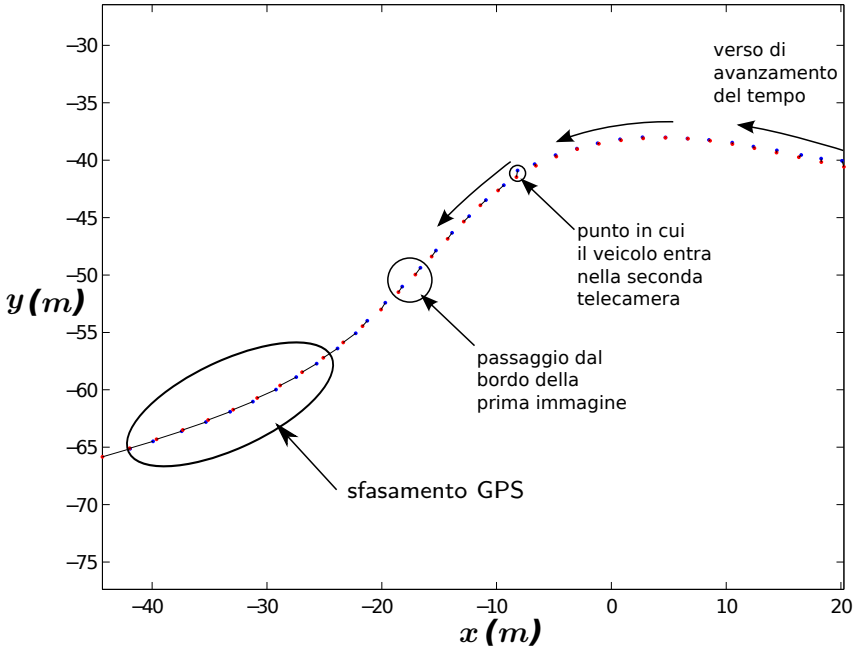
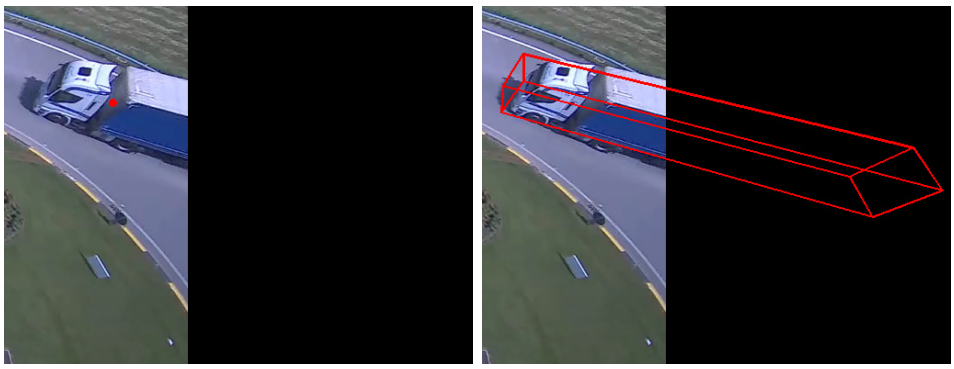


Figura 5.5: Confronto tra le posizioni stimate con l'RTK-GPS (in rosso) e le posizioni stimate dal tracker (in blu) temporalmente più vicine per il tracking multicamera.



(a) *Stima con il tracker 2D.* (b) *Stima con il tracker 3D.*

Figura 5.6: Errori del tracking del veicolo in moto rettilineo dalla terza prospettiva.

mo evidenziato in grassetto i valori migliori tra quelli trovati con i sistemi proposti e con il tracker 2D. non consideriamo nel confronto anche i dati sul filmato simulato, che ovviamente sono migliori eccetto la deviazione standard della stima dell'orientamento. Questo è legato a degli outlier ai bordi che compromettono il valore complessivo della deviazione standard. I valori della MAD, della mediana e dell'IQR sono in questo senso più indicativi, danno infatti un valore non corrotto dalla presenza di outlier.

Confrontando i risultati si nota come il tracking 3D permette di migliorare la stima della posizione dei veicoli rispetto a quella del tracker 2D. Questo miglioramento sarebbe ancora più evidente se, il veicolo tracciato non fosse un'autovettura, ma un camion, per cui l'effetto dell'errore esposto nel paragrafo 3.2. Come già sottolineato, risulta più chiaro da questa tabella che tra i due algoritmi il particle smoother dà risultati migliori. Notiamo inoltre che molti dei risultati evidenziati in grassetto sono relativi al tracking multicamera, nonostante l'errore di sincronizzazione tra le due telecamere si sommi all'errore di stima. Possiamo immaginare quindi che, con due telecamere perfettamente sincronizzate il tracking con particle smoother multicamera sia la migliore scelta.

5.3 Classificazione e stima delle dimensioni dei veicoli

Presentiamo ora una valutazione della capacità di classificazione dei due algoritmi. Abbiamo visto come classificano i veicoli scegliendo tra tre categorie di veicoli: automobile, camion e moto che sono rispettivamente la classe 1, 2 e 3. Nella realtà però esistono anche altri veicoli come gli autoarticolati, i furgoni, i van. Negli esperimenti fatti consideriamo mezzi pesanti, autoarticolati e camion appartenenti alla classe 2, i furgoni, van, e auto appartenenti alla classe 1 e le moto alla classe 3. Osserviamo che tra i veicoli che abbiamo elencato ci sono gli autoarticolati. Negli algoritmi proposti non gestiamo il grado di libertà che caratterizza gli autoarticolati. Approssimiamo le dimensioni e il comportamento di questi considerandoli come camion. Questo porta ad una stima meno precisa quando viene tracciato un autoarticolato.

I tempi di elaborazione piuttosto lunghi non ci permettono di eseguire degli esperimenti su numerose traiettorie 2D. Per offrire una valutazione significativa, abbiamo scelto di eseguire i due algoritmi su un frammento di video di durata di 6 minuti, in cui transitano veicoli di forme differenti. In Tabella 5.6 mostriamo i risultati della classificazione. I veicoli vengono tutti classificati correttamente a parte un'automobile, una Smart, che a causa delle sue piccole dimensioni viene classificata come moto. Si noti come entrambe i sistemi raggiungono risultati uguali: questo è dovuto al fatto che i modelli di veicolo utilizzati sono i medesimo e che il modulo in cui viene calcolata la verosimiglianza della proiezione del modello è comune.

Tabella 5.5: Tabella riassuntiva degli errori sulla rotatoria reale.

		media ABS	media	SDev	mediana	MAD	IQR
$x(m)$	**V Sim 3D**	0.064	0.026	0.080	0.036	0.042	0.090
	P Sim 3D	0.054	−0.018	0.081	−0.027	0.022	0.050
	2D	0.520	0.286	0.688	0.292	0.338	**0.348**
	V SC 3D	0.343	−0.211	0.445	−0.180	0.236	0.468
	P SC 3D	**0.247**	−0.125	**0.264**	−0.171	**0.171**	**0.348**
	V MC 3D	0.327	0.115	0.382	0.154	0.305	0.512
	P MC 3D	0.262	**−0.004**	0.313	**−0.056**	0.261	0.475
$y(m)$	**V Sim 3D**	0.079	−0.029	0.114	−0.039	0.055	0.113
	P Sim 3D	0.036	0.005	0.045	0.004	0.023	0.046
	2D	0.789	0.159	2.0133	−0.317	0.284	0.575
	V SC 3D	0.330	−0.122	0.414	−0.136	0.316	0.500
	P SC 3D	**0.189**	**0.072**	0.238	0.058	0.135	0.277
	V MC 3D	0.241	−0.011	0.300	**−0.041**	0.241	0.384
	P MC 3D	0.200	0.144	**0.235**	0.094	**0.179**	**0.248**
$\theta(deg)$	**V Sim 3D**	4.983	−4.019	4.124	−4.137	1.281	2.478
	P Sim 3D	3.583	−0.879	16.471	0.803	1.176	2.459
	2D	-	-	-	-	-	-
	V SC 3D	11.214	**−1.928**	22.734	−3.664	5.123	10.227
	P SC 3D	**4.882**	3.900	**4.942**	4.156	**2.446**	**4.875**
	V MC 3D	6.161	−4.653	6.885	−3.759	3.845	7.718
	P MC 3D	7.592	5.322	10.330	**2.401**	7.629	8.465

Tabella 5.6: Veicoli classificati correttamente.

	auto	**camion**	**moto**	**totale(%)**
Particle	37 su 38	8 su 8	3 su 3	98%
Viterbi	37 su 38	8 su 8	3 su 3	98%

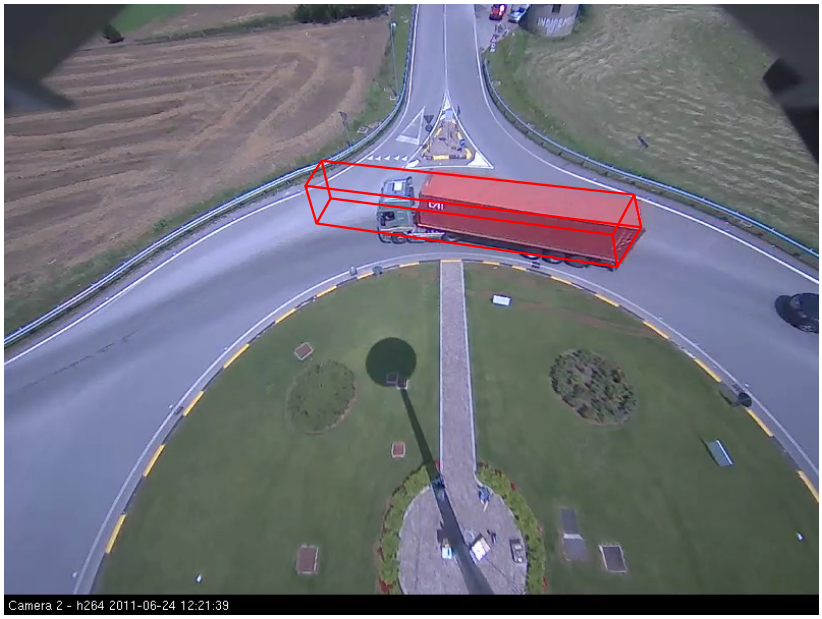


Figura 5.7: Esempio di errore compiuto nella stima delle dimensioni di un camion.

Non presentiamo in questa sede una valutazione della bontà della stima delle dimensioni dei veicoli. Questo perché, in primo luogo bisognerebbe conoscere le dimensioni reali dei veicoli, e in secondo luogo stimare con precisione le dimensioni esula dallo scopo di questa tesi. Abbiamo notato comunque che le auto e le moto sono approssimate bene dal parallelepipedo proiettato sull'immagine. L'altezza dei furgoni e van, viene approssimata per difetto, dato che il modello che li rappresenta utilizza una scala più consona alle dimensioni delle automobili. La lunghezza dei camion, a causa della varianza molto elevata, a volte viene approssimata abbastanza grossolanamente come in Figura 5.7, questo influisce indirettamente anche sulla precisione della stima della posisione del camion. Un metodo per superare questo problema consiste nell'aumentare il numero di dimensioni del modello per cui si esegue il tracking (6).

5.4 Efficacia del tracking

I due sistemi presentati basano il tracking sui risultati del tracker 2D: se quest'ultimo non esegue una associazione delle misure al filtro corretto, l'errore si ripercuoterà anche nel tracking 3D. Per dare una valutazione esclusivamente sui nostri algoritmi consideriamo quindi quei veicoli per cui la

traiettoria 2D è corretta. Il tracker che utilizza Viterbi, campionando frame per frame a partire dai centri che compongono la traiettoria 2D, non perde mai i veicoli tracciati. Quando la traiettoria è affetta da molto rumore, i campioni estratti nella prima fase possono essere poco significativi, costringendo quindi l'algoritmo di Viterbi ad una scelta tra pose molto differenti da quella reale.

Il sistema che utilizza il particle smoother non soffre invece di questo problema, infatti le particelle vengono generate solo in accordo con il termine a priori, senza considerare i centri estratti dal tracker 2D. D'altra parte, nel caso in cui manchi la misura del veicolo associata al filtro per un lungo periodo, le particelle potrebbero disperdersi, non effettuando il passo di ricampionamento, ed è il caso del veicolo perso riportato in Tabella 5.7.

Figura 5.8: Esempio di errore compiuto nella stima da Viterbi.

In Tabella 5.7 riassumiamo i risultati del tracking per la parte di filmato considerata nel paragrafo precedente. Evidenziamo i casi in cui viene perso un veicolo, ovvero il sistema non insegue il veicolo per tutto il percorso compiuto nella rotonda, e i casi in cui la stima della pose è errata, ovvero i casi in cui l'errore non è trascurabile. Per quest'ultimo caso, in Figura 5.8 riportiamo un esempio di stima errata nel tracking basato su Viterbi. L'errore è causato dal fatto che la basckground subtraction stima in modo errato il blob del veicolo, quindi il suo centro è una cattiva approssimazione del vero centroide del veicolo. Tra questi ultimi problemi di stima, non

consideriamo nel computo in tabella i casi in cui la stima della posizione è errata per problemi legati alla dimensione del modello (Figura 5.7), in questi casi sarebbe infatti sufficiente avere un'implementazione efficiente dell'algoritmo, eventualmente in linguaggio C/C++, che permetta di considerare un numero più elevato di campioni.

Tabella 5.7: Risultati del tracking su un totale di 53 veicoli.

	veicoli persi	**errori di stima**
Particle	1 (2%)	0 (0%)
Viterbi	0 (0%)	1 (2%)

Infine in Tabella 5.8 riportiamo tutti i valori dei parametri usati negli algoritmi. Tra i valori, merita un commento il valore di τ_{fermo} che ha due valori differenti per il due sistemi. Questo parametro viene confrontato con la velocità stimata per distinguere i casi in cui il moto è fermo da quelli in cui è in moto. Dato che nell'algoritmo basato su Viterbi la stima della velocità è calcolata dalla retroproiezioni delle stime 2D dei centroidi date dal tracker 2D, questo valore è più rumoroso rispetto al valore stimato nel particle smoother, il quale basa invece la stima sui i valoi dello spostamento campionati. Per questo motivo il valore del τ_{fermo} del sistema basato su Viterbi è più elevato.

Tabella 5.8: Tabella riassuntiva dei valori dei parametri

Modulo	Nome	Valore
Proiezione modello (4.2)	L^1	4 m
	r^1_{la}	$\frac{1.4}{4}$ m
	r^1_{lp}	$\frac{1.7}{4}$ m
	L^2	15 m
	r^2_{la}	$\frac{3}{15}$ m
	r^2_{lp}	$\frac{2}{15}$ m
	L^3	2 m
	r^3_{la}	$\frac{1}{2}$ m
	r^3_{lp}	$\frac{0.5}{2}$ m
	σ_1	0.5 m
	σ_2	2.5 m
	σ_3	0.25 m
	n	10
Verosimiglianza (4.3)	λ_1	0.4
	λ_2	0.6
Campionamento per singolo frame (4.4)	σ_x	0.5 m
	σ_y	0.5 m
	σ_θ	$\frac{\pi}{12}$ rad
Termine a priori - Viterbi (4.5.1)	σ_ρ	0.3 m
	$\sigma_{\theta_{xy}}$	$\frac{\pi}{24}$ m
	σ_{σ_θ}	$\frac{\pi}{48}$ m
	τ_{fermo}	1 m
	λ_{fermo}	$\frac{1}{0.5\tau_{fermo}}$ rad
Termine a priori - Particle (4.5.2)	σ_ρ	0.35 m
	σ_{σ_θ}	$\frac{\pi}{12}$ m
	τ_{fermo}	0.5 m
	ρ_{fermo}	0.6 rad

Capitolo 6

Conclusioni e sviluppi futuri

Abbiamo presentato in questa tesi due sistemi di tracking 3D model-based di veicoli che attraversano una scena ripresa da una o più telecamere. Per modellare il veicolo nel mondo, utilizziamo un parallelepipedo di dimensioni variabili che permette di effettuare una grezza classificazione in tre differenti categorie: automobili, camion, e moto. Entrambe i sistemi inferiscono le pose di un veicolo a partire da dati estratti da un tracker 2D implementato in un lavoro precedente; grazie a questo, a differenza della gran parte dei lavori presenti in letteratura che utilizzano tecniche di filtraggio, i nostri algoritmi sfruttano la conoscenza dell'intera traiettoria per eseguire il tracking attraverso due stimatori smoothed.

Il primo sistema esegue il tracking con un metodo strutturato in modo analogo a quello in [58], ma alla luce dei dati 2D da cui partiamo abbiamo semplificato l'algoritmo e abbiamo reso possibile il riconoscimento di veicoli di differenti misure che non è previsto originariamente. Il secondo sistema utilizza un particle smoother che, tenendo conto dell'intera traiettoria, permette una stima più precisa e smussata rispetto ai classici algoritmi di filtraggio basati su particelle, diminuendo soprattutto dispersione dei campioni attorno alle stime della traiettoria corrispondenti ai bordi dell'immagine.

Il tracking 3D permette di ottenere una stima più precisa dell'interazione di un veicolo con la rotonda rispetto al tracking 2D. Per esempio, quando un veicolo sta entrando o uscendo dall'immagine il tracker 2D si limita a stimare il centro della parte visibile del veicolo. Abbiamo visto come questo errore può risultare molto elevato nel tracking di camion o veicoli di grandi dimensioni. I sistemi di tracking proposti non limitano a priori la ricerca della pose solo a quei punti che che si proiettano nell'immagine e quindi offrono uno strumento più preciso e flessibile per la stima. Questo in particolar modo è vero per il caso multicamera in cui la presenza di due o più telecamere che vedono lo stesso veicolo consente una stima robusta dell'intera traiettoria.

Gli esperimenti condotti sui dati simulati e sui dati reali evidenziano

che l'algoritmo di particle smoothing offre stime migliori rispetto al sistema basato sull'algoritmo di Viterbi. Per le ragioni esposte nel seguito, crediamo inoltre che questo algoritmo sia più flessibile in prospettiva futura e possa essere migliorato con maggior semplicità.

L'utilizzo di modelli 3D per rappresentare i veicoli nel mondo rende anche possibile progettare un sistema per riconoscere le ombre, ad esempio generando l'ombra dei modelli sul piano della rotonda, come in [58] e tenendo conto anche di questa regione nel calcolo della verosimiglianza.

Abbiamo visto come entrambe i sistemi eseguono il tracking partendo dai dati estratti da un algoritmo di tracking 2D. Un possibile sviluppo potrebbe portare a rendere un sistema, o entrambe indipendenti dal tracker 2D. Questo renderebbe possibile ragionare sulla posizione dei veicoli gestendo in modo migliore le occlusioni tra veicoli rispetto a quanto già fatto nel tracker 2D. Conoscendo la posizione, l'orientamento e la dimensione dei veicoli nella scena, sarebbe infatti possibile ragionare sulle configurazioni assunte dai veicoli evitando quelle impossibili o improbabili, ad esempio quando due veicoli si compenetrano. Si potrebbe anche ragionare sul risultato della background subtraction, "spiegando" le situzioni in cui due veicoli sono così vicini e le misure si fondono: in questi casi il tracker 2D individuerebbe un solo blob unico il cui centro è distante dai centri dei due veicoli, e quindi assocerebbe la misura errata ad uno solo dei filtri che seguivano i veicoli, e all'altro filtro non viene associata alcuna misura. Con un sistema 3D indipendente dalle misure del tracker 2D, il calcolo della verosimiglianza verrebbe fatto solo confrontando la proiezione del modello con l'area di foreground intersecata.

L'algoritmo basato sul particle smoothing ci sembra che abbia le potenzialità per adattarsi più semplicemente a questo cambiamento: il sistema utilizza per ora solo il primo centro come inizializzazione del sistema, e la data association implementata in 2D per calcolare la verosimiglianza. Implementando un algoritmo di data association, e un metodo di inizializzazione del sistema si raggiungerebbe l'obiettivo di indipendenza dal tracker 2D. Con il sistema Viterbi-based, invece, il compito è più complesso: durante tutta la prima fase usiamo le stime della traiettoria 2D per trovare i campioni. Un metodo che rende il sistema autonomo, è quello presentato da Song e Nevatia, che farebbe venir meno le semplificazioni fatte nel Paragrafo 3.3, si dovrebbe quindi studiare un approccio ibrido tra quest'ultimo e quello proposto in questo lavoro.

Durante lo sviluppo del tracker 2D si è notato come l'utilizzo delle coordinate polari con centro nel centro della rotonda abbia permesso una stima migliore delle covarianze che modellano gli errori per il filtro di Kalman nella fase di data association. Si potrebbe verificare se anche nel nostro caso, una volta reso il tracking indipendente dalla data association in 2D, rappresentare la posizione del veicolo in coordinate polari possa essere una scelta migliore rispetto al sistema di coordinate utilizzato attualmente.

Infine, avendo implementato i sistemi presentati in Matlab, un migliora-

mento significativo in termini di prestazioni si potrebbe avere implementando i sistemi in linguaggio C++ attraverso l'utilizzo delle librerie OpenCV e di opportuni meccanismi per sfruttare le architetture multi-core. Matlab infatti offre un ambiente adatto allo sviluppo di prototipi di algoritmi, ma la sua flessibilità si paga in termini di onere computazionale. Abbiamo notato come il solo utilizzo di una funzione C abbia diminuito il tempo di esecuzione di una funzione di un ordine di grandezza (Paragrafo 4.3.2). Con un algoritmo più efficiente si potrebbe pensare anche di aumentare il numero di campioni classe-scala per cui viene eseguito il campionamento. Abbiamo visto come i due sistemi eseguano una classificazione alquanto grezza dei veicoli. Utilizzando una batteria più ampia di dimensioni per cui effettuiamo tracking, potremmo anche conoscere con maggiore precisione le dimensioni di un veicolo.

In sintesi, i sistemi presentati offrono due approcci differenti al tracking 3D. A nostro parere il metodo che utilizza il particle smoothing fornisce risultati migliori ed è maggiormente flessibile per cambiamenti futuri. Confidiamo nel fatto che quest'ultimo possa essere reso indipendente dal tracking per integrare in un unico sistema il nostro sistema con un metodo efficacie di data association.

Appendice A

Progetto logico del software

Mostriamo in questo capitolo i due diagrammi che rappresentano schematicamente l'architettura di funzionamento dei sistemi sviluppati. Ogni riquadro è diviso in due parti, in alto, il titolo in grassetto è il nome della funzione il cui compito viene spiegato nella parte sottostante. Il diagramma in Figura A.1 rappresenta il sistema basato sull'algoritmo di Viterbi, il diagramma in Figura A.2 rappresenta il sistema che esegue il tracking tramite il particle smoother..

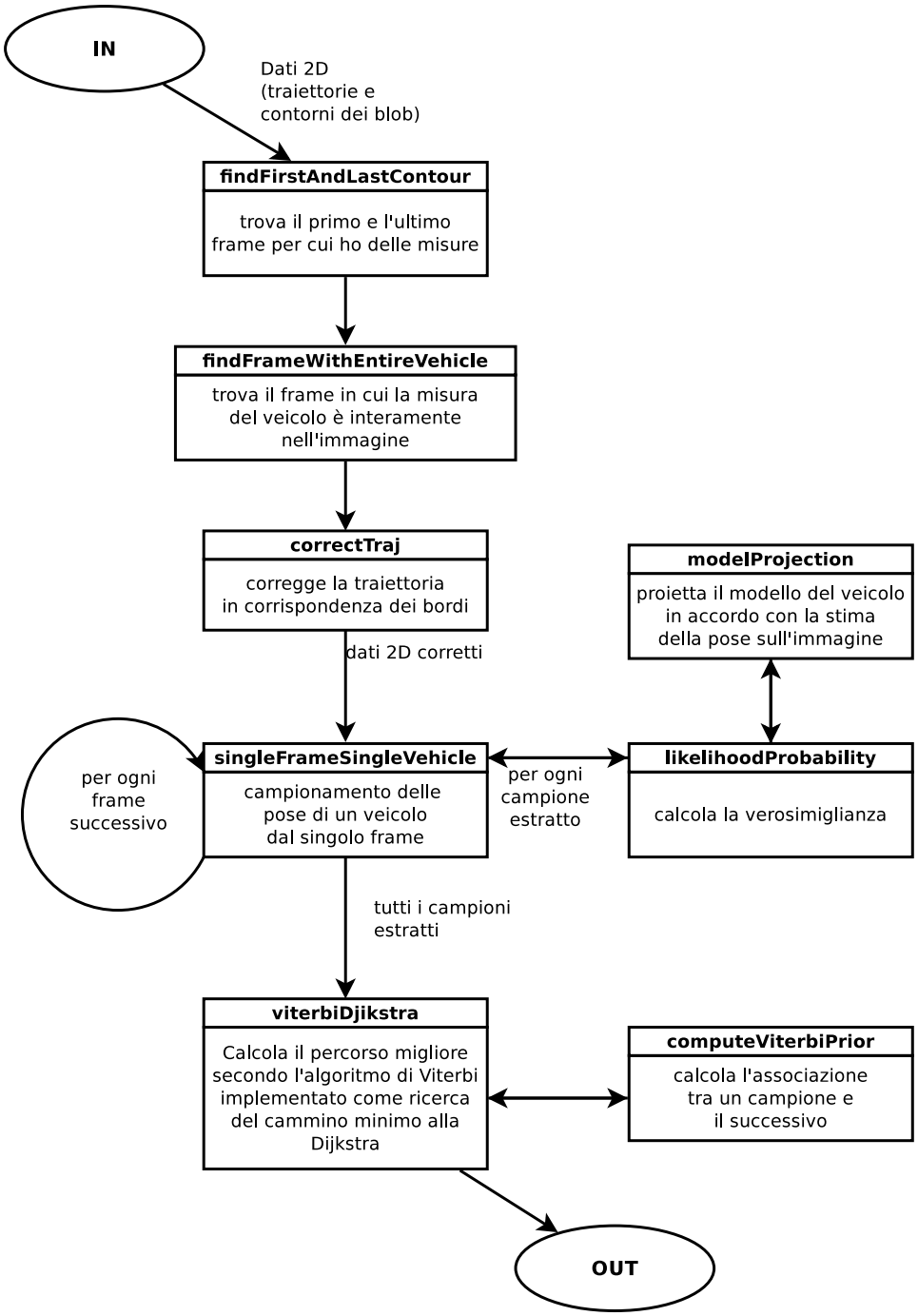


Figura A.1: Diagramma del sistema di tracking con Viterbi.

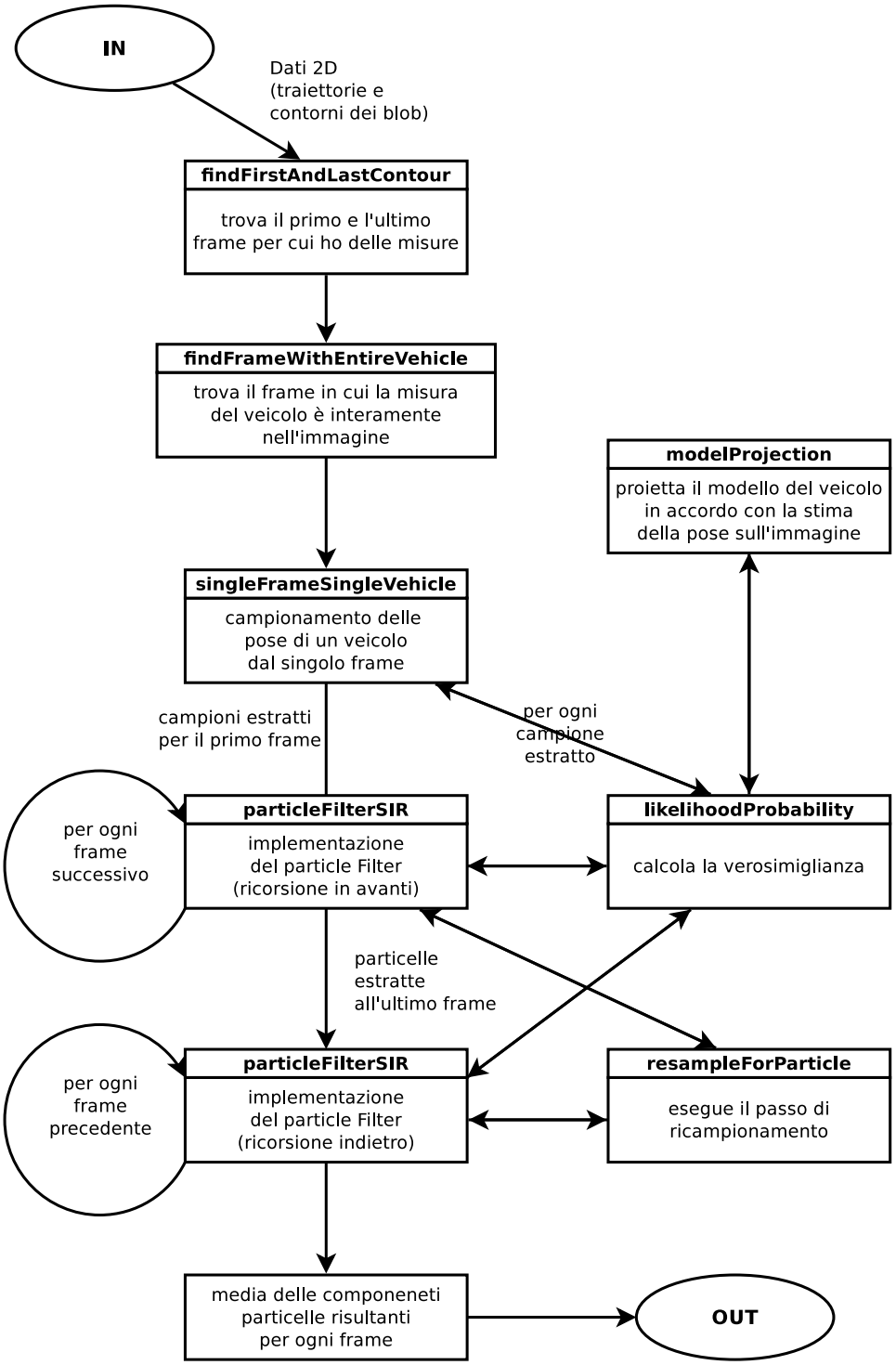


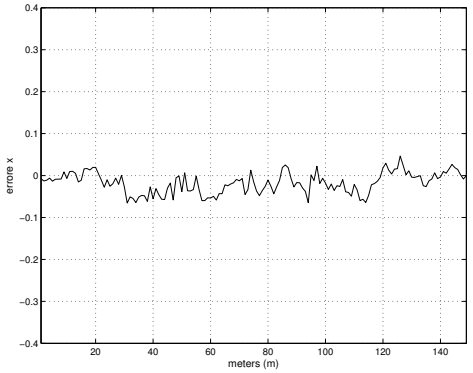
Figura A.2: Diagramma del sistema di tracking con il Particle smoother.

Appendice B

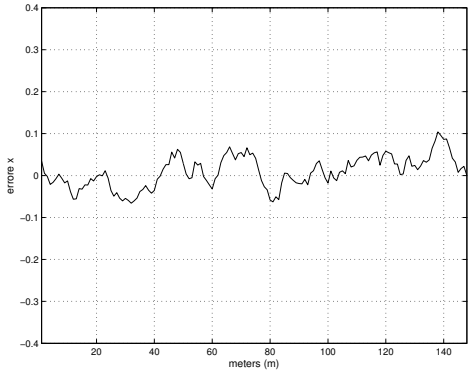
Risultati sui filmati simulati

In questa appendice mostriamo i risultati del confronto tra le pose stimate dai due sistemi proposti e le pose che abbiamo utilizzato per generare i filmati simulati su cui abbiamo fatto tracking. Abbiamo condotto due esperimenti. Nel primo abbiamo scelto due serie di pose: in una il moto rappresentato è circolare uniforme, nell'altra è un moto rettilineo in cui un veicolo in movimento si arresta, staziona nella posizione raggiunta poi riparte. Il primo caso modella il comportamento di un veicolo che percorre l'intersezione rotatoria, il secondo invece modella l'immissione del veicolo nella rotatoria. Nel secondo esperimento simuliamo una traiettoria analoga a quelle del caso reale, con la stessa prospettiva adottata da una delle telecamere reali e le stesse dimensioni della rotatoria reale.

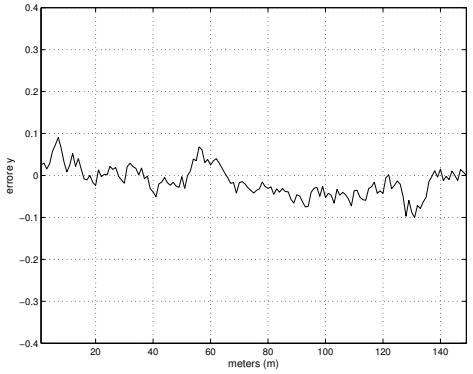
In ogni pagina presentiamo sei grafici, la prima colonna corrisponde agli errori sulle tre componenti della pose stimate dal particle smoother, la seconda con il sistema basato sull'algoritmo di Viterbi. Per ogni grafico, in ascissas è indicato il numero di frame e in ordinata il valore dell'errore. Per ogni grafico riportiamo inoltre l'indicazione della media dei valori assoluti.



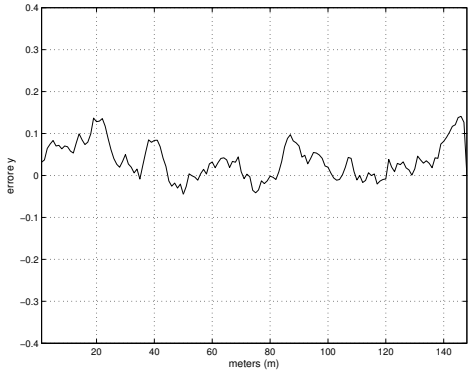
(a) *Tracking con Particle Smoother: errore x.*



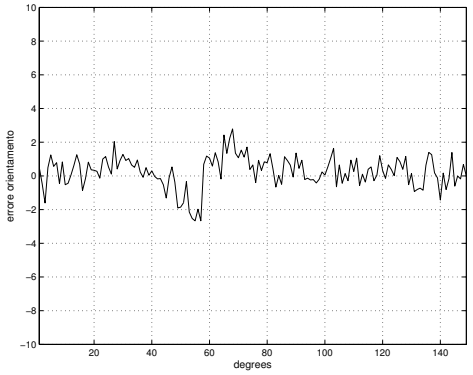
(b) *Tracking con Viterbi: errore x.*



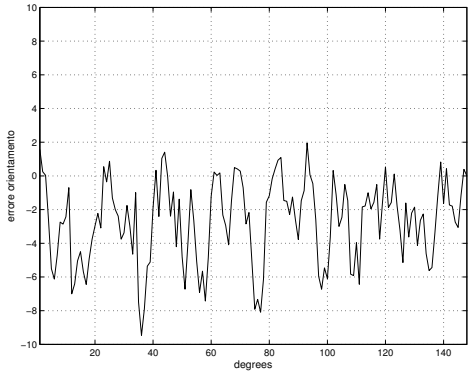
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*

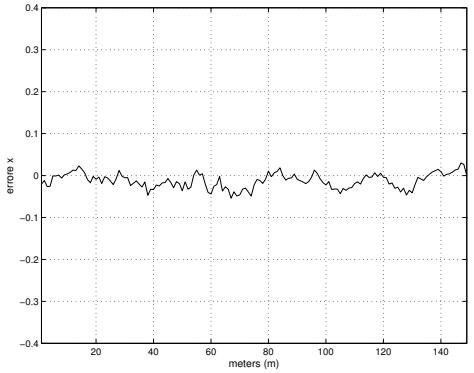


(e) *Tracking con Particle Smoother: errore θ.*

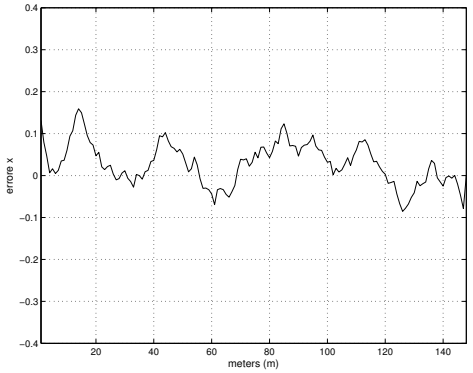


(f) *Tracking con Viterbi: errore θ.*

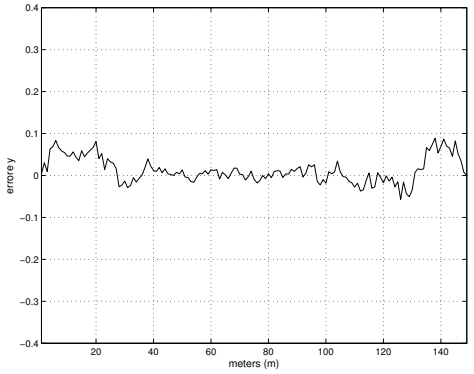
Figura B.1: Errori del tracking del veicolo in moto circolare uniforme dalla prima prospettiva.



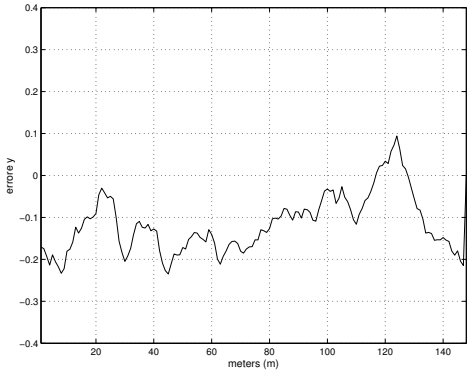
(a) *Tracking con Particle Smoother: errore x.*



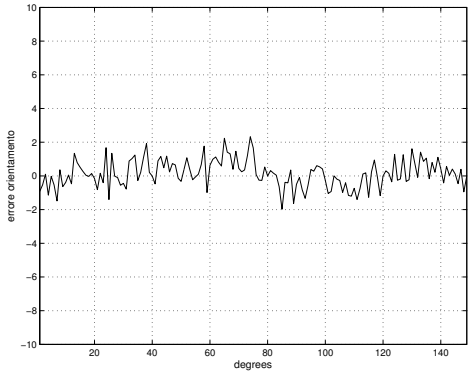
(b) *Tracking con Viterbi: errore x.*



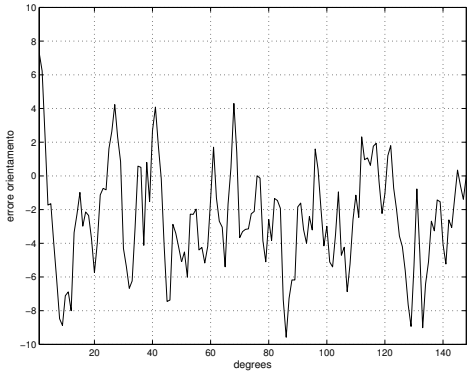
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*

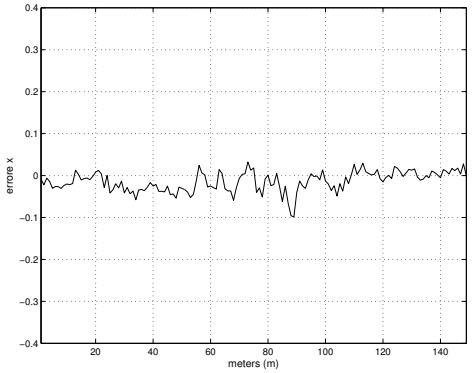


(e) *Tracking con Particle Smoother: errore θ.*

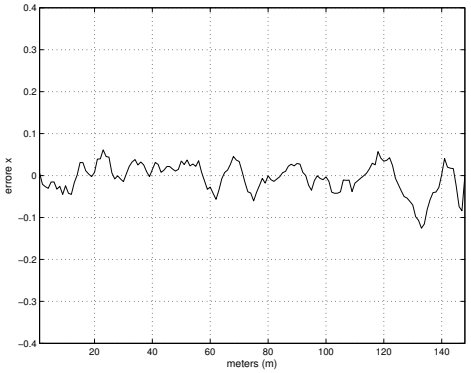


(f) *Tracking con Viterbi: errore θ.*

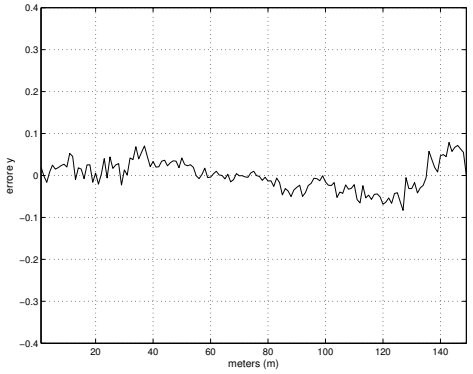
Figura B.2: Errori del tracking del veicolo in moto circolare uniforme dalla seconda prospettiva.



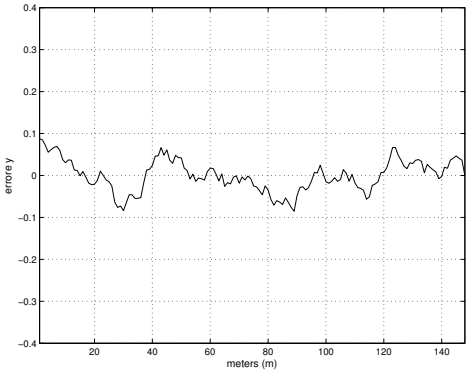
(a) *Tracking con Particle Smoother: errore x.*



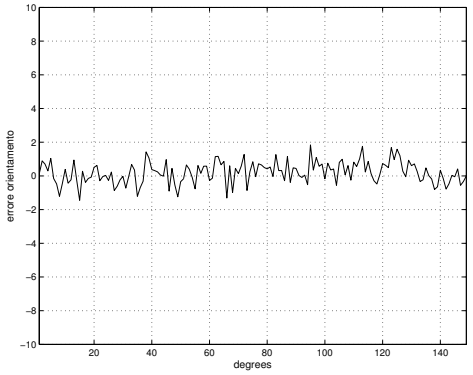
(b) *Tracking con Viterbi: errore x.*



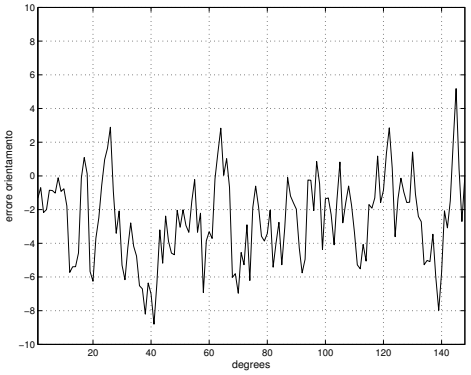
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*

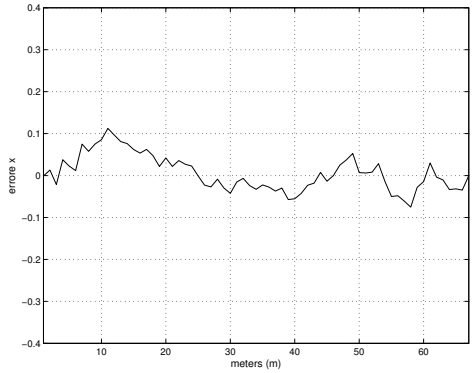


(e) *Tracking con Particle Smoother: errore θ.*

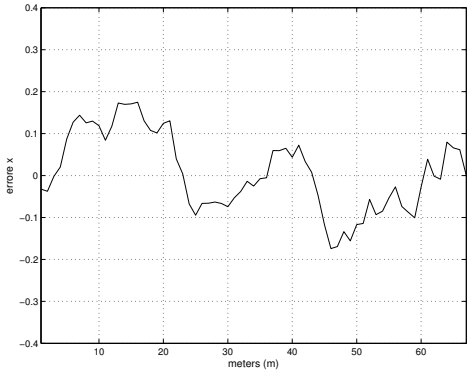


(f) *Tracking con Viterbi: errore θ.*

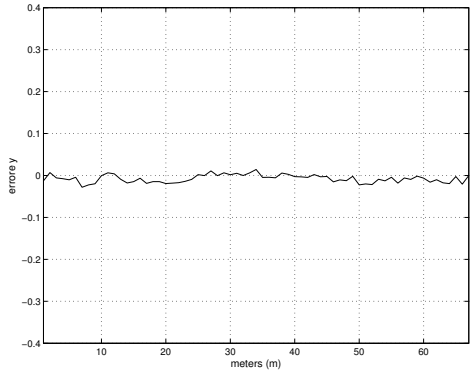
Figura B.3: Errori del tracking del veicolo in moto circolare uniforme dalla terza prospettiva.



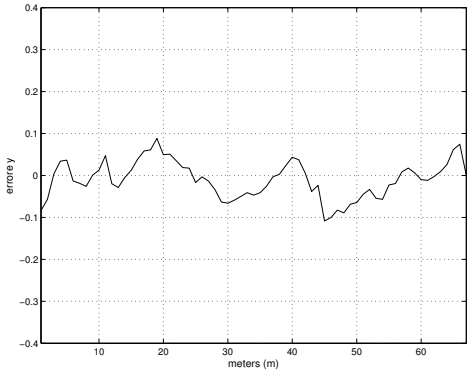
(a) *Tracking con Particle Smoother: errore* x.



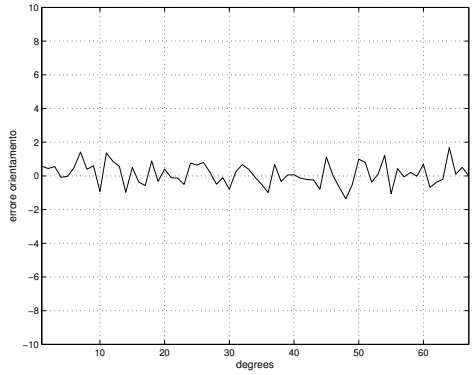
(b) *Tracking con Viterbi: errore* x.



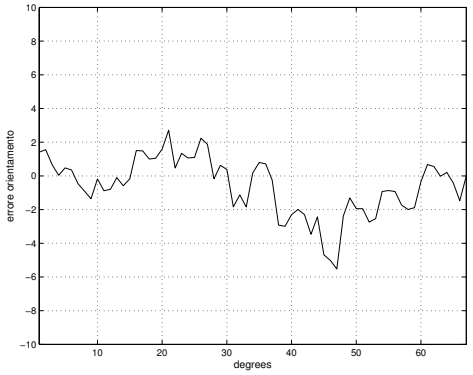
(c) *Tracking con Particle Smoother: errore* y.



(d) *Tracking con Viterbi: errore* y.

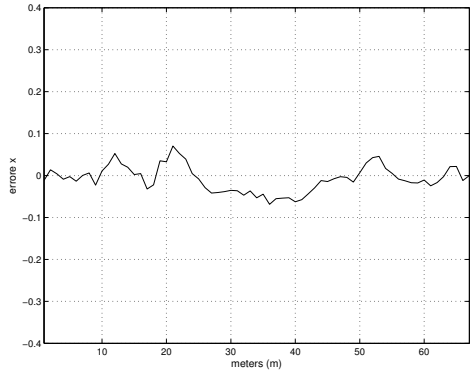


(e) *Tracking con Particle Smoother: errore* θ.

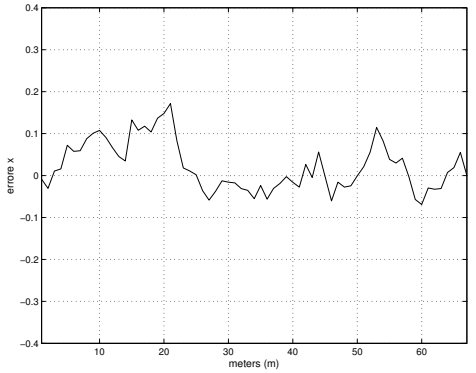


(f) *Tracking con Viterbi: errore* θ.

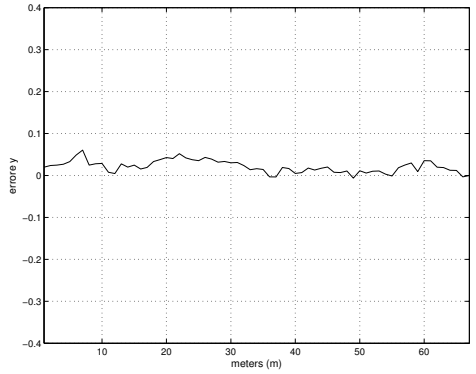
Figura B.4: Errori del tracking del veicolo in moto rettilineo dalla prima prospettiva.



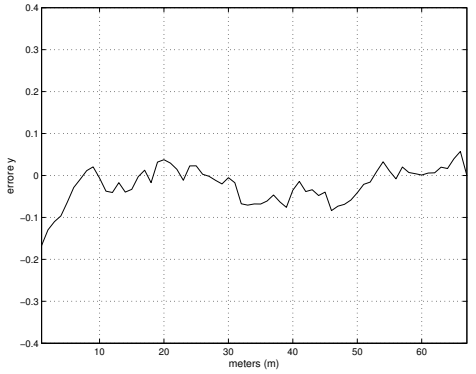
(a) *Tracking con Particle Smoother: errore x.*



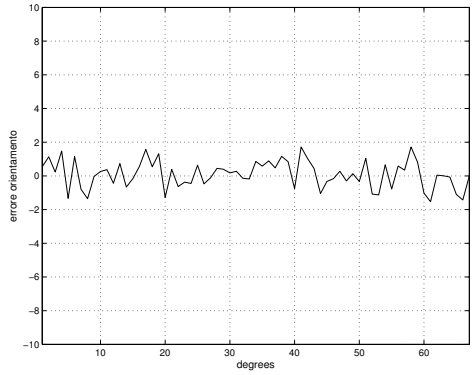
(b) *Tracking con Viterbi: errore x.*



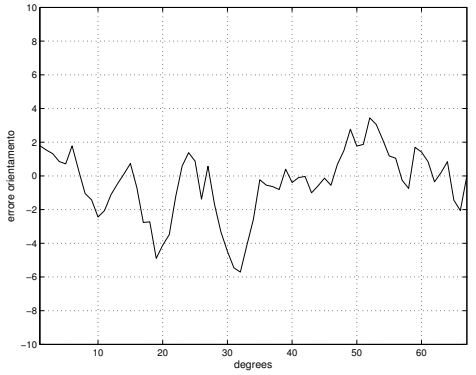
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*

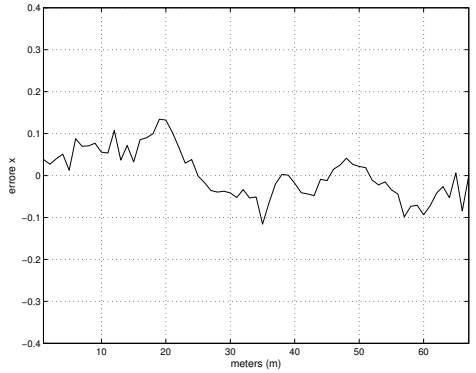


(e) *Tracking con Particle Smoother: errore θ.*

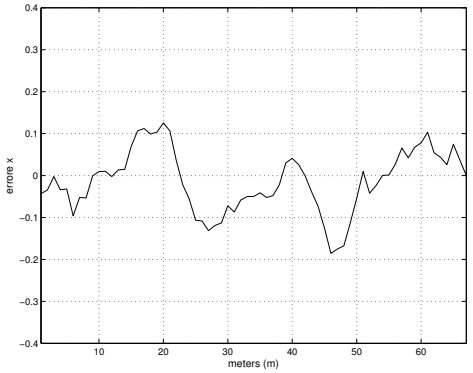


(f) *Tracking con Viterbi: errore θ.*

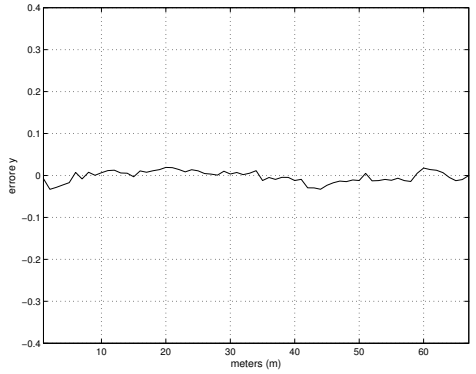
Figura B.5: Errori del tracking del veicolo in moto rettilineo dalla seconda prospettiva.



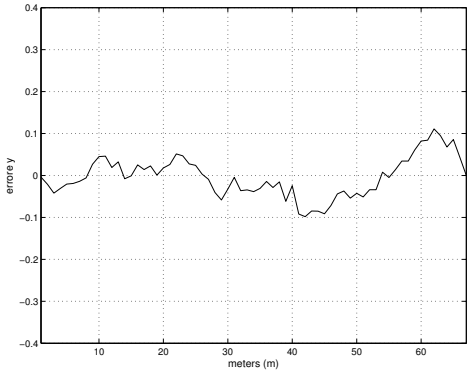
(a) *Tracking con Particle Smoother: errore* x.



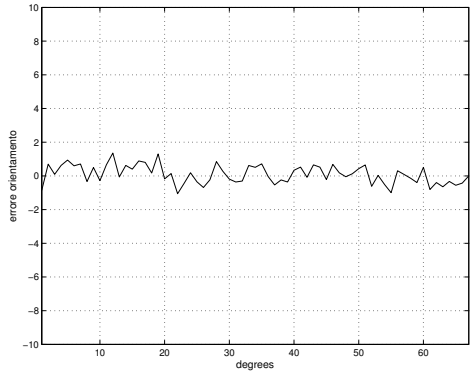
(b) *Tracking con Viterbi: errore* x.



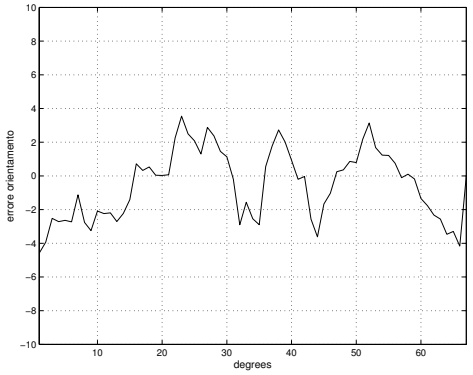
(c) *Tracking con Particle Smoother: errore* y.



(d) *Tracking con Viterbi: errore* y.

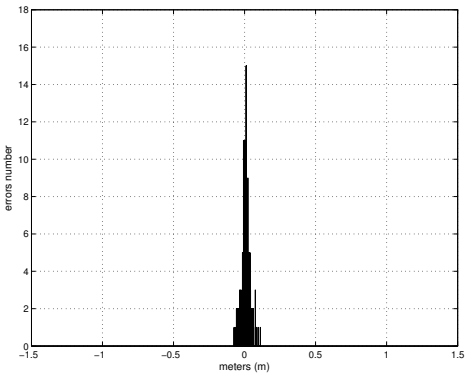


(e) *Tracking con Particle Smoother: errore* θ.

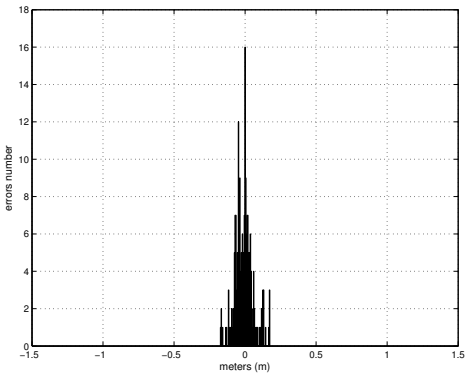


(f) *Tracking con Viterbi: errore* θ.

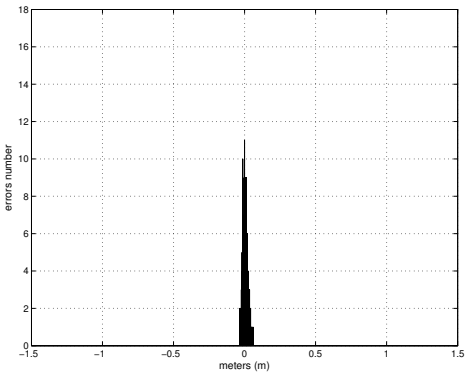
Figura B.6: Errori del tracking del veicolo in moto rettilineo dalla terza prospettiva.



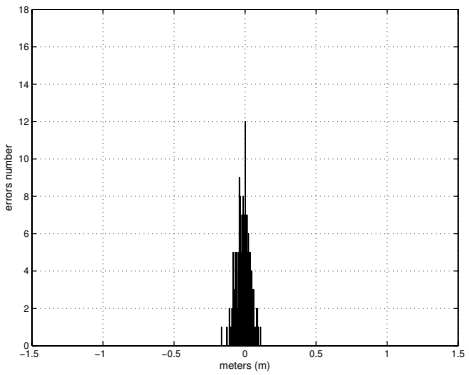
(a) *Tracking con Particle Smoother: errore x.*



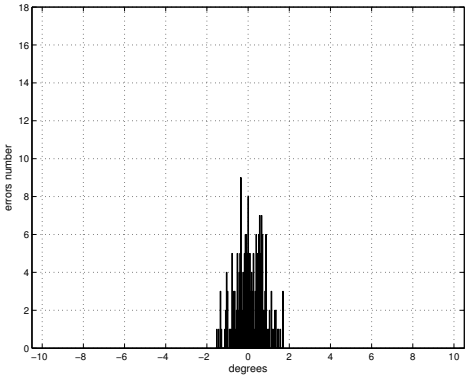
(b) *Tracking con Viterbi: errore x.*



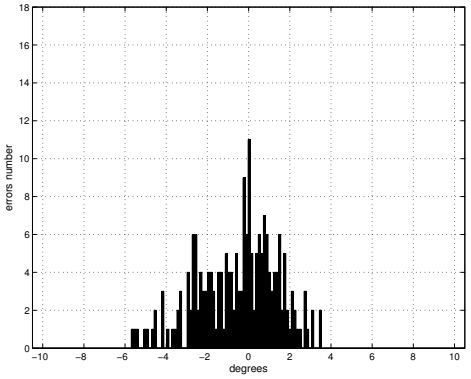
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*

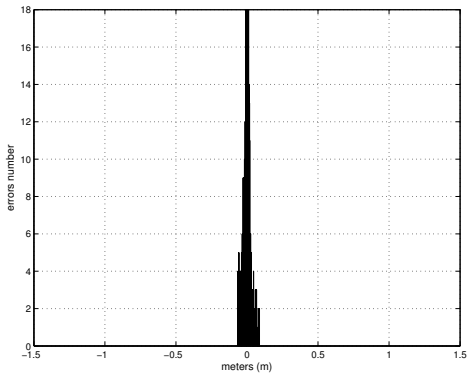


(e) *Tracking con Particle Smoother: errore θ.*

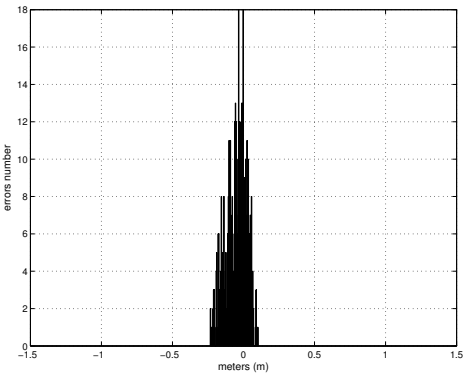


(f) *Tracking con Viterbi: errore θ.*

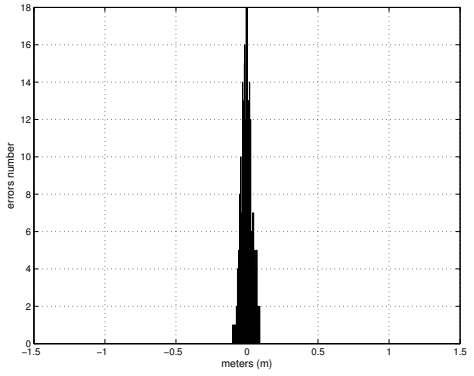
Figura B.7: Istogrammi degli errori per il tracking sulla traiettoria retta.



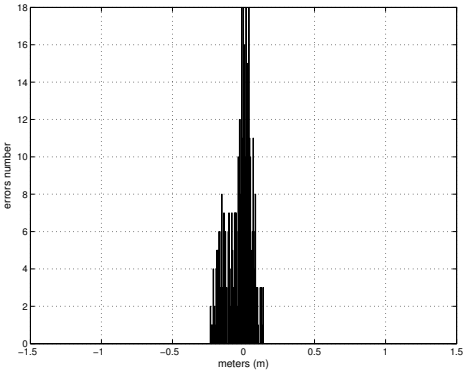
(a) *Tracking con Particle Smoother: errore x.*



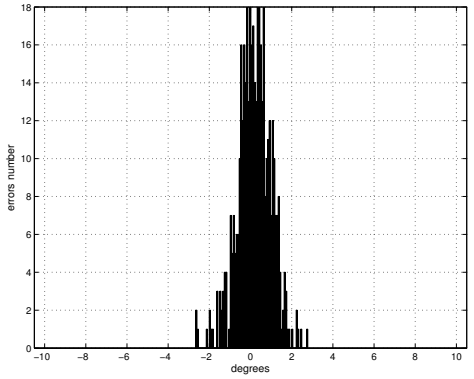
(b) *Tracking con Viterbi: errore x.*



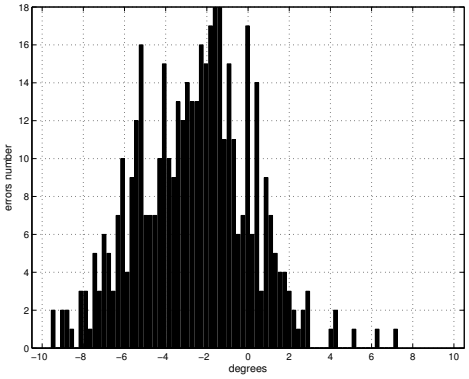
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*

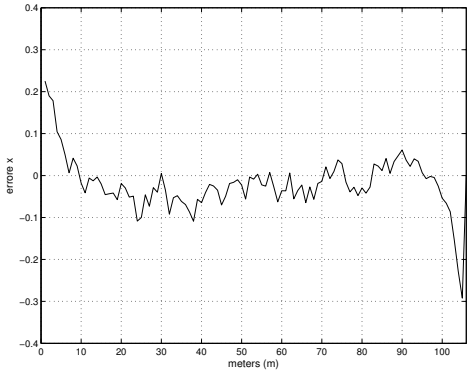


(e) *Tracking con Particle Smoother: errore θ.*

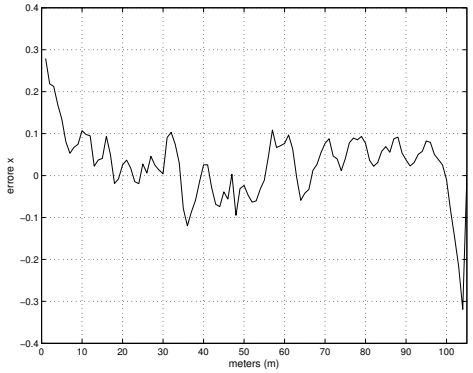


(f) *Tracking con Viterbi: errore θ.*

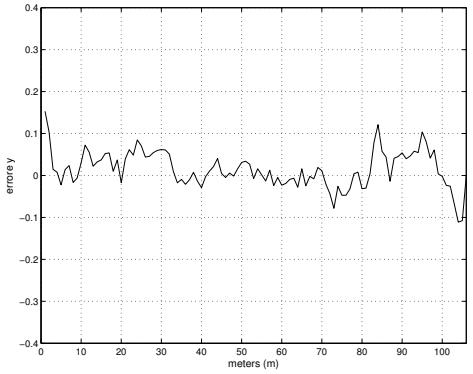
Figura B.8: Istogrammi degli errori per il tracking per il moto circolare uniforme.



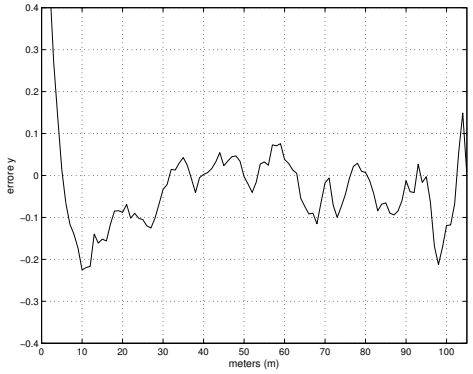
(a) *Tracking con Particle Smoother: errore x.*



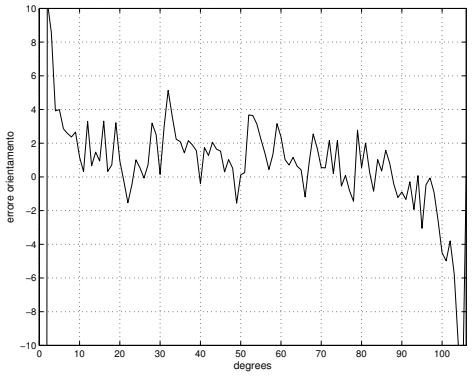
(b) *Tracking con Viterbi: errore x.*



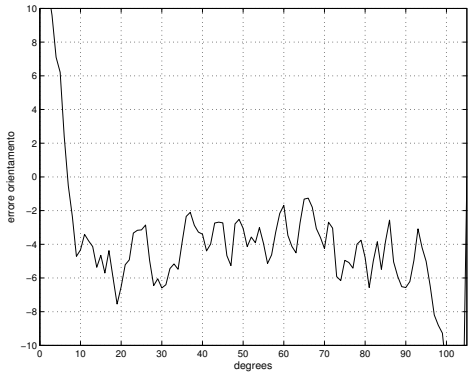
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*

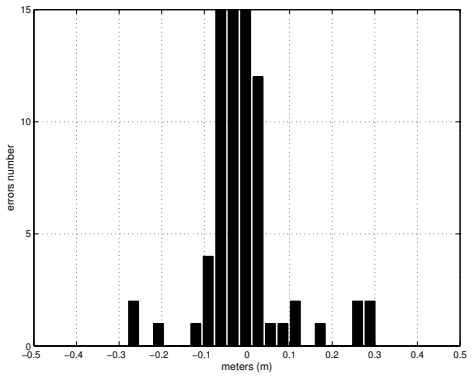


(e) *Tracking con Particle Smoother: errore θ.*

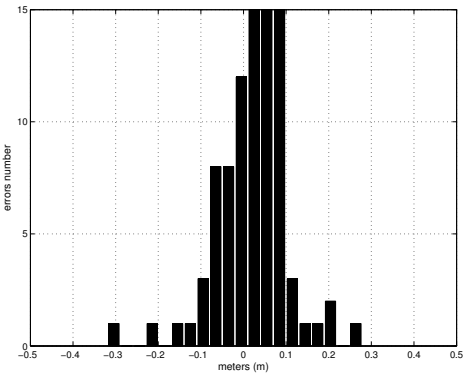


(f) *Tracking con Viterbi: errore θ.*

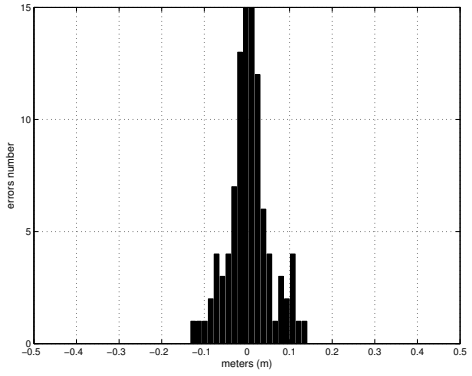
Figura B.9: Errori del tracking per l'esperimento che simula una traiettoria analoga a quelle del caso reale.



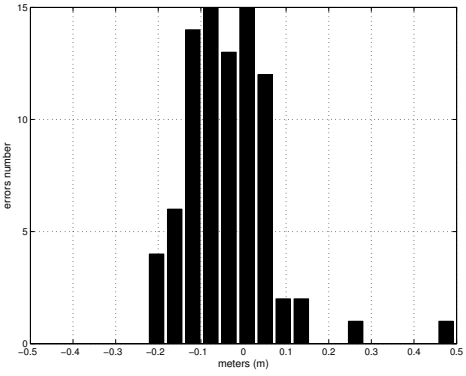
(a) *Tracking con Particle Smoother: errore x.*



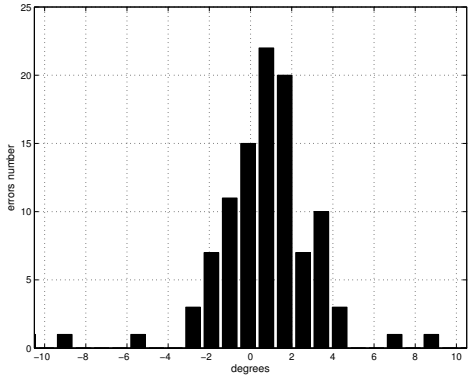
(b) *Tracking con Viterbi: errore x.*



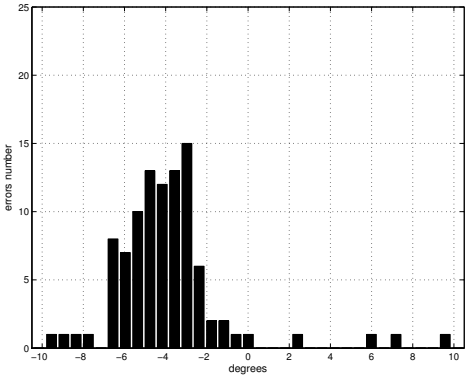
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*



(e) *Tracking con Particle Smoother: errore θ.*



(f) *Tracking con Viterbi: errore θ.*

Figura B.10: Istogrammi degli errori per l'esperimento che simula una traiettoria analoga a quelle del caso reale.

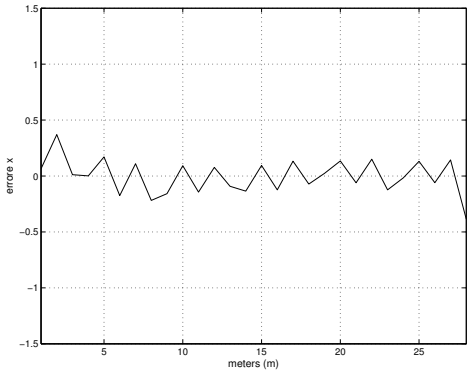
Appendice C

Risultati sui filmati reali

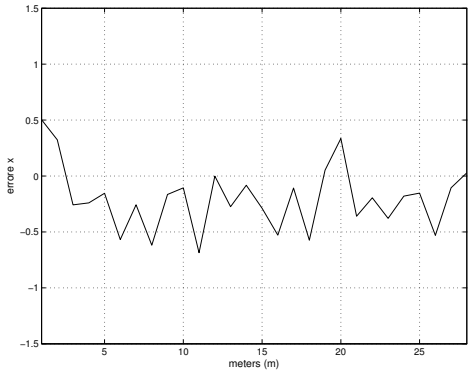
In questa appendice riportiamo i grafici che descrivono gli errori compiuti dai due algoritmi nel caso reale. Il confronto è eseguito tra la pose del caso reale, stimata con un misuratore di posizione GPS e un misuratore dell'orientamento IMU, e la pose stimata con i due sistemi proposti.

Come nel caso reale i grafici rappresentano per ogni colonna gli errori sulle tre componenti della pose per ognuno dei due algoritmi. Riportiamo gli errori sia per il caso a singola telecamera che per il caso a telecamera multipla per ognuna delle traiettorie che il veicolo compie. Nel caso multicamera le traiettorie sono in numero minore perchè non sempre il veicolo è passato per entrambe le telecamere considerate nel tracking. Per ogni grafico in ordinata indichiamo il numero di frame e in ascissa il valore dell'errore. A causa della pbassa frequesnza di funzionamento del GPS gli errori sulle coordinate x e y sono riferiti ad un numero minore di frame rispetto l'orientamento. La telecamera campiona circa quattro immagini Per ogni posizione stimata con il GPS, quindi confrontiamo quest'ultima con la posizione stimata corrispondente al frame più vicino temporalmente.

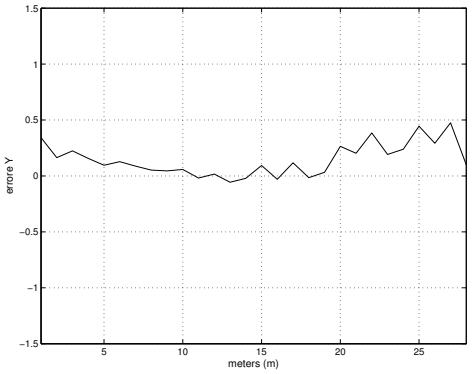
Infine, gli ultimi grafici rappresentano degli istogrammi riassuntivi sugli errori compiuti in totale dai due algoritmi per tutte le traiettorie nel caso a singola telecamera e nel caso a telecamera multipla. Per ogni coordinata abbiamo deciso di tracciare l'istogramma con 75 bin.



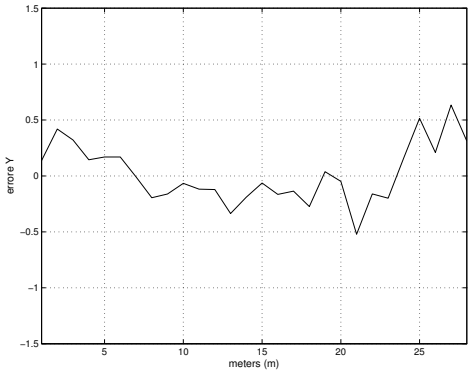
(a) *Tracking con Particle Smoother: errore x.*



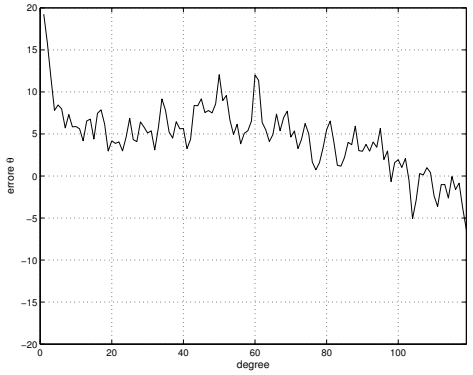
(b) *Tracking con Viterbi: errore x.*



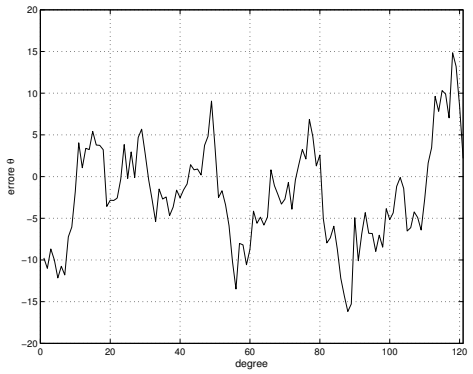
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*

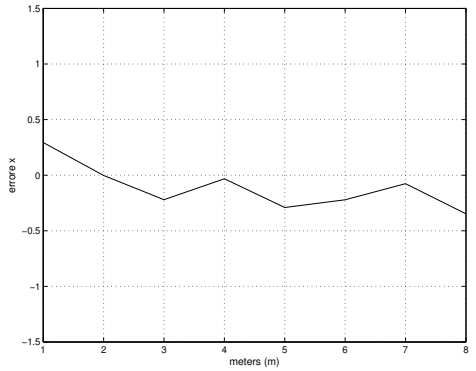


(e) *Tracking con Particle Smoother: errore θ.*

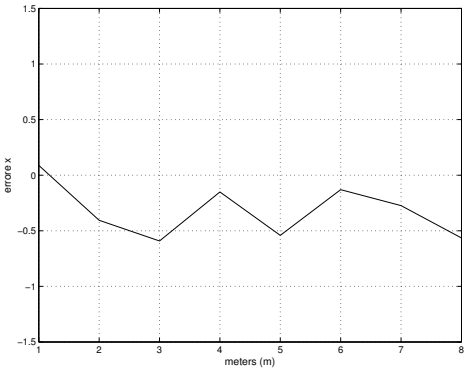


(f) *Tracking con Viterbi: errore θ.*

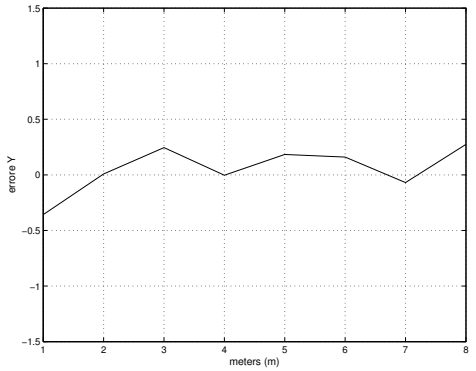
Figura C.1: Errori per la prima traiettoria a singola telecamera.



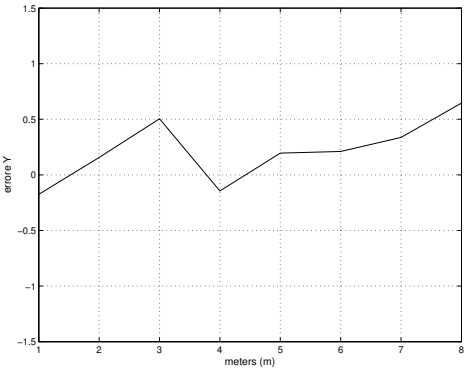
(a) *Tracking con Particle Smoother: errore x.*



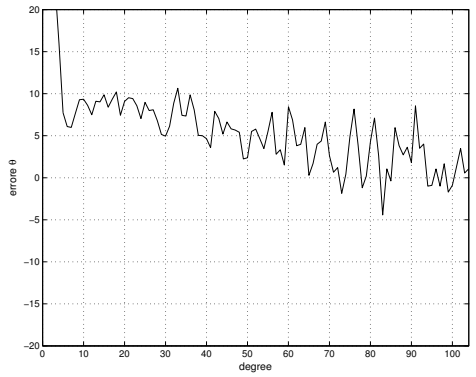
(b) *Tracking con Viterbi: errore x.*



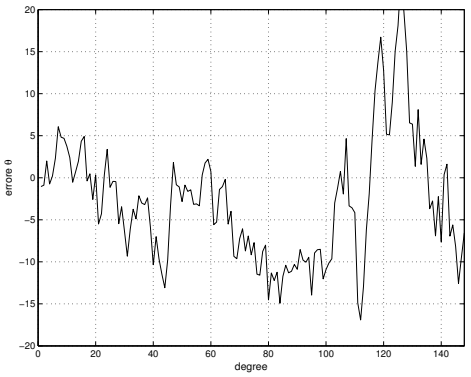
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*

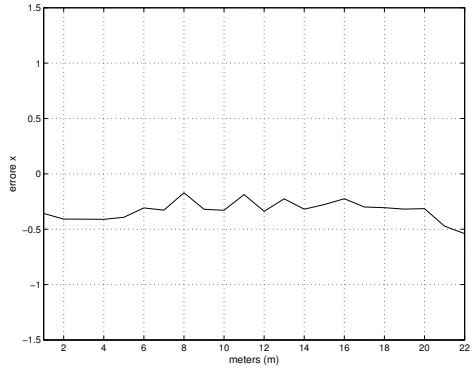


(e) *Tracking con Particle Smoother: errore θ.*

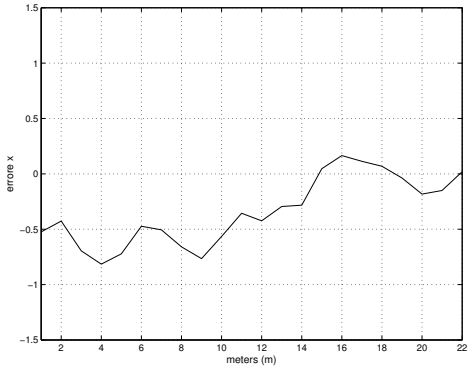


(f) *Tracking con Viterbi: errore θ.*

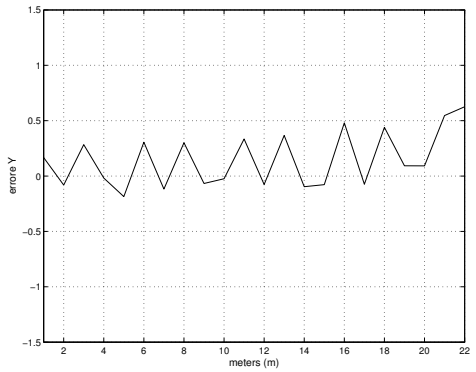
Figura C.2: Errori per la seconda traiettoria a singola telecamera.



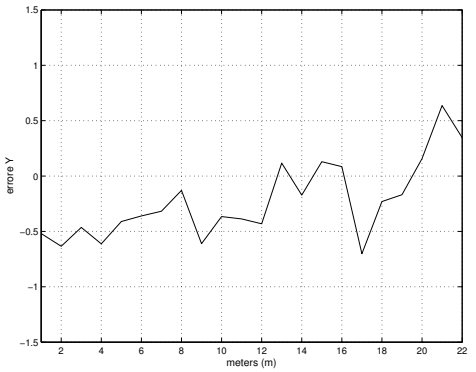
(a) *Tracking con Particle Smoother: errore x.*



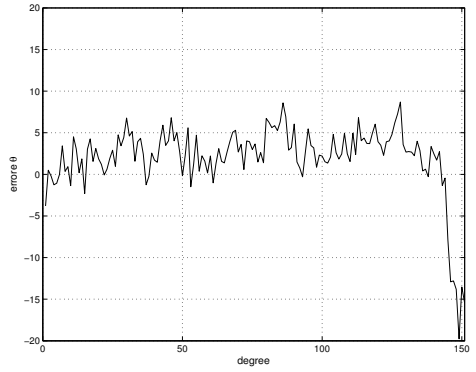
(b) *Tracking con Viterbi: errore x.*



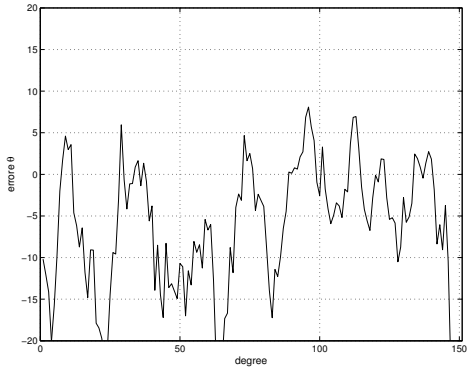
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*

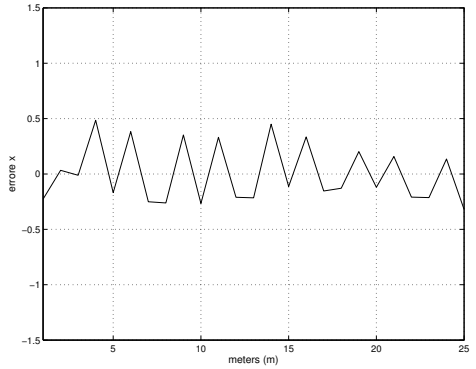


(e) *Tracking con Particle Smoother: errore θ.*

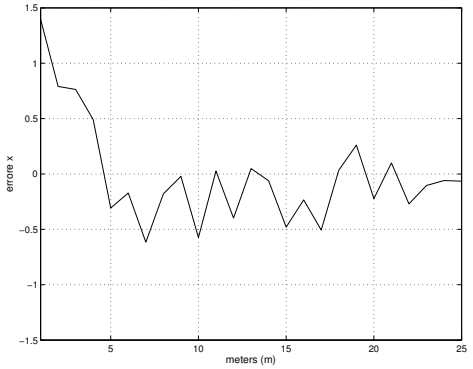


(f) *Tracking con Viterbi: errore θ.*

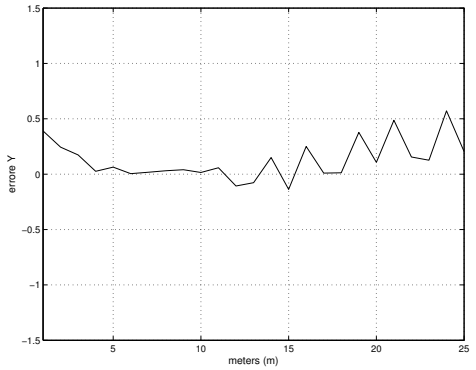
Figura C.3: Errori per la terza traiettoria a singola telecamera.



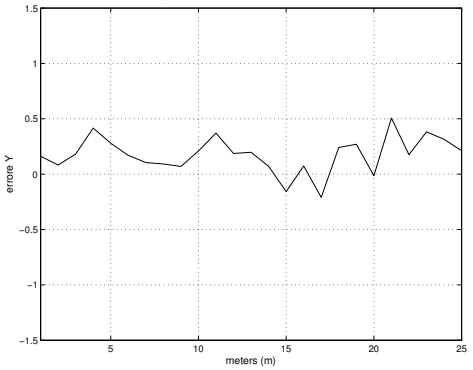
(a) *Tracking con Particle Smoother: errore x.*



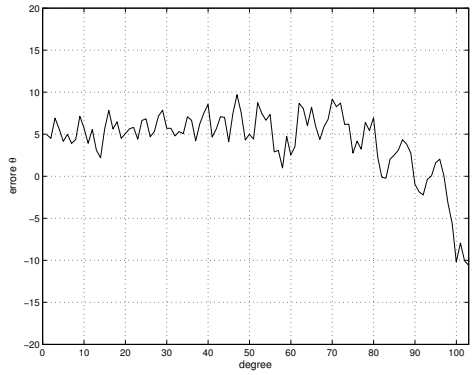
(b) *Tracking con Viterbi: errore x.*



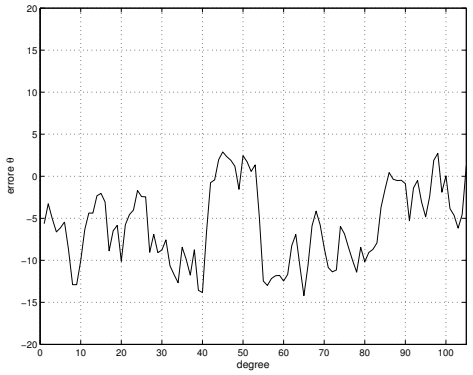
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*

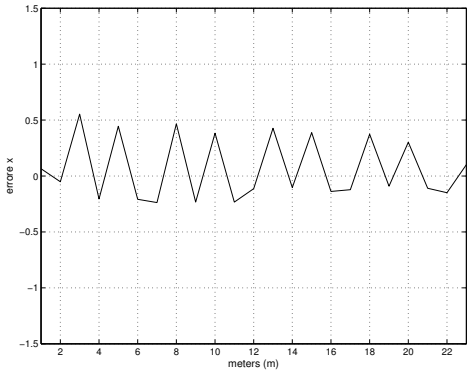


(e) *Tracking con Particle Smoother: errore θ.*

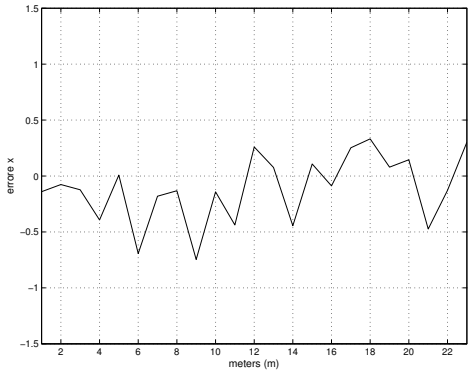


(f) *Tracking con Viterbi: errore θ.*

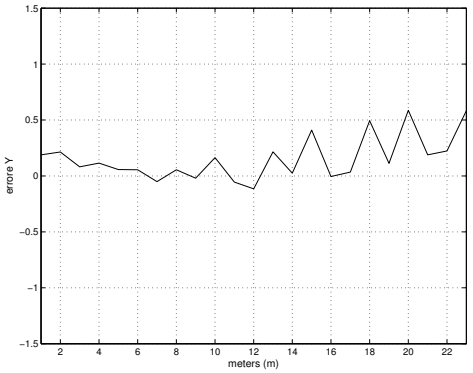
Figura C.4: Errori per la quarta traiettoria a singola telecamera.



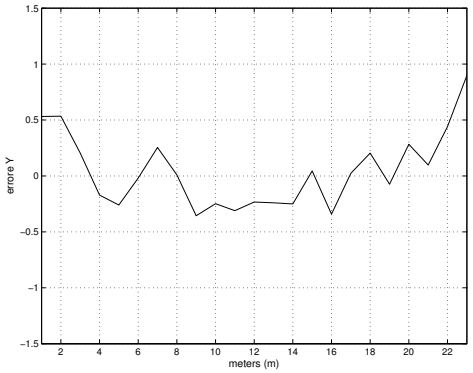
(a) *Tracking con Particle Smoother: errore x.*



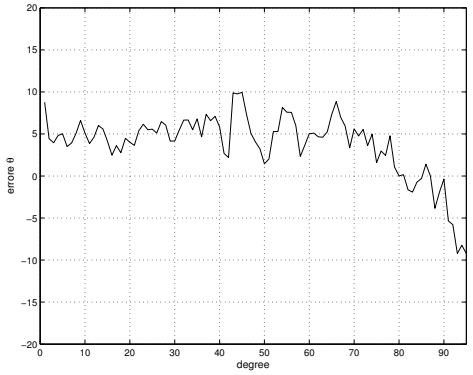
(b) *Tracking con Viterbi: errore x.*



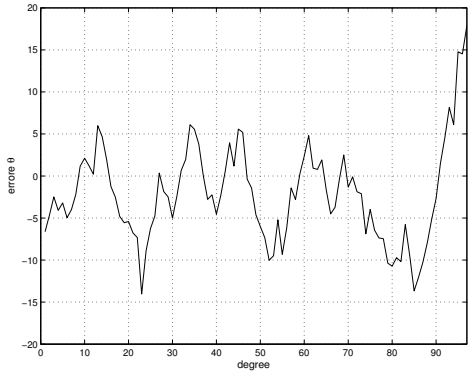
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*

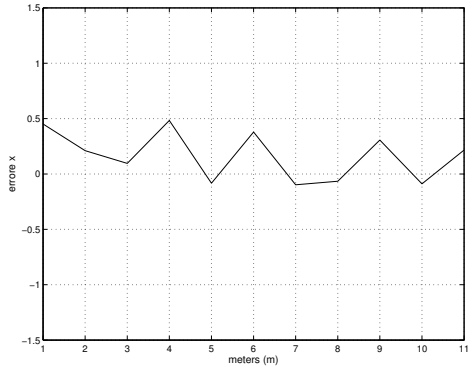


(e) *Tracking con Particle Smoother: errore θ.*

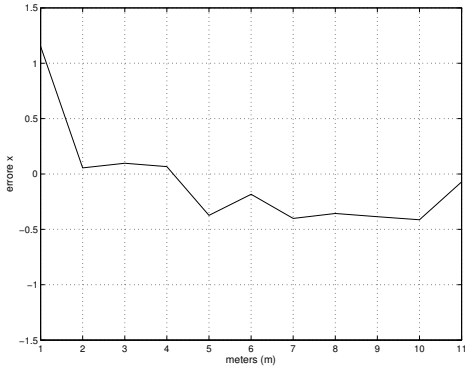


(f) *Tracking con Viterbi: errore θ.*

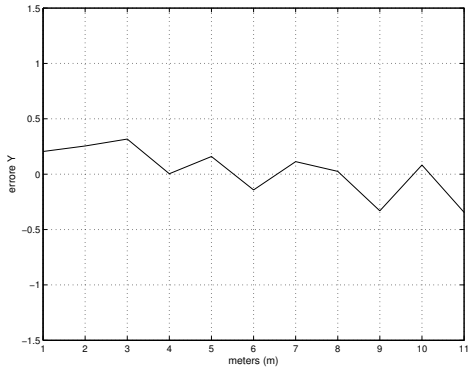
Figura C.5: Errori per la quarta traiettoria a singola telecamera.



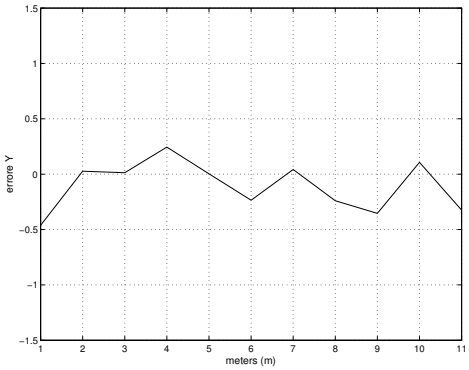
(a) *Tracking con Particle Smoother: errore x.*



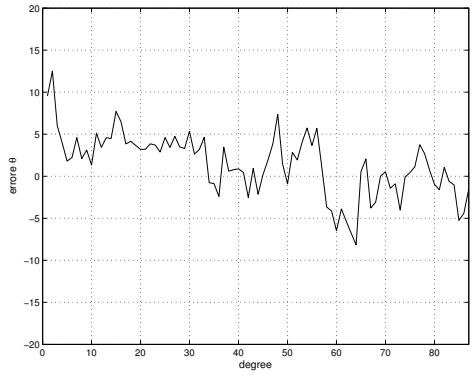
(b) *Tracking con Viterbi: errore x.*



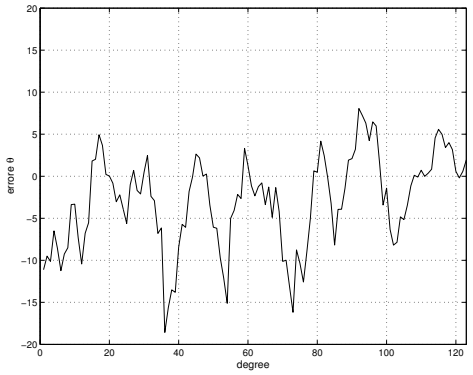
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*

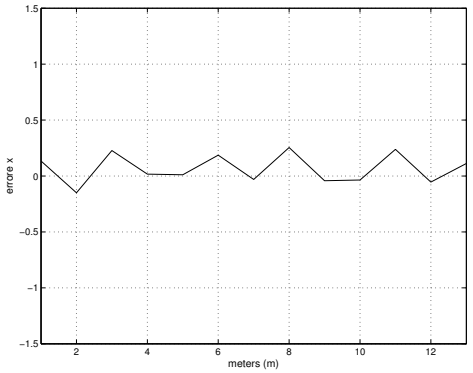


(e) *Tracking con Particle Smoother: errore θ.*

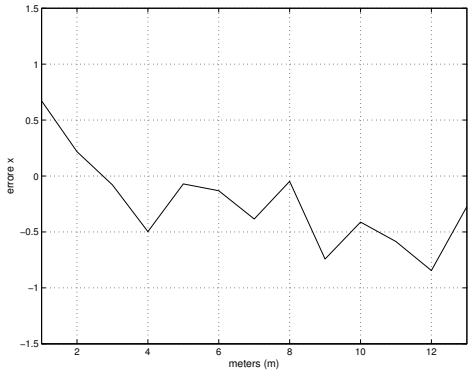


(f) *Tracking con Viterbi: errore θ.*

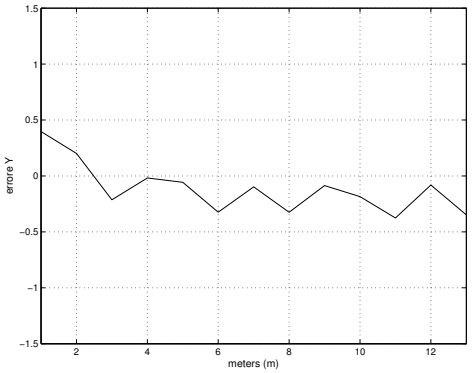
Figura C.6: Errori per la quinta traiettoria a singola telecamera.



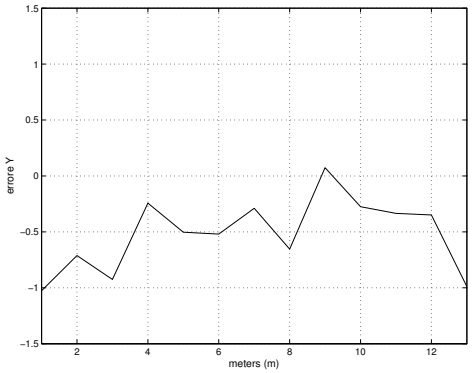
(a) *Tracking con Particle Smoother: errore x.*



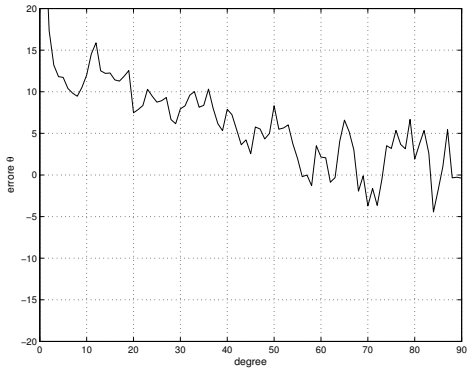
(b) *Tracking con Viterbi: errore x.*



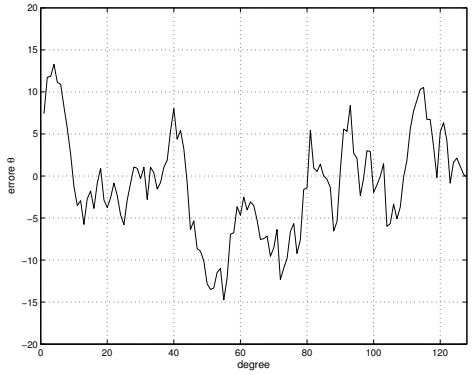
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*

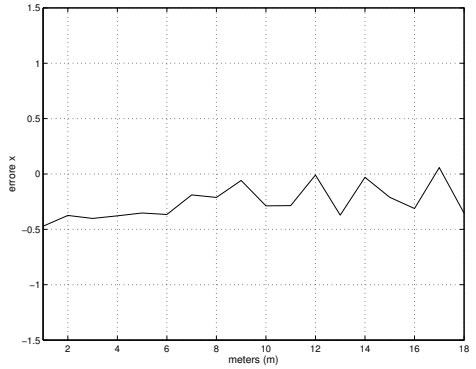


(e) *Tracking con Particle Smoother: errore θ.*

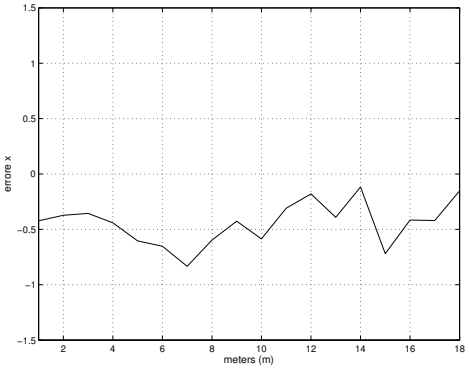


(f) *Tracking con Viterbi: errore θ.*

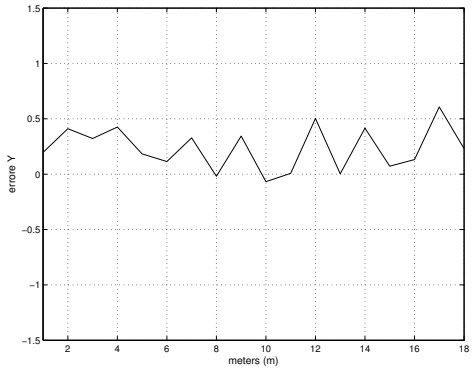
Figura C.7: Errori per la settima traiettoria a singola telecamera.



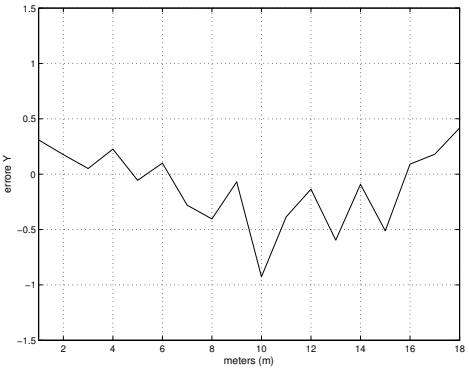
(a) *Tracking con Particle Smoother: errore x.*



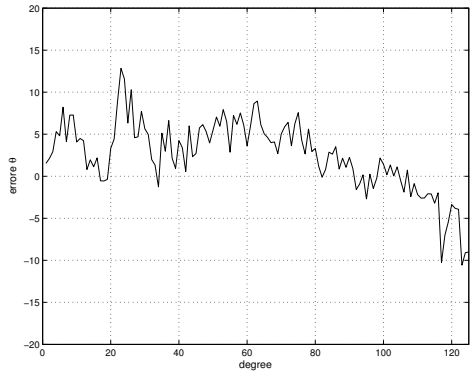
(b) *Tracking con Viterbi: errore x.*



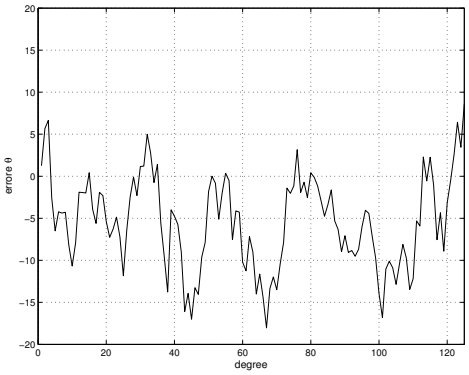
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*

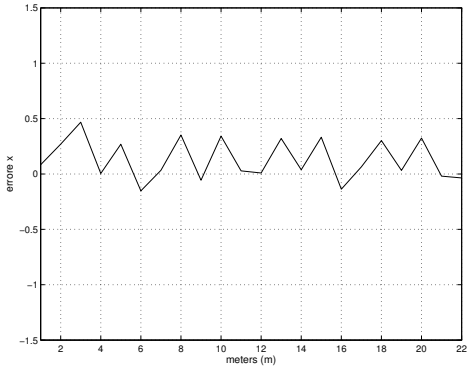


(e) *Tracking con Particle Smoother: errore θ.*

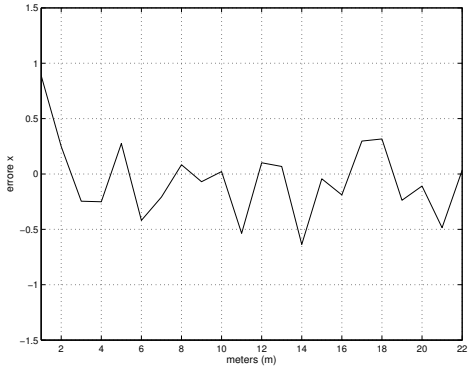


(f) *Tracking con Viterbi: errore θ.*

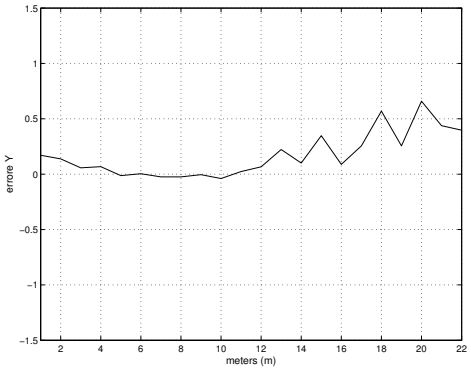
Figura C.8: Errori per la ottava traiettoria a singola telecamera.



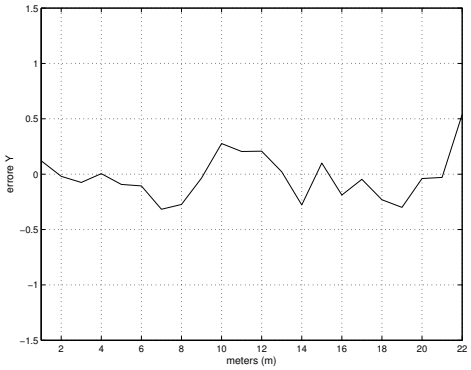
(a) *Tracking con Particle Smoother: errore x.*



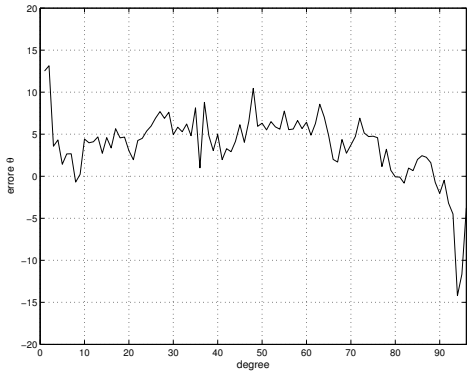
(b) *Tracking con Viterbi: errore x.*



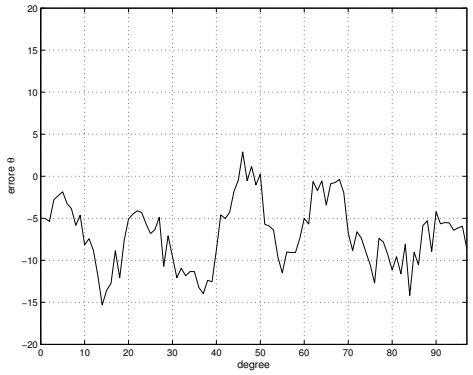
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*

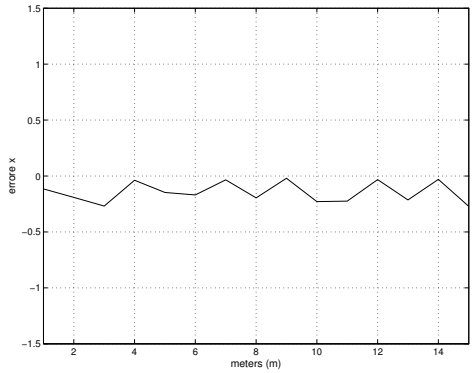


(e) *Tracking con Particle Smoother: errore θ.*

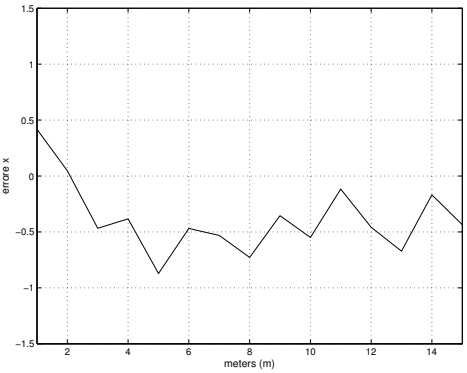


(f) *Tracking con Viterbi: errore θ.*

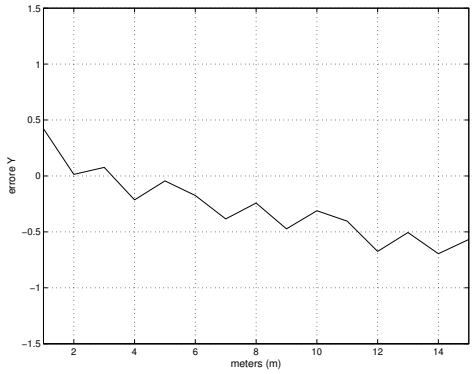
Figura C.9: Errori per la nona traiettoria a singola telecamera.



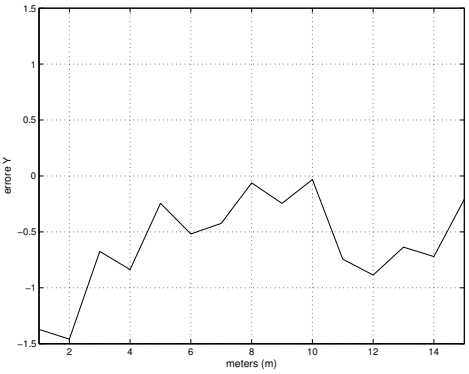
(a) *Tracking con Particle Smoother: errore x.*



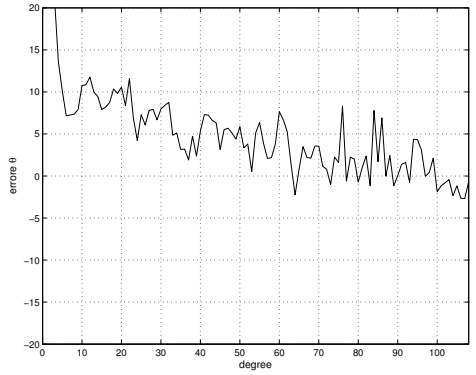
(b) *Tracking con Viterbi: errore x.*



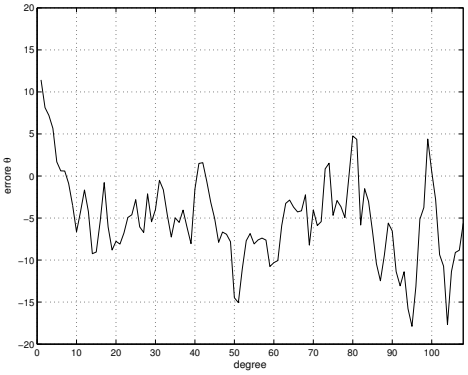
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*

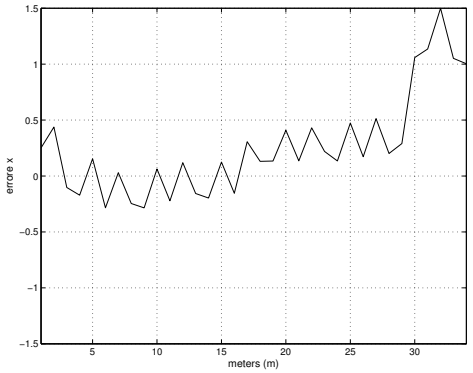


(e) *Tracking con Particle Smoother: errore θ.*

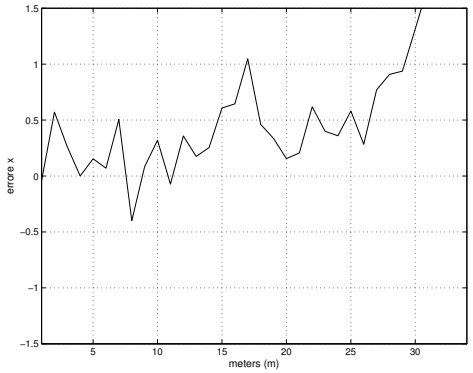


(f) *Tracking con Viterbi: errore θ.*

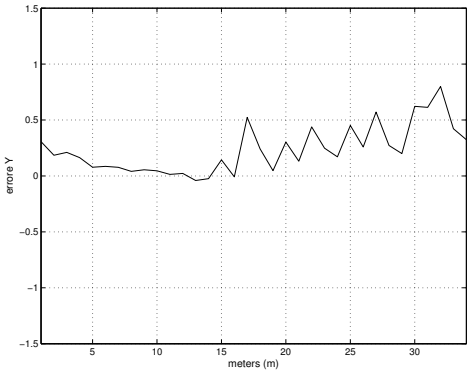
Figura C.10: Errori per la decima traiettoria a singola telecamera.



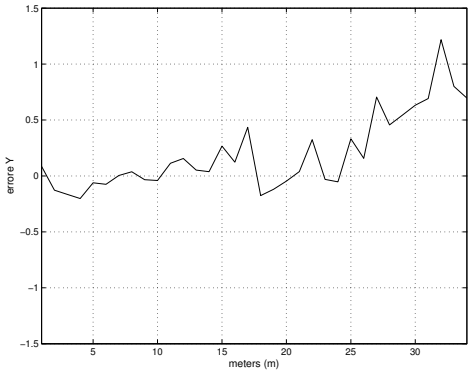
(a) *Tracking con Particle Smoother: errore x.*



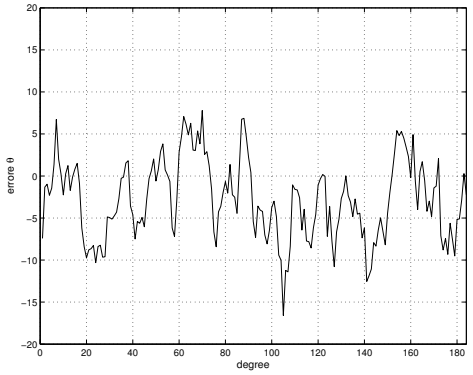
(b) *Tracking con Viterbi: errore x.*



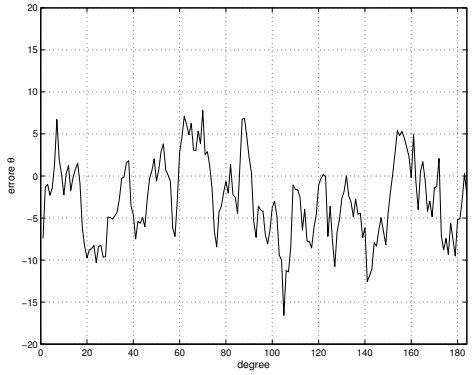
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*

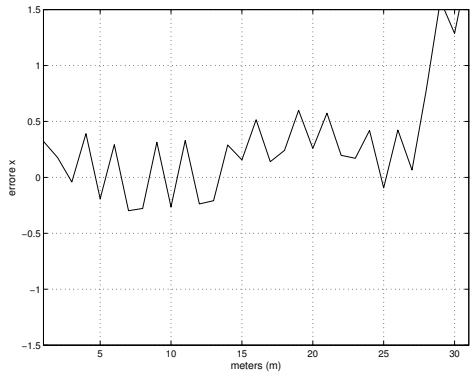


(e) *Tracking con Particle Smoother: errore θ.*

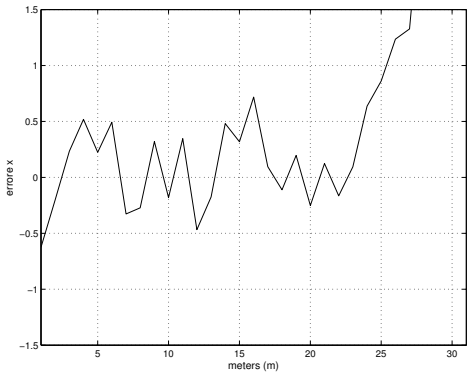


(f) *Tracking con Viterbi: errore θ.*

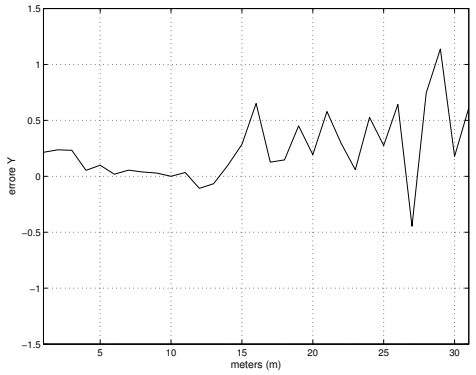
Figura C.11: Errori per la prima traiettoria con due telecamere.



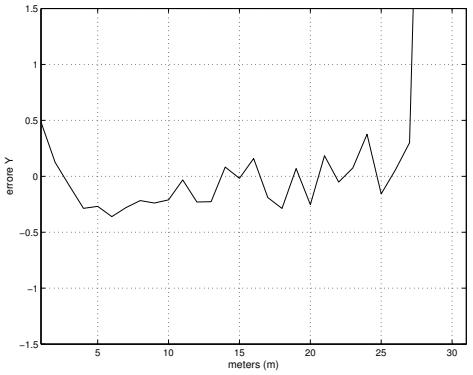
(a) *Tracking con Particle Smoother: errore x (errore medio* $0.017303m$*).*



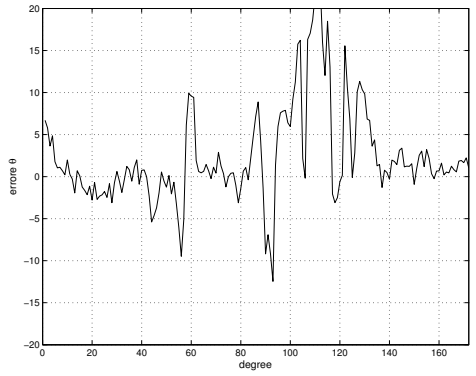
(b) *Tracking con Viterbi: errore x (errore medio* $0.04721m$*).*



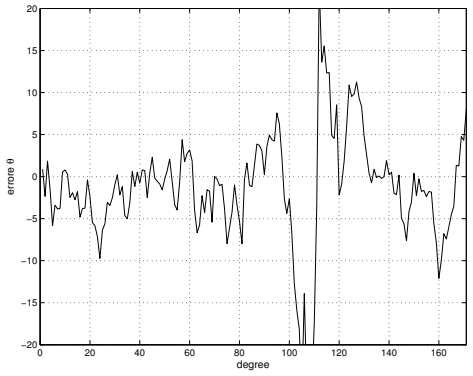
(c) *Tracking con Particle Smoother: errore y (errore medio* $0.024009m$*).*



(d) *Tracking con Viterbi: errore y.*

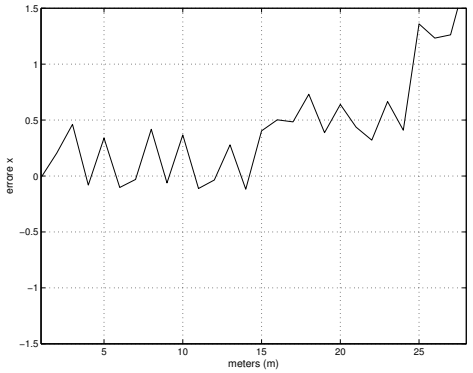


(e) *Tracking con Particle Smoother: errore* θ*.*

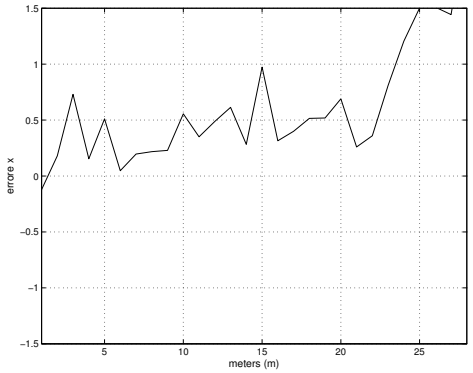


(f) *Tracking con Viterbi: errore* θ*.*

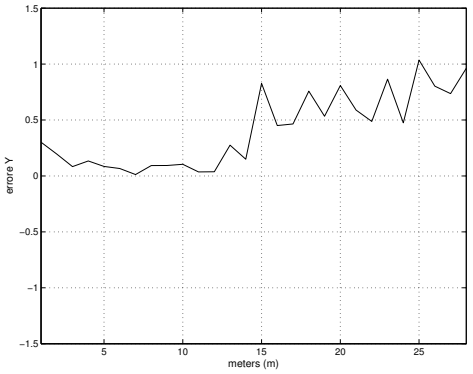
Figura C.12: Errori per la seconda traiettoria con due telecamere.



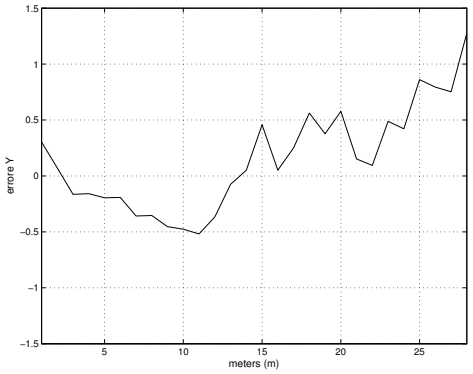
(a) *Tracking con Particle Smoother: errore x.*



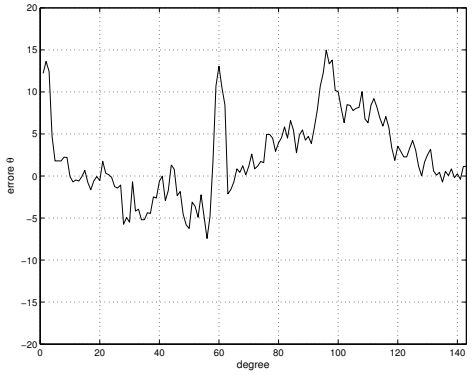
(b) *Tracking con Viterbi: errore x.*



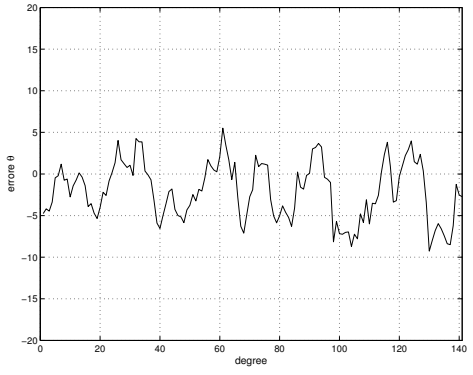
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*

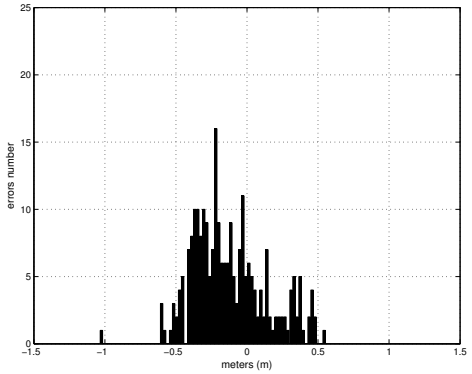


(e) *Tracking con Particle Smoother: errore θ.*

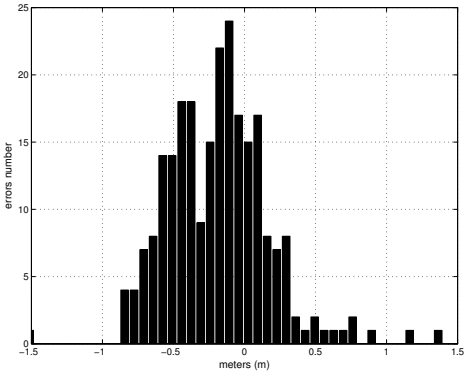


(f) *Tracking con Viterbi: errore θ.*

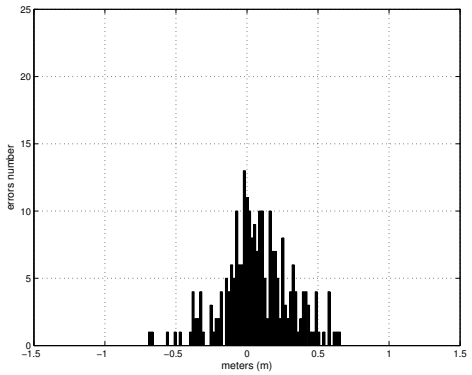
Figura C.13: Errori per la terza traiettoria con due telecamere.



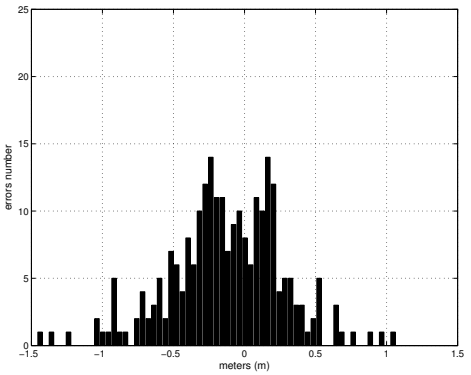
(a) *Tracking con Particle Smoother: errore x.*



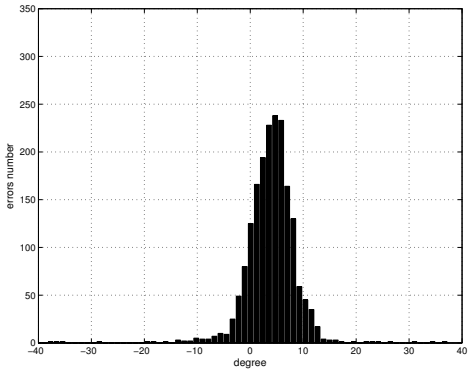
(b) *Tracking con Viterbi: errore x.*



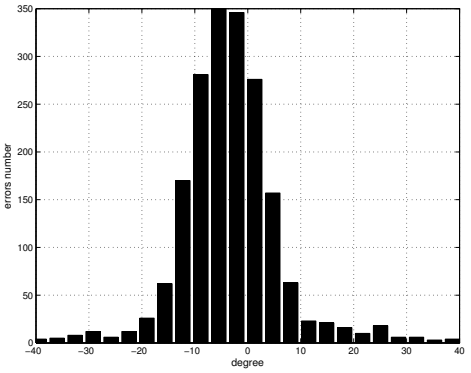
(c) *Tracking con Particle Smoother: errore y.*



(d) *Tracking con Viterbi: errore y.*

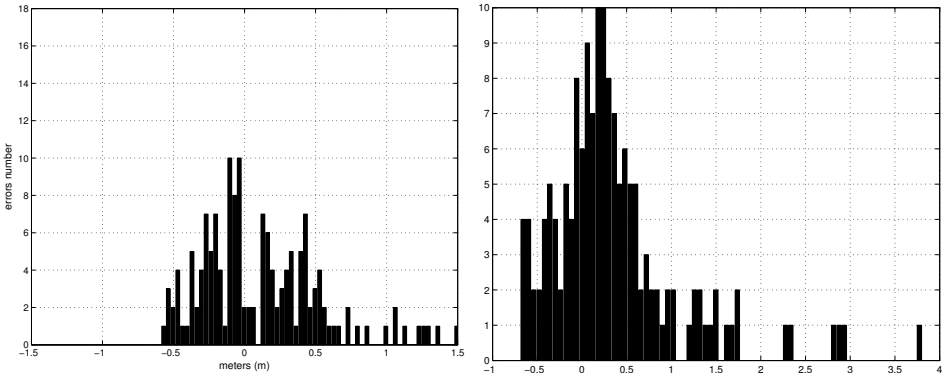


(e) *Tracking con Particle Smoother: errore θ.*

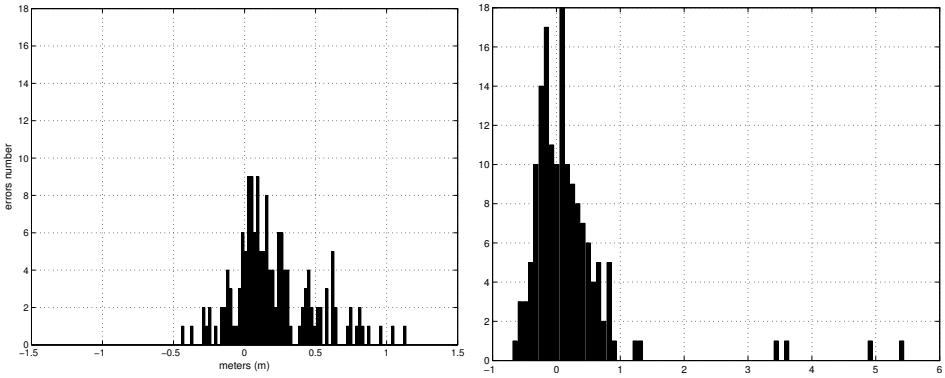


(f) *Tracking con Viterbi: errore θ.*

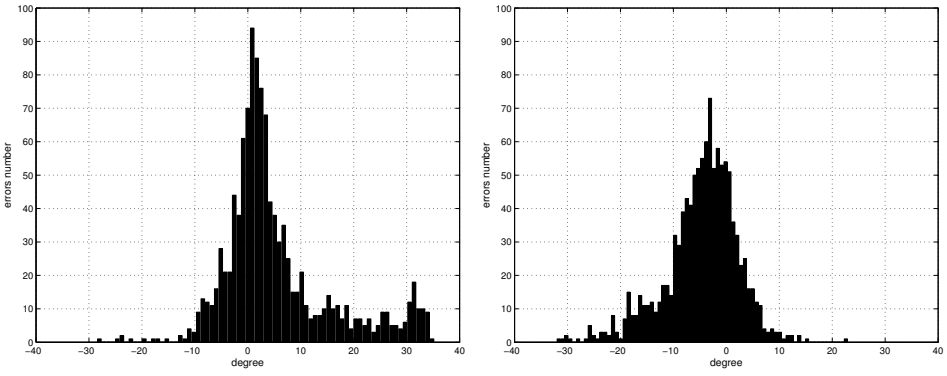
Figura C.14: Istogrammi degli errori per il tracking a singola telecamera.



(a) *Tracking con Particle Smoother: errore x.* (b) *Tracking con Viterbi: errore x.*



(c) *Tracking con Particle Smoother: errore y.* (d) *Tracking con Viterbi: errore y.*



(e) *Tracking con Particle Smoother: errore θ.* (f) *Tracking con Viterbi: errore θ.*

Figura C.15: Istogrammi degli errori per il tracking a multi camera.

Bibliografia

[1] A. Ambardekar, M. Nicolescu e G. Bebis. "Efficient Vehicle Tracking and Classification for an Automated Traffic Surveillance System". In: *Signal and Image Processing*. A cura di P. Cristea. Ago. 2008.

[2] M.S. Arulampalam et al. "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking". In: *Signal Processing, IEEE Transactions on* 50.2 (feb. 2002), pp. 174–188. ISSN: 1053-587X. DOI: 10.1109/78.978374.

[3] B. Aytekin e E. Altug. "Increasing driving safety with a multiple vehicle detection and tracking system using ongoing vehicle shadow information". In: *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*. Ott. 2010, pp. 3650–3656. DOI: 10.1109/ICSMC.2010.5641879.

[4] S. Basu, I. Essa e A. Pentland. "Motion regularization for model-based head tracking". In: *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*. Vol. 3. Ago. 1996, 611–616 vol.3. DOI: 10.1109/ICPR.1996.547019.

[5] S. Bittanti. *Serie Temporali e Processi Casuali*. Pitagora Editrice, 2005.

[6] J. Black e T. Ellis. "Multi camera image tracking". In: *Image and Vision Computing* 24.11 (2006). Performance Evaluation of Tracking and Surveillance, pp. 1256–1267. ISSN: 0262-8856. DOI: 10.1016/j.imavis.2005.06.002.

[7] J.A. Brown e D.W. Capson. "A Framework for 3D Model-Based Visual Tracking Using a GPU-Accelerated Particle Filter". In: *Visualization and Computer Graphics, IEEE Transactions on* 18.1 (gen. 2012), pp. 68–80. ISSN: 1077-2626. DOI: 10.1109/TVCG.2011.34.

[8] T. Brox et al. "Combined Region and Motion-Based 3D Tracking of Rigid and Articulated Objects". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 32.3 (mar. 2010), pp. 402–415. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2009.32.

[9] T. Brox et al. “High Accuracy Optical Flow Estimation Based on a Theory for Warping”. In: *Computer Vision - ECCV 2004*. A cura di Tomás Pajdla e Jirí Matas. Vol. 3024. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2004, pp. 25–36. ISBN: 978-3-540-21981-1. DOI: 10.1007/978-3-540-24673-2_3.

[10] N. Buch et al. “Urban Vehicle Tracking Using a Combined 3D Model Detector and Classifier”. In: *Knowledge-Based and Intelligent Information and Engineering Systems*. A cura di Juan Velásquez et al. Vol. 5711. Lecture Notes in Computer Science. 10.1007/978-3-642-04595-0_21. Springer Berlin / Heidelberg, 2009, pp. 169–176. ISBN: 978-3-642-04594-3. DOI: 10.1007/978-3-642-04595-0_21.

[11] J. Canny. “A Computational Approach to Edge Detection”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* PAMI-8.6 (nov. 1986), pp. 679–698. ISSN: 0162-8828. DOI: 10.1109/TPAMI.1986.4767851.

[12] D. Comaniciu, V. Ramesh e P. Meer. “Kernel-Based Object Tracking”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25 (2003), pp. 564–575. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2003.1195991.

[13] D. DeCarlo e D. Metaxas. “Optical Flow Constraints on Deformable Models with Applications to Face Tracking”. In: *International Journal of Computer Vision* 38 (2 2000), pp. 99–127. ISSN: 0920-5691. DOI: 10.1023/A:1008122917811.

[14] C. Dehais, G. Morin e V. Charvillat. “From rendering to tracking point-based 3D models”. In: *Image and Vision Computing* 28.9 (2010), pp. 1386–1395. ISSN: 0262-8856. DOI: 10.1016/j.imavis.2010.03.001. URL: http://www.sciencedirect.com/science/article/pii/S0262885610000405.

[15] A Doucet e A M Johansen. “A tutorial on particle filtering and smoothing: Fifteen years later”. In: *The Oxford Handbook of Nonlinear Filtering Oxford University Press To appear* December (2009), pp. 4–6.

[16] A. Elgammal, D. Harwood e L. Davis. “Non-parametric Model for Background Subtraction”. In: *Computer Vision — ECCV 2000*. A cura di David Vernon. Vol. 1843. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2000, pp. 751–767. ISBN: 978-3-540-67686-7. DOI: 10.1007/3-540-45053-X_48.

[17] J. M. Ferryman, A. D. Worrall e S. J. Maybank. “Learning Enhanced 3D Models for Vehicle Tracking”. In: *In BMVC*. 1998, pp. 873–882.

[18] J.M. Ferryman et al. “A generic deformable model for vehicle recognition”. In: *British Machine Vision Conference*. Vol. 1. Citeseer. 1995, pp. 127–136.

[19] G.David Jr. Forney. “The viterbi algorithm”. In: *Proceedings of the IEEE* 61.3 (mar. 1973), pp. 268–278. ISSN: 0018-9219. DOI: 10.1109/PROC.1973.9030.

[20] D. Fox et al. “Bayesian filters for location estimation”. In: *IEEE Pervasive Computing* 2.3 (2003), pp. 24–33.

[21] D.M. Gavrila e L.S. Davis. “3-D model-based tracking of humans in action: a multi-view approach”. In: *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR '96, 1996 IEEE Computer Society Conference on*. Giu. 1996, pp. 73–80. DOI: 10.1109/CVPR.1996.517056.

[22] M. Haag e H. H. Nagel. “Combination of Edge Element and Optical Flow Estimates for 3D-Model-Based Vehicle Tracking in Traffic Image Sequences”. In: *International Journal of Computer Vision* 35 (3 1999), pp. 295–319. ISSN: 0920-5691. DOI: 10.1023/A:1008112528134.

[23] M. Haag et al. “Influence of an Explicitly Modelled 3D Scene on the Tracking of Partially Occluded Vehicles”. In: *Computer Vision and Image Understanding* 65.2 (1997), pp. 206–225. ISSN: 1077-3142. DOI: 10.1006/cviu.1996.0575.

[24] C. Harris e C. Stennet. “RAPiD – A video-rate object tracker”. In: *British Machine Vision Conference*. Oxford, set. 1990, pp. 73–77.

[25] C. Harris e M. Stephens. “A combined corner and edge detector”. In: *Alvey Vision Conference* (1988), pp. 147–152.

[26] R. Hartley e A. Zisserman. *Multiple view geometry in computer vision*. Cambridge Univ Press, 2000.

[27] B. K.P. Horn e B. G. Schunck. “Determining optical flow”. In: *Artificial Intelligence* 17.1-3 (1981), pp. 185–203. ISSN: 0004-3702. DOI: 10.1016/0004-3702(81)90024-2.

[28] Z. Hu, C. Wang e K. Uchimura. “3D Vehicle Extraction and Tracking from Multiple Viewpoints for Traffic Monitoring by using Probability Fusion Map”. In: *Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE*. Vol. 3. Ott. 2007, pp. 30–35. DOI: 10.1109/ITSC.2007.4357665.

[29] B. Johansson et al. “Combining shadow detection and simulation for estimation of vehicle size and position”. In: *Pattern Recognition Letters* 30.8 (2009), pp. 751–759. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2009.03.005.

[30] W. L. Khong et al. “Enhancement of Particle Filter Approach for Vehicle Tracking Via Adaptive Resampling Algorithm”. In: *Computational Intelligence, Communication Systems and Networks (CICSyN), 2011 Third International Conference on*. Lug. 2011, pp. 259–263. DOI: 10.1109/CICSyN.2011.62.

[31] D. Koller. “Moving object recognition and classification based on recursive shape parameter estimation”. In: *Proc. 12th Israel Conf. Artificial Intelligence, Computer Vision*. Citeseer. 1993, pp. 27–28.

[32] D. Koller, K. Daniilidis e H. H. Nagel. “Model-based object tracking in monocular image sequences of road traffic scenes”. In: *International Journal of Computer Vision* 10 (3 1993), pp. 257–281. ISSN: 0920-5691. DOI: 10.1007/BF01539538.

[33] A. Kosaka e G. Nakazawa. “Vision-based motion tracking of frigid objects using prediction of uncertainties”. In: *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*. Vol. 3. Mag. 1995, 2637–2644 vol.3. DOI: 10.1109/ROBOT.1995.525655.

[34] P. Kumar et al. “Co-operative Multi-target Tracking and Classification”. In: *Computer Vision - ECCV 2004*. A cura di Tomás Pajdla e Jirí Matas. Vol. 3021. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2004, pp. 376–389. ISBN: 978-3-540-21984-2. DOI: 10.1007/978-3-540-24670-1_29.

[35] M.J. Leotta e J.L. Mundy. “Vehicle Surveillance with a Generic, Adaptive, 3D Vehicle Model”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 33.7 (lug. 2011), pp. 1457–1469. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2010.217.

[36] V. Lepetit e P. Fua. “Monocular Model-Based 3D Tracking of Rigid Objects: A Survey”. In: *Foundations and Trends in Computer Graphics and Vision*. 2005, pp. 1–89. ISBN: 1933019034. DOI: 10.1561/0600000001.

[37] H. Li, P. Roivainen e R. Forchheimer. “3-D motion estimation in model-based facial image coding”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 15.6 (giu. 1993), pp. 545–555. ISSN: 0162-8828. DOI: 10.1109/34.216724.

[38] J. Lou et al. “3-D model-based vehicle tracking”. In: *Image Processing, IEEE Transactions on* 14.10 (ott. 2005), pp. 1561–1569. ISSN: 1057-7149. DOI: 10.1109/TIP.2005.854495.

[39] M.I.A. Lourakis. “A brief description of the Levenberg-Marquardt algorithm implemented by levmar”. In: *matrix* 3 (2005), p. 2.

[40] D. G. Lowe. "Robust model-based motion tracking through the integration of search and estimation". In: *International Journal of Computer Vision* 8 (2 1992), pp. 113–122. ISSN: 0920-5691. DOI: 10.1007/BF00127170.

[41] B. D. Lucas e T. Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision". In: *Proceedings of Imaging Understanding Workshop*. 1981, pp. 121–130.

[42] S. Messelodi, C. Modena e M. Zanin. "A computer vision system for the detection and classification of vehicles at urban road intersections". In: *Pattern Analysis & Applications* 8 (1 2005), pp. 17–31. ISSN: 1433-7541. DOI: 10.1007/s10044-004-0239-9.

[43] R. Mohedano e N. Garcia. "Robust 3D multi-camera tracking from 2D mono-camera tracks by Bayesian association". In: *Consumer Electronics (ICCE), 2010 Digest of Technical Papers International Conference on*. Gen. 2010, pp. 313–314. DOI: 10.1109/ICCE.2010.5418780.

[44] E. Munoz, J.M. Buenaposada e L. Baumela. "Efficient model-based 3D tracking of deformable objects". In: *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*. Vol. 1. Ott. 2005, 877–882 Vol. 1. DOI: 10.1109/ICCV.2005.84.

[45] L. Mussone et al. "Traffic Analysis in Roundabout Intersections by Image Processing". In: *Proceedings of the 18th IFAC World Congress*. Vol. 18. Set. 2011. DOI: 10.3182/20110828-6-IT-1002.02626.

[46] N. El Nabbout. "Vehicle Tracking in Outdoor Environments using 3D Models". Tesi di laurea mag. University of Waterloo, 2008.

[47] A. Ottlik e H.H. Nagel. "Initialization of model-based vehicle tracking in video sequences of inner-city intersections". In: *International Journal of Computer Vision* 80.2 (2008), pp. 211–225.

[48] Michele Pellegrini e Andrea Romanoni. *Tracking e stima della traiettoria di veicoli in intersezioni rotatorie*. Set. 2009.

[49] M. Piccardi. "Background subtraction techniques: a review". In: *Systems, Man and Cybernetics, 2004 IEEE International Conference on*. Vol. 4. Ott. 2004, 3099–3104 vol.4. DOI: 10.1109/ICSMC.2004.1400815.

[50] D. Ponsa et al. "Multiple vehicle 3D tracking using an unscented Kalman". In: *Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE*. Set. 2005, pp. 1108–1113. DOI: 10.1109/ITSC.2005.1520206.

[51] S. Pundlik e S. Zadgaonkar. "Vehicle boundary detection and tracking". In: *Dept. of Electrical and Computer Engineering, Clemson University* (2004).

[52] D. Reid. “An algorithm for tracking multiple targets”. In: *Automatic Control, IEEE Transactions on* 24.6 (1979), pp. 843–854.

[53] A. Ruf et al. “Visual tracking of an end-effector by adaptive kinematic prediction”. In: *Intelligent Robots and Systems, 1997. IROS ’97., Proceedings of the 1997 IEEE/RSJ International Conference on*. Vol. 2. Set. 1997, 893–899 vol.2. DOI: 10.1109/IROS.1997.655115.

[54] J. Scharcanski et al. “A Particle-Filtering Approach for Vehicular Tracking Adaptive to Occlusions”. In: *Vehicular Technology, IEEE Transactions on* 60.2 (feb. 2011), pp. 381–389. ISSN: 0018-9545. DOI: 10.1109/TVT.2010.2099676.

[55] C. Shan et al. “Real time hand tracking by combining particle filtering and mean shift”. In: *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*. Mag. 2004, pp. 669–674. DOI: 10.1109/AFGR.2004.1301611.

[56] J. Shi e C. Tomasi. “Good features to track”. In: *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR ’94., 1994 IEEE Computer Society Conference on*. Giu. 1994, pp. 593–600. DOI: 10.1109/CVPR.1994.323794.

[57] X. Song e R. Nevatia. “A model-based vehicle segmentation method for tracking”. In: *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*. Vol. 2. Ott. 2005, 1124–1131 Vol. 2. DOI: 10.1109/ICCV.2005.11.

[58] X. Song e R. Nevatia. “Detection and Tracking of Moving Vehicles in Crowded Scenes”. In: *Motion and Video Computing, 2007. WMVC ’07. IEEE Workshop on*. Feb. 2007, p. 4. DOI: 10.1109/WMVC.2007.13.

[59] C. Stauffer e W.E.L. Grimson. “Adaptive background mixture models for real-time tracking”. In: *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.* Vol. 2. 1999, 2 vol. (xxiii+637+663). DOI: 10.1109/CVPR.1999.784637.

[60] T.N. Tan, G.D. Sullivan e K.D. Baker. “Model-Based Localisation and Recognition of Road Vehicles”. In: *International Journal of Computer Vision* 27 (1 1998). 10.1023/A:1007924428535, pp. 5–25. ISSN: 0920-5691.

[61] C. Tomasi e T. Kanade. *Detection and tracking of point features*. Rapp. tecn. Carnegie Mellon University, apr. 1991.

[62] C. Tomasi, S. Petrov e A. Sastry. “3d tracking= classification+ interpolation”. In: *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*. IEEE. 2003, pp. 1441–1448.

[63] B. Tordoff et al. “Head pose estimation for wearable robot control”. In: *British Machine Vision Conference*. Citeseer. 2002, pp. 807–816.

[64] A. Tyagi et al. “Fusion of Multiple Camera Views for Kernel-Based 3D Tracking”. In: *Motion and Video Computing, 2007. WMVC ’07. IEEE Workshop on*. Feb. 2007, p. 1. DOI: 10.1109/WMVC.2007.15.

[65] L. Vacchetti, V. Lepetit e P. Fua. “Combining Edge and Texture Information for Real-Time Accurate 3D Camera Tracking”. In: *Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality*. ISMAR ’04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 48–57. ISBN: 0-7695-2191-6. DOI: 10.1109/ISMAR.2004.24.

[66] L. Vacchetti, V. Lepetit e P. Fua. “Stable real-time 3d tracking using online and offline information”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 26.10 (2004), pp. 1385–1391.

[67] C.S. Wiles et al. “Hyper-patches for 3D model acquisition and tracking”. In: *cvpr*. Published by the IEEE Computer Society. 1997, p. 1074.

[68] H. Yang et al. “Efficient and robust vehicle localization”. In: *Image Processing, 2001. Proceedings. 2001 International Conference on*. Vol. 2. Ott. 2001, 355–358 vol.2. DOI: 10.1109/ICIP.2001.958501.

[69] Z. Zhang et al. “3D model based vehicle localization by optimizing local gradient based fitness evaluation”. In: *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*. Dic. 2008, pp. 1–4. DOI: 10.1109/ICPR.2008.4761603.

[70] Zhaoxiang Zhang et al. “3D Model Based Vehicle Tracking Using Gradient Based Fitness Evaluation under Particle Filter Framework”. In: *Pattern Recognition (ICPR), 2010 20th International Conference on*. Ago. 2010, pp. 1771–1774. DOI: 10.1109/ICPR.2010.437.

[71] Z. Zivkovic e F. van der Heijden. “Efficient adaptive density estimation per image pixel for the task of background subtraction”. In: *Pattern Recognition Letters* 27.7 (2006), pp. 773–780. ISSN: 0167-8655. DOI: DOI:10.1016/j.patrec.2005.11.005.