Analisi del movimento della camera e tracking degli oggetti in un video utilizzando i descrittori SIFT

Sandro Bernardinello

Università di Padova — 5 Ottobre 2010

Ringraziamenti I ringraziamenti... è sempre difficile scrivere i ringraziamenti! Comunque sia le persone che devo ringraziare sono tante, innanzitutto ringrazio la mia famiglia, papà, mamma e sorella, perchè se sono arrivato a questo importante traguardo è solo perchè loro me lo hanno permesso,supportandomi e non dicendo mai di no, anche quando gli ho detto che sarei andato in Giappone per sei mesi. Quindi grazie di cuore perchè sono stati gli anni più belli della mia vita ed un'esperienza indimenticabile. Grazie alla mia ragazza, Sara, perchè è stata la persona che più mi è stata vicina, che più mi ha aiutato, che ha sempre fatto per me tutto il possibile, e anche di più, che mi ha supportato e sopportato in questi anni e per ben due lauree e che non si è mai tirata indietro se avevo un problema, perchè è una persona speciale, grazie di cuore veramente! Ringrazio i miei vecchi coninquilini, il Fox, il Trevi, il Balto, il Poles e il Boss per avermi regalato tutti questi anni pieni di risate, gioie, dolori e tanto tanto divertimento! Ringrazio le Vicine per avermi sempre dato alloggio in questo ultimo anno e per tutte le cene, le serate e le risate che abbiamo fatto insieme, grazie mille in particolare a Marta, Lalli, Eli e Giulia, per tutto ma sopratutto per la vostra amicizia! Ringrazio tutti i miei amici dell'università per tutto quello che hanno fatto per me e per tutte le giornate passate insieme a studiare e a divertirci e a cazzeggiare e a bere spritz: grazie Mauro, Biondone, Menny, Silvia cattiva, Teo, Bambi, Totò, Mandi e a tutti gli altri: grazie!. Ringrazio la Francy perchè è un'amica speciale, per tutte le lunghe chiaccherate che ci siamo fatti, per le circa sei milioni di pause caffè fatte insieme e i gelati in terrazza la sera, per tutte le volte che mi è stata vicina quando ce n'era bisogno e per tutto quello che ha fatto per me, grazie Francy!

Ringrazio l'Enri, per tutte le belle giornate passate insieme, per le serate a fare stronzate in giro per la città, per tutte le volte che mi hai aiutato e mi hai fatto ridere e sopratutto grazie per la tua amicizia! Ringrazio tutti gli amici di Verona, Giorgione, Vale, Spiaggi, Ste, Cos, Saretta, Baracca, Giugi, Giovi e ancora Sofy, Teo, Pdo, Celi, Dani, Piovra, Ale, Giuli, Dony e tutti gli altri perchè chi più chi meno hanno contribuito a regalarmi dei bei ricordi e sono stati una parte importante della mia vita. Per tutte le belle cose fatte insieme, per tutte le super serate, per tutti i momenti belli e brutti, i concerti, i viaggi, le vacanze, le esperienze e tutto il tempo passato insieme, veramente grazie mille! Ringrazio il Prof. Zanuttigh per avermi aiutato in tutto questo tempo e per essere stato sempre presente e disponibile. E ancora grazie al Dei, all'università, alle aule studio, alle mense, alle piazze del centro, ai tre appartamenti in cui ho vissuto, agli spritz, agli aperitivi, all'a&o sotto casa, al bar12 e la sua promozione brioche gratis con il caffe e a tutto ciò che ha fatto parte di questi anni passati qui. Un grazie speciale alla mia maestra delle elementari Coraly per tutto quello che mi ha insegnato e perchè so che sarebbe stata contenta di vedermi laureato. Un grazie a tutti quelli che mi hanno aiutato, che mi hanno voluto bene e che hanno contribuito a rendere questi anni un'esperienza speciale! Grazie di cuore!

Indice

1 Introduzione **1**

2 SIFT: Scale Invariant Feature Transform **5**

2.1 Algoritmo 6

2.2 Ricerca della corrispondenza tra gruppi di feature 11

3 Motion tracking **13**

3.1 Filtro di Kalman 14

4 Applicativo **17**

4.1 La struttura dell'applicativo 17

4.2 Estrazione delle feature 22

4.3 Ricerca delle corrispondenze 24

4.4 Rilevamento degli oggetti 27

4.4.1 La segmentazione d'immagine 28

4.4.2 Identificazione degli oggetti e tracciamento della posizione 29

4.5 Calcolo e visualizzazione dei vettori di moto 31

5 Risultati sperimentali **35**

6 Conclusioni e sviluppi futuri **45**

A Appendice **51**

A.1 File principali 52

A.2 Struct *feature* 53

A.3 Struct *featureTrace* 54

A.4 Struct *object* 55

Bibliografia **57**

Capitolo 1

Introduzione

Nel campo dell'elaborazione delle immagini uno degli obbiettivi principali è quello di riuscire ad estrapolare quelle che sono le informazioni fondamentali per caratterizzare e descrivere nel modo più univoco possibile l'immagine stessa. A questo proposito esistono molti algoritmi e descrittori che si basano su diversi aspetti caratterizzanti un'immagine come ad esempio il colore o l'intensità dei pixel, la presenza di zone con contrasto elevato, la presenza di linee e così via. Se spostiamo la nostra attenzione su un video, il che non è altro che una sequenza di immagini, è naturale che descrivere ogni singola immagine non è più sufficiente ma è necessario andare a capire quali sono le caratteristiche che connettono un'immagine alla successiva e vedere come esse cambiano nel tempo ottenendo così un'analisi della "dinamica" della sequenza stessa.

Lo scopo di questo lavoro è appunto quello di analizzare un video di una scena reale e, a fronte di alcune ipotesi, in questo riconoscere ciò che si sta muovendo e tener traccia del movimento nell'evolversi della scena nel tempo. Le situazioni possibili a questo punto sono tre:

- La scena filmata è dinamica con oggetti in movimento e la telecamera

è ferma in una posizione fissa.

- La scena è statica mentre la telecamera si muove.

- Sia la scena che la telecamera possono essere in movimento.

Nel contesto attuale sono presenti diverse tecniche per l'object tracking e/o motion detection a seconda di quale delle tre situazioni si deve analizzare. Nel primo caso si utilizzano fondamentalmente metodi di background substraction per individuare quali sono le parti "dinamiche" all'interno di un frame video andando a calcolare la differenza tra frame successivi evidenziando così solo i pixel che variano e successivamente tecniche di edge detection, thresholding, histogram comparison, ecc... per identificare univocamente ogni singola parte di cui tener traccia. Nel secondo caso è necessario che tutta la scena sia considerata statica così da poter analizzare come tutta l'immagine si muova e da questo movimento calcolare, previa conoscenza del modello geometrico della camera, come si muove la camera stessa. Nell'ultimo caso entrambi gli approcci precedenti falliscono perchè il movimento individuato, ad esempio, su un oggetto realmente in movimento all'interno della scena va a sfalsare il movimento della scena globale utilizzato per il calcolo del movimento della camera; inoltre se la camera sta seguendo l'oggetto in movimento, questo risulterebbe fermo ad un'analisi di questo tipo mentre in realtà non lo è.

Quello che questo studio si propone è di analizzare quindi una scena completamente dinamica, ponendo delle ipotesi essendo questo problema abbastanza complesso. L'approccio che ci si propone non richiede nessun input da parte dell'utente il che implica due principali fasi:

- Distinzione tra foreground, ovvero gli oggetti in movimento, e background, ovvero quella che è la scena statica in cui gli oggetti si muovono.
- Individuazione degli oggetti tra un frame e l'altro, calcolo del movimento degli oggetti e della camera e tracciamento dello stesso.

La fase di individuazione del background non può essere affrontata in modo standard utilizzando il metodo di background substraction che prevede di calcolare la differenza pixel a pixel tra un frame video e il precedente cosi da ottenere solo i pixel che sono cambiati ovvero quelli appartenenti alle parti in movimento, questo perchè avendo una scena completamente dinamica tutti i pixel possono cambiare. L'approccio scelto si basa sulla segmentazione d'immagine e utilizza l'algoritmo proposto da Pedro F. Felzenszwalb in [8] che permette di suddividere l'immagine in regioni con caratteristiche simili e che andranno a costituire poi i nostri oggetti di cui tracciare la posizione nel tempo. Per individuare quale tra le regioni sia da considerare il background, si pone l'ipotesi che questa sia la regione con l'area più grande.

Questa fase di identificazione viene effettuata, oltre che all'inizio, ogni qualvolta l'applicazione individua un cambio di scena nel video o quando tutti gli oggetti, la cui posizione si stava tracciando, sono usciti di scena o se non sono state trovate corrispondenze con le feature associate ad essi.

La fase successiva consiste nel confronto tra frame successivi per l'individuazione del movimento dell'intera scena sia come moto "assoluto" ovvero quello della videocamera, sia come moto "relativo" ovvero quello degli oggetti in movimento rispetto alla scena e alla camera stesse.

Il cuore di questo passo è rappresentato dall'estrazione delle feature d'immagine utilizzando la **Scale Invariant Feature Transform - SIFT**; la scelta di questa tipologia di feature è dovuta alle sue varie proprietà di inva-

rianza rispetto a trasformazioni geometriche quali rotazioni, trasformazioni affini e proiezioni, a scalatura e a parziali cambi di illuminazione e spostamenti del punto di osservazione. Sulla base delle informazioni ricavate dalle feature di frame successivi vengono cercate le corrispondenze tra esse e da queste calcolati i vettori di movimento; successivamente vengono trovate le corrispondenze tra gli insiemi di feature rappresentanti gli oggetti precedentemente individuati e l'insieme di feature del frame corrente, infine vengono calcolati movimento relativo ed assoluto.

Questa seconda fase può essere riassunta come segue:

- estrazione delle SIFT
- matching delle feature tra gli ultimi N frame successivi
- aggiornamento della posizione delle feature di cui si sta tenendo traccia utilizzando il fitro di Kalman
- calcolo per ogni corrispondenza trovata del vettore di movimento, del suo modulo e direzione
- identificazione di ogni singolo oggetto utilizzando le corrispondenze tra le feature associate all'oggetto nel frame precedente e l'insieme delle feature del nuovo frame
- calcolo del movimento relativo di tutti gli oggetti, della camera e stima del movimento reale degli oggetti nella scena

Capitolo 2

SIFT: Scale Invariant Feature Transform

La Scale Invariant Feature Transform - SIFT[1] è un algoritmo che serve per estrarre da un'immagine un set di feature, letteralmente caratteristiche, che descrivono univocamente l'immagine stessa. Questo tipo di approccio ha dimostrato di avere diversi vantaggi in varie applicazioni tipiche della computer vision; alcuni esempi sono l'object recognition, visual mapping e navigation applicate alla robotica autonoma, image stiching, gesture recognition, motion tracking, ricostruzione di ambienti 3D e tante altre. La principale differenza, ed il suo punto di forza, tra questo tipo di descrittore e tanti altri comunemente usati nella computer vision, è data dalla sua dimostrata invarianza da trasformazioni quali traslazioni, rotazioni, trasformazioni affini, prospettive e dalla parziale invarianza rispetto a cambiamenti nell'illuminazione e spostamenti del punto di osservazione della scena; sono inoltre descrittori altamente distintivi e ben localizzati sia nel dominio della frequenza sia nel dominio dello spazio e permettono quindi di trovare corrispondenze corrette tra essi con alta probabilità anche in presenza di un certo livello di rumore o

di occlusione rispetto alla scena originale. Ogni singola feature è rappresentata univocamente da un descrittore formato da un vettore a 128 dimensioni le cui componenti sono calcolate in base alle caratteristiche più descrittive della feature stessa.

2.1 Algoritmo

Per calcolare questo descrittore l'immagine è innanzitutto filtrata mediante convoluzioni gaussiane, creando così il cosidetto "*scale space*". Per ogni scala vengono quindi calcolate le "*DoG, Difference of Gaussian*", differenze tra gaussiane adiacenti, i cui massimi rappresentano i "*keypoint*", punti di interesse a cui ogni feature fa riferimento. Il punto di forza di questo tipo di feature è appunto dato dal fatto che ricercando questi punti sia nello spazio delle scale sia sulle immagini originali si ottiene che i keypoints estratti risultano invarianti alla scalatura dell'immagine.

Per minimizzare il costo computazionale del processo di estrazione delle feature si applica un approccio cosidetto "a cascata" in modo tale che le operazioni più costose in termini di risorse siano eseguite solo sui punti rimasti dopo le fasi precedenti e quindi con più elevato "potere descrittivo".

Le fasi principali dell'algoritmo che calcola questa "trasformazione" (così chiamata perchè trasforma un'immagine in una serie di coordinate) sono i seguenti:

1. **Scale-space extrema detection:** l'individuazione delle regioni invarianti a cambi di scala e che possono quindi essere associate anche con diversi punti di vista dello stesso oggetto, viene effettuata utilizzando una funzione di scala conosciuta come "scale-space" e definita

come segue:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y),$$

dove $*$ è l'operatore di convoluzione e

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

Questa funzione è definita come la convoluzione tra una gaussiana e l'immagine di input e per calcolare efficientemente le regioni invarianti su scale diverse viene applicata alla differenza di gaussiane adiacenti, " DoG - Difference of Gaussians" che può essere calcolata facendo una semplice differenza tra immagini(Figura 2.1).

$$\begin{aligned} DoG(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma). \end{aligned}$$

Per individuare i massimi e minimi locali della funzione ogni punto è

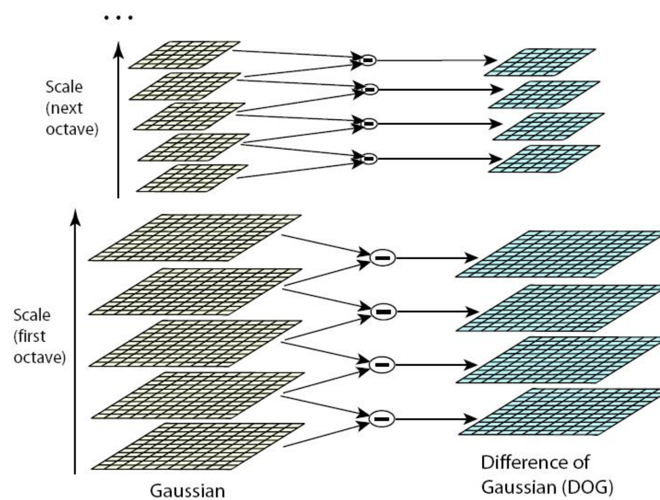


Figura 2.1: Calcolo della DoG.

confrontato con i suoi otto vicini nell'immagine e con i suoi nove vicini nella scala superiore ed inferiore ed è assegnato come massimo se

è superiore a tutti gli stessi o come minimo se inferiore(Figura 2.2). Mediante risultati sperimentali si sa che ci sono un numero molto ele-

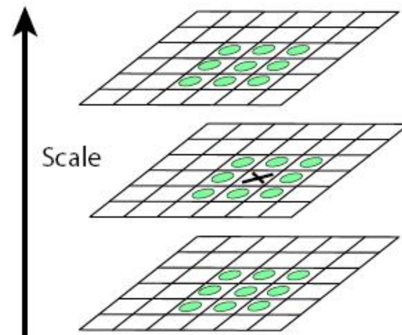


Figura 2.2: Ricerca degli estremi della DoG.

vato di estremi il cui calcolo avrebbe un costo computazionalmente troppo alto, a fronte di ciò sapendo che estremi adiacenti sono instabili a piccole variazioni locali dell'immagine, dovute ad esempio a presenza di rumore, è stato quindi calcolato sperimentalmente un limite nel numero di scale per ogni ottava in cui cercare gli estremi in modo tale da ottenere il giusto equilibrio tra tempo d'esecuzione e completezza dei risultati.

2. **Accurate keypoint localization:** questa seconda fase ha come obiettivo, una volta trovati i keypoint "candidati", di cercare quali tra questi sono punti con basso contrasto, e quindi più sensibili al rumore d'immagine, o quali sono collocati lungo un spigolo. Inizialmente il filtraggio dei punti di minor interesse era effettuato ponendo i keypoint nella posizione e scala del campione centrale della regione corrente. Successivamente è stato introdotto da Brown un metodo che utilizza l'espansione di Taylor al termine quadratico della funzione "scale-space" traslata in modo che l'origine coincida con il campione

centrale; calcolando ora la derivata in zero della stessa si ottiene la posizione dell'estremo; il valore della funzione proposta da Brown nell'estremo risulta essere un valore utile per eliminare punti con basso contrasto(Figura 2.3).

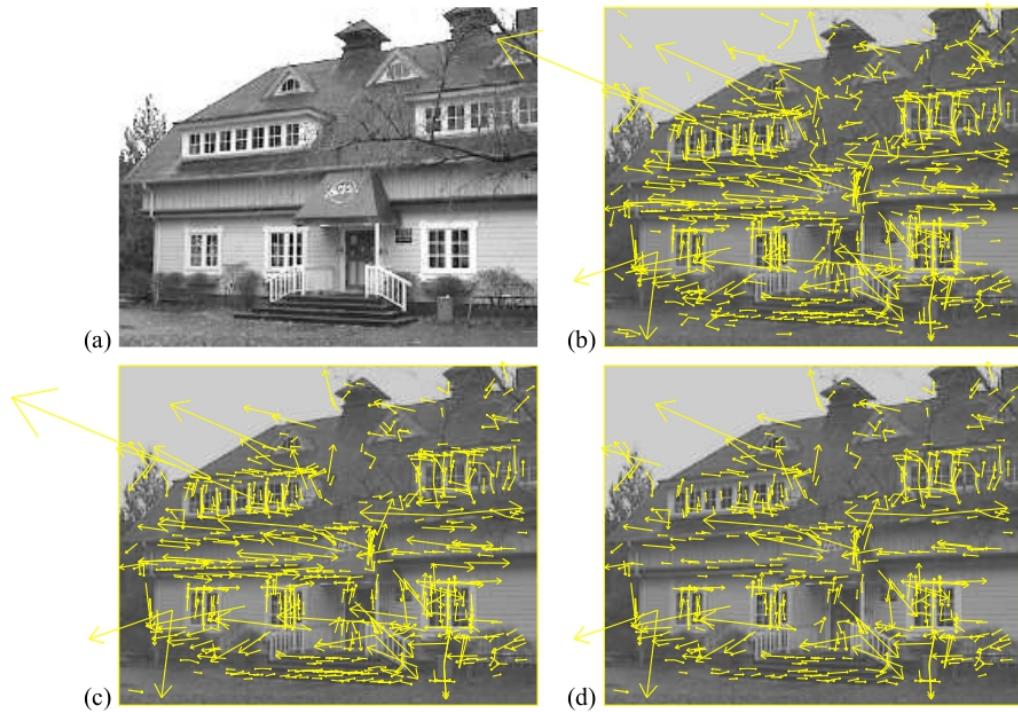


Figura 2.3: Fasi di filtraggio dei keypoint: a) immagine iniziale; b) tutti i massimi e minimi della DoG; c) dopo aver eliminato i punti a basso contrasto; d) dopo l'eliminazione dei punti localizzati lungo i bordi.

Come accennato precedentemente, oltre ai punti a basso contrasto, per la stabilità è necessario eliminare anche i keypoint localizzati lungo uno spigolo(Figura 2.3); la funzione di "Difference of Gaussian" ha una forte risposta e quindi una forte curvatura in corrispondenza di un lato/edge ma una debole risposta nella direzione perpendicolare a questo. Calcolando gli autovalori dell'Hessiano della funzione proposta

da Brown si possono ottenere le due curvature, ma il valore che a noi interessa è semplicemente il rapporto tra le due; è possibile dimostrare che questo stesso rapporto è proporzionale al rapporto tra il quadrato della traccia dell'Hessiano e il suo determinante. Ponendo ora una soglia il cui valore ottimale è stato calcolato sperimentalmente, si avrà che se il rapporto è maggiore il keypoint viene eliminato. La matrice Hessiana ed il rapporto sono riportati di seguito.

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r}.$$

3. **Orientation assignment:** lo scopo di questa fase è quello di ottenere l'invarianza del descrittore della feature alla rotazione dell'immagine. L'idea consiste nell'assegnare una orientazione ad ogni keypoint che dipenda esclusivamente da proprietà dell'immagine nell'intorno del punto stesso. Utilizzando la differenza tra pixel vengono calcolati modulo e angolo del gradiente dell'immagine e viene poi formato un istogramma contenente 36 diverse orientazioni prese dai campioni nella regione attorno al keypoint. Successivamente viene ricercato il picco massimo all'interno dell'istogramma stesso e per ogni altro picco con valore maggiore dell'80% del massimo si crea un nuovo keypoint con uguale posizione ma diversa orientazione; questo metodo comporta un notevole aumento della stabilità.

4. **Keypoint descriptor computation:** per calcolare il descrittore di un keypoint vengono prima calcolati tutti i moduli e le orientazioni dei

gradienti di ogni punto campione in un intorno del keypoint, successivamente questi vengono pesati utilizzando una finestra gaussiana in modo da enfatizzare il valore descrittivo dei gradienti vicini al centro della regione, il che aumenta la stabilità diminuendo la sensibilità del descrittore a piccoli cambiamenti nella posizione della finestra. Ogni orientazione viene poi inserita in un istogramma per ogni regione di 4x4 campioni ottenendo così che ogni campione può subire uno shift massimo di 4 posizioni andando ancora a contribuire allo stesso descrittore, questo garantisce un ottima invarianza alle distorsioni dell'immagine. Nell'algoritmo viene utilizzato un array di 4x4 istogrammi d'orientazione ognuno con 8 orientazioni all'interno; un descrittore sarà quindi formato da un vettore in 4x4x8 = 128 dimensioni contenente tutti i valori salvati negli istogrammi stessi. Per garantire poi una certa invarianza a cambi di illuminazione il vettore viene normalizzato all'unità, il che diminuisce la sensibilità a variazioni uniformi di contrasto e luminosità; sperimentalmente si è visto che andando a filtrare tutti i valori al di sopra di 0.2 e rinormalizzando all'unità il vettore si ottiene un miglioramento nell'invarianza a cambiamenti di contrasto e luminosità non lineari.

2.2 Ricerca della corrispondenza tra gruppi di feature

Nella ricerca delle corrispondenze tra due insiemi di feature si utilizza un sistema di indicizzazione basato sul **k-dtree** che velocizza notevolmente la ricerca in grandi collezioni di feature. Questo processo di indicizzazione è reso necessario dal fatto che il descrittore ha un numero elevato di dimen-

sioni e il calcolo della distanza tra ogni coppia di feature avrebbe un costo computazionalmente troppo elevato. Il metodo di ricerca utilizzato è detto *Best-Bin-First* e si basa su una versione modificata del kd-tree che trova la miglior corrispondenza, ossia il "nearest neighbor" con alta probabilità; il criterio utilizzato per calcolare la distanza tra descrittori è la semplice distanza euclidea tra i due vettori. Per migliorare l'efficienza della ricerca è inoltre stato scelto un limite al massimo numero di "vicini" da ricercare pari a 200 dopo il quale la ricerca viene interrotta.

È importante notare che per diminuire la possibilità di trovare false corrispondenze invece che utilizzare una soglia sulla distanza minima con il primo vicino, si utilizza il rapporto tra la distanza con il primo e con il secondo "nearest neighbor"; da prove sperimentali è stato trovato che un valore pari a 0.8 filtra circa il 90% delle false corrispondenze compromettendo solo il 5% di quelle corrette.

Capitolo 3

Motion tracking

Il motion tracking consiste fondamentalmente nell'estrapolazione del movimento in un video, sia esso il movimento di un oggetto nella scena o il movimento della telecamera. Le problematiche che si presentano sono molte e spesso la difficoltà di identificare con precisione l'oggetto di cui si sta tracciando il movimento può portare ad errori che rendono difficile il corretto calcolo della traiettoria e il tracciamento della posizione nel tempo; in particolare l'imprecisione aumenta se il rapporto tra la velocità di movimento dell'oggetto e il frame-rate del video è alto ovvero se le variazioni di posizione tra un frame e il successivo sono troppo marcate e non lineari.

Per far fronte a questo tipo di problematica è necessario utilizzare un "modello del moto" che descriva come come l'oggetto da tracciare si possa muovere e adattare il tracciamento di conseguenza; è inoltre fondamentale utilizzare un sistema probabilistico per stimare il comportamento dinamico della scena andando a monitorare quelle variabili che possono influenzare lo stato della scena stessa. Uno dei più diffusi "strumenti" utilizzati in casi come questo è il **Filtro di Kalman** che andremo ora ad analizzare più in dettaglio.

3.1 Filtro di Kalman

Il filtro di **Kalman** è un metodo matematico inventato da Rudolf E. Kalman; il cui scopo è quello di stimare il valore reale di un'insieme di variabili il cui valore cambia nel tempo e sopratutto è soggetto a "rumore" ed imprecisioni dovute ad errori di misurazione o semplicemente a variazioni casuali non prevedibili. Questo stesso insieme di variabili costituisce lo *stato del sistema* dinamico di cui il filtro cerca di prevedere le variazioni. Le stime dei valori delle variabili osservate sono calcolati facendo una predizione del valore futuro delle variabili stesse, facendo una stima dell'incertezza dello stesso e calcolando poi la media pesata del valore predetto e il valore misurato dal sistema che si sta osservando.

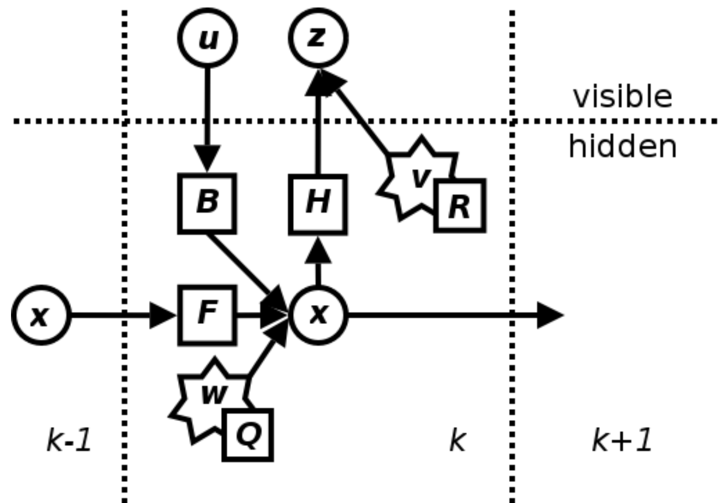


Figura 3.1: Funzionamento del filtro di Kalman

Il filtro di Kalman si basa su sistemi dinamici lineari discretizzati nel dominio del tempo e lo stato è costituito da un vettore di valori rappresentanti le variabili da "monitorare"; ad ogni "avanzamento" del tempo viene applicato l'operatore lineare derivato dalla descrizione matematica del modello

reale (i.e. equazione del moto) per generare/predirre il nuovo stato con un certo ammontare di rumore aggiunto allo stesso; succcessivamente a partire dallo stato reale ovvero l'ultima misurazione effettuata, viene generato lo stato osservato.

Per permettere al filtro di stimare i valori dello stato dinamico osservato è necessario adattare il modello al processo "incaricato" di variare le variabili caratterizzanti il sistema. Questo significa in termini pratici che vanno specificate delle matrici ognuna delle quali ha uno significato particolare nell'evoluzione del filtro stesso; le matrici in questione sono:

- **F:** è la matrice di transizione dello stato
- **H:** matrice di misurazione
- **Q:** matrice di covarianza del rumore del processo
- **R:** matrice di covarianza del rumore di misurazione
- **B:** matrice di controllo (può essere non utilizzata)

L'equazione che governa la variazione dallo stato *k-1* allo stato k è:

$$x_k = F_k x_{k-1} + B_k u_k + w_k$$

dove w_k è il rumore del processo modellato su una variabile normale a media nulla e covarianza Q_k: $w_k \sim N(0, Q_k)$. Ad ogni incremento temporale viene calcolato lo stato "osservato" utilizzando l'ultimo valore reale avuto in input e seguendo questa equazione:

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k$$

nella quale H_k ha il compito di mappare lo stato reale in quello osservato dal filtro e dove v_k rappresenta il rumore di misura modellato con una variabile normale a media nulla e covarianza R_k: $v_k \sim N(0, R_k)$.

Capitolo 4

Applicativo

L'applicativo sviluppato analizza una sequenza video con l'obiettivo di identificare gli oggetti presenti sulla scena e successivamente studiarne il moto relativo ed assoluto. Con moto "relativo" si intende il moto che l'oggetto fa in relazione al piano immagine ovvero quello rappresentato dalla differenza di posizione nel rettangolo video tra un frame e il successivo; con moto "assoluto" si intende invece il moto che l'oggetto fa rispetto alla scena reale ovvero rispetto al background. Il movimento degli oggetti viene poi rappresentato in output con dei vettori il cui modulo e direzione descrivono la stima approssimativa del moto dei singoli oggetti e della camera, rimanendo però comunque sul piano immagine senza andare a mappare i punti nello spazio 3D. Questo punto sarà discusso nell'ultimo capitolo.

Analizziamo ora come l'applicazione è strutturata.

4.1 La struttura dell'applicativo

Il funzionamento del programma è suddiviso in varie fasi principali(Figura 4.2):

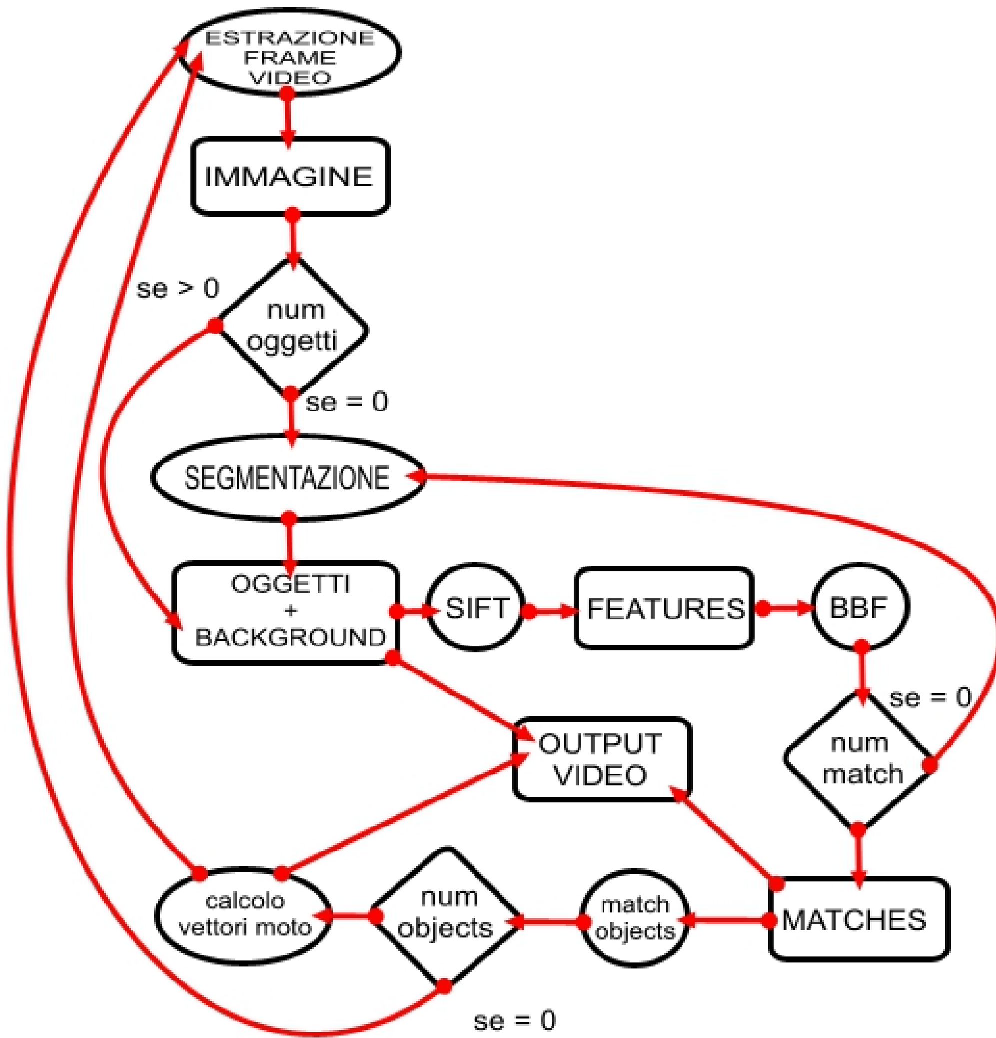


Figura 4.1: Schema del funzionamento dell'applicazione.

1. **Estrazione di un nuovo frame video:** viene estratto un nuovo frame dal video
2. **Individuazione degli oggetti:** in questa fase l'immagine viene suddivisa in un certo numero di regioni che corrispondono a zone i cui pixel hanno caratteristiche simili tra loro e molto diverse a quelle delle regioni adiacenti. Questa suddivisione si ottiene utilizzando un algoritmo di segmentazione d'immagine e ogni regione andrà a rappresentare un singolo oggetto; per identificare quale di queste regioni sia associata al background, si assume che sia quella con area maggiore.
3. **Estrazione delle feature:** vengono estratte le feature del frame corrente
4. **Ricerca delle corrispondenze:** i descrittori delle feature appena estratte vengono indicizzati con il kd-tree e successivamente viene effettuata la ricerca, con il metodo Best Bin First, delle corrispondenze tra il frame corrente e tutte le feature salvate nel buffer negli N frame precedenti. Successivamente viene aggiornato lo stato del filtro di Kalman di ogni singola feature nel buffer per cui sia stato trovato un match, le nuove feature vengono invece inserite e quelle già presenti per cui non è stato trovato un match per più di N frame consecutivi vengono eliminate.
5. **Calcolo e visualizzazione dei vettori di moto:** si calcolano i vettori di moto dei singoli oggetti e del background prendendo come modulo la media dei moduli di tutte le singole feature appartenenti all'oggetto e come direzione la media pesata degli angoli, usando come peso il modulo. Viene poi aggiornato lo stato dei filtri di Kalman ed infine visualizzati i risultati delle stime del movimento.

Figura 4.2: Fasi principali.

Lo schema di figura 4.1 illustra come l'applicazione si comporta nelle varie fasi. All'inizio il programma estrae le feature dal primo frame del video e le utilizza per inizializzare il buffer delle feature; viene poi applicato un ciclo di segmentazione dal cui risultato viene individuato il background e di conseguenza tutti gli oggetti di cui tener traccia. Per ogni oggetto viene inizializzato il filtro di Kalman e vengono salvate le referenze a tutte le feature associate ad esso. Terminata questa fase di inizializzazione viene estratto un nuovo frame e di questo calcolate le SIFT, si passa poi alla ricerca delle corrispondenze tra l'insieme di feature appena trovato e quelle contenute nel buffer che viene poi aggiornato come spiegato precedentemente. A questo punto dell'esecuzione si controlla se esistono oggetti che l'applicazione sta osservando e se sono state trovate corrispondenze con il frame corrente; se una o entrambe queste condizioni non sono vere viene ipotizzato un completo cambio di scena nel video e di conseguenza viene rieseguita tutta la fase di inizializzazione fatta all'inizio, così da trovare nuovi oggetti e corrispondenze con il frame successivo. L'ultima fase consiste nel calcolo dei vettori di moto per ogni oggetto e per la camera che vengono poi riportati sul video di output.

Le varie fasi possono essere sintetizzate brevemente come nell'algoritmo che segue:

```
firstFrame <- query_video_frame;
featureTrace <- extract_features(firstFrame);
segmented_img <- segmentation(firstFrame);
objectTrace <- extractObject(segmented_img);
do{
    img <- query_video_frame;
    features <- extract_features(img);
    nmatch <- findMatches(featureTrace,features);
    updateTrace(featureTrace);
    if(nmatch > 0){
        computeMotionVectors(objs);
        updateObjectTrace();
        draw_vectors(img);
    }
    else{
        segmented_img <- segmentation(img);
        objectTrace <- extractObject(segmented_img);
    }
    writeFrame(img,outputVideo);
}while(img exist)
clean_up_memory;
```

Andremo ora a studiare più in dettaglio le fasi salienti.

4.2 Estrazione delle feature

In questa fase vengono estratte le SIFT feature dell'immagine relativa al frame corrente(Figura 4.3). Successivamente viene creato un oggetto di tipo

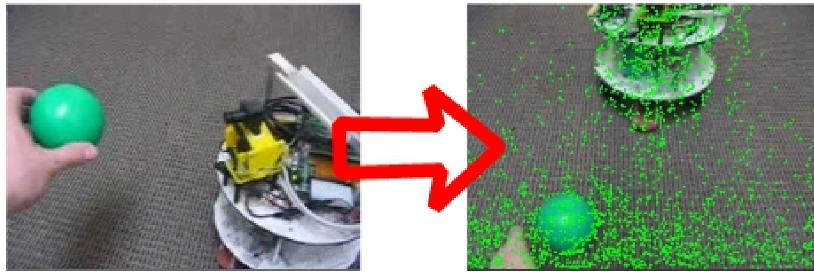


Figura 4.3: Fase 1: estrazione delle feature.

frameBuffer il cui utilizzo è semplicemente quello di mantenere per ogni frame il numero di feature trovate, il puntatore al vettore contenente le feature stesse, le dimensioni dell'immagine e il puntatore ai dati "grezzi" come array di float; si può dire che questa struttura dati funga da interfaccia tra i dati relativi all'immagine e le funzioni di estrazione delle feature e ricerca delle corrispondenze.

La funzione che interfaccia i dati immagine con la funzione che implementa l'algoritmo SIFT è *vector<feature> findFeatures(float * data, int width, int height)* , i parametri in ingresso sono *data*, ovvero il puntatore ai dati di basso livello dell'immagine, e le dimensioni della stessa, tutte informazioni contenute nella struttura framebuffer; in uscita viene invece ritornato un vettore di tipo "std::vector" di feature. All'interno della funzione(nel file support.cpp) è possibile configurare i vari parametri su cui l'algoritmo si basa per l'estrazione dei keypoint, ovvero tutti i valori e le soglie che caratterizzano il funzionamento della Scale Invariant Feature Transform; i principali sono riportati di seguito:

- *octaves* il numero di ottave (default: -1; il che significa che viene ricavato automaticamente dall'algoritmo)

- *levels* il numero di livelli per ottava (default: 3)

- *sigman* il valore nominale del livello di blurring dell'immagine (default: 0.5)

- *sigma0* il valore di base del livello di blurring dell'immagine (default: $1.6 * powf(2.0, 1.0/levels)$)

- *threshold* la soglia per l'eliminazione di keypoint a basso contrasto (default: $0.04/levels/2.0$; Lowe in [1] suggerisce un valore pari a 0.02)

- *edgeThreshold* la soglia per l'eliminazione di keypoint situati su bordi (default: 10.0)

I valori di default sono quelli che Lowe consiglia nel suo paper per ottenere il rapporto ottimale tra completezza dei risultati e velocità d'esecuzione.

Il primo passo è quello di calcolare lo *scale-space* della funzione DoG inizializzando un oggetto di tipo *Sift* chiamando il suo costruttore direttamente, una volta precalcolati i valori, si può passare alla ricerca degli estremi massimi e minimi che verranno in seguito filtrati in base alle soglie prefissate(*threshold, edgeThreshold*) utilizzando la funzione *Sift::detectKeypoints(...).* Ora per ogni keypoint trovato vanno calcolati orientazione *computeKeypointOrientations(...)* e descrittore(funzione *computeKeypointDescriptor(...)*) e, in caso di orientazioni multiple viene creato per ognuna di esse un nuovo keypoint con la stessa posizione ma angolo diverso. Infine il descrittore viene normalizzato all'unità. Passiamo ora alla fase successiva.

4.3 Ricerca delle corrispondenze

In questa fase si ricercano le corrispondenze (i “match”) tra il vettore delle feature appartenenti al frame corrente e il vettore delle feature precedenti (*trace*). La funzione per trovare i match tra due vettori di feature è *int findMatches (vector <feature> & feats1, vector <feature>& feats2, int currentFrame)*, la quale si appoggia alle implementazioni di Hess[3] per l’utilizzo del kd-tree e della ricerca Best-Bin-First; prima viene costruito l’albero sull’insieme dei descrittori delle feature associate ai frame precedenti, poi per ogni feature del frame corrente si cercano i “nearest nieghbor”. Quando un match è trovato viene salvato il puntatore nel campo *fwd_match* della feature. La funzione restituisce il numero di match trovati. *currentFrame* è l’indice del frame corrente.

Inizialmente la ricerca delle corrispondenze era effettuata solo tra coppie di frame adiacenti ma così facendo non si ottenevano risultati soddisfacienti nel tracciamento delle feature per più di due frame, con la conseguenza che la stima di intensità e direzione del moto risultava molto instabile. Per ovviare a questo problema si utilizza un vettore di feature che funge da “buffer” temporaneo in cui vengono mantenute le feature per un numero N di frame consecutivi, questo valore è configurabile a piacere ma è intrinseco che ad un valore maggiore corrisponderà un numero di feature maggiore, pari a quelle degli N frame precedenti a meno dei match trovati di volta in volta per cui sono solo state aggiornate alcune feature senza aggiungerne di nuove. Crescendo la grandezza del buffer aumenta ovviamente il tempo necessario per l’indicizzazione e la ricerca, il valore ottimale è stato trovato sperimentalmente ed è pari a 4. Il funzionamento di questo buffer, il vettore *vector<feature> trace* nel codice, si basa sull’uso di una struttura contenuta in ogni feature, il campo *ft*, di tipo *featureTrace* nella quale vengono mantenute le informa-

zioni relative al moto (filtro di Kalman) e all'età della feature definita come il numero di frame consecutivi in cui non sono state trovate corrispondenze con la feature stessa.

Il mantenimento del buffer è effettuato fondamentalmente con due cicli successivi, uno per l'inserimento delle feature trovate nell'ultimo frame ed uno per l'aggiornamento dello stato di tutte le feature della traccia che non sono state modificate nel ciclo precedente.

La prima fase consiste nel ciclare su tutte le feature estratte nel frame corrente e per ognuna di esse controllare se ha o meno una corrispondenza con una feature contenuta nella traccia, di conseguenza:

- **se la feature ha una corrispondenza:** la feature contenuta nella traccia viene sostituita con la nuova e lo stato di kalman nel campo *ft* viene aggiornato tramite la funzione *updateKalman(...)* utilizzando come nuova misura la posizione della nuova feature. Successivamente vengono salvate le coordinate delle due feature formanti il match, in un oggetto di tipo *movector* inserito poi in *trj*, un vettore "std::vector<movector>" che funge da contenitore di tutte le corrispondenze trovate con il frame corrente e che andremo ad analizzare più tardi.

- **altrimenti:** se la feature non ha trovato nessuna corrispondenza significa che è con buona probabilità una nuova feature e viene quindi inizializzato il suo stato di kalman e aggiunta al buffer.

La struttura dati *movector* non fa altro che salvare le coordinate dei punti appertenenti alle due feature che formano il match come *floatPoint* ovvero una semplice struttura contenente le due coordinate come float.

La seconda fase serve invece per mantenere "pulito" e aggiornato il buffer rispetto allo stato corrente; è costituita da un unico ciclo principale sugli elementi del buffer, per ogni feature viene fatto un controllo sull'"età" della feature stessa, calcolata come differenza tra l'indice dell'ultimo frame in cui ne è stata trovata una corrispondenza(*lastFrame*) e l'indice del frame in cui è stata estratta(*firstFrame*); se l'età è maggiore di una certa soglia (*TRACKING_BUFFER_LENGHT* = *N* definita in support.h) allora viene eliminata dal buffer. In caso contrario ne viene aggiornato lo stato di kalman e inserite le coordinate nella struttura *movector* salvata poi in *trj*.

Kalman, inizializzazione e aggiornamento Il filtro di Kalman viene applicato singolarmente ad ogni feature per stimarne la posizione su frame successivi; al momento dell'inserimento di una nuova feature, viene inizializzata la struttura *CvKalman* utilizzando la funzione *initKalman*. Per calcolare la stima della posizione di un punto su di un piano, in questo caso quello dell'immagine, sono state utilizzate le seguenti matrici:

- **Matrice di transizione *F***

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **Matrice di misurazione *H***

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

- **Matrice di covarianza del rumore del processo *Q***

$$\begin{bmatrix} 100 & 0 & 1 & 0 \\ 0 & 100 & 0 & 1 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 100 \end{bmatrix}$$

- **Matrice di covarianza del rumore di misurazione *R***

$$\begin{bmatrix} 0.2845 & 0.0045 \\ 0.0045 & 0.0455 \end{bmatrix}$$

- **Matrice di controllo *B***

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

4.4 Rilevamento degli oggetti

Questa fase è molto importante perchè definisce quali saranno gli oggetti di cui tracciare il movimento. La soluzione adottata consiste nell'utilizzare un algoritmo di segmentazione d'immagine in modo da ottenere una suddivisione tale da poter distinguere con buona probabilità quale segmento rappresenti il background, di conseguenza tutte le altre parti verranno trattate come oggetti.

La segmentazione è un problema complesso e ottenere i risultati desiderati non è mai facile, soprattutto in casi come questo in cui le immagini possono essere molto diverse e l'automatismo dell'applicazione è fondamentale. È comunque possibile ottenere con una certa probabilità risultati che rappresentano una buona approssimazione della soluzione ottimale.

Andiamo ora ad analizzare più approfonditamente questa fase.

4.4.1 La segmentazione d'immagine

Il processo di segmentazione di un'immagine consiste nel partizionare un'immagine in più regioni confinanti chiamate anche "segmenti", più precisamente si può definire questo processo come l'assegnazione di un'etichetta (rappresentata solitamente da un colore o un'intensità) ad ogni pixel in modo tale che ogni insieme di pixel con uguale etichetta abbia in comune una certa caratteristica "visiva".

L'obiettivo principale del processo di segmentazione è quello di "semplificare" l'analisi e l'utilizzo di quelle che sono le caratteristiche più significative di un'immagine; difatti la segmentazione viene spesso utilizzata per i processi di edge detection o, come in questo caso, di object detection.

Il risultato che si ottiene da questo processo è costituito da un insieme di regioni che coprono l'intera immagine, queste hanno caratteristiche visive, come il colore o l'intensità, simili nell'immagine sorgente ma che differiscono molto tra regioni confinanti.

Il lavoro di Felzenszwalb [8] implementa un algoritmo di segmentazione che si basa sull'utilizzo dei grafi e delle metodologie per il loro partizionamento; il risultato del processo dipende da una serie di parametri che possono essere configurati a priori:

- **sigma** è il livello di blurring che viene applicato all'immagine prima di cominciare il processo di segmentazione e più è alto minore è l'influenza che i bordi hanno sul risultato; il valore di default è 0.5

- **k** è proporzionale alla sensibilità con cui l'algoritmo divide l'immagine e di conseguenza al numero di regioni; maggiore è il suo valore, minore è la sensibilità.

- **min** questo parametro rappresenta una soglia alla grandezza minima delle regioni.

Nel nostro caso, i valori utilizzati sono stati derivati sperimentalmente e sono $k = 2000$, $sigma = 1$ e $min = 400$.

Nelle figure 4.4 sono illustrati alcuni esempi di segmentazione in cui si nota un buon “isolamento” degli oggetti e sopratutto una buona uniformità del background. Nel caso della partita di calcio si può notare che il campo è suddiviso in due zone, l’applicazione prende come background quella con area maggiore il che, nonostante una parte del background non sia considerata tale, rende il calcolo della stima del moto della camera comunque veritiera perchè basata su tutte le feature all’interno dell’area maggiore.

4.4.2 Identificazione degli oggetti e tracciamento della posizione

Una volta finita la fase di segmentazione, il risultato di tale processo viene utilizzato per identificare le aree che dovrebbero quindi essere associate agli oggetti di cui si vuole analizzare il moto. Per distinguere il background dal resto si ricerca quale tra le differenti regioni è quella con l’area maggiore andando semplicemente a “contare” quanti pixel ci sono per ogni differente colore e prendendo quello corrispondente al massimo.

Una volta inizializzata la struttura dati, ogni *object* viene salvato in un vettore *vector<object> objsTrace* che funge da contenitore per tutti gli oggetti nella scena. Quando l’applicazione passa al frame successivo, ne estrae le feature, ne cerca i match con le feature precedenti ed esegue un controllo su due variabili:

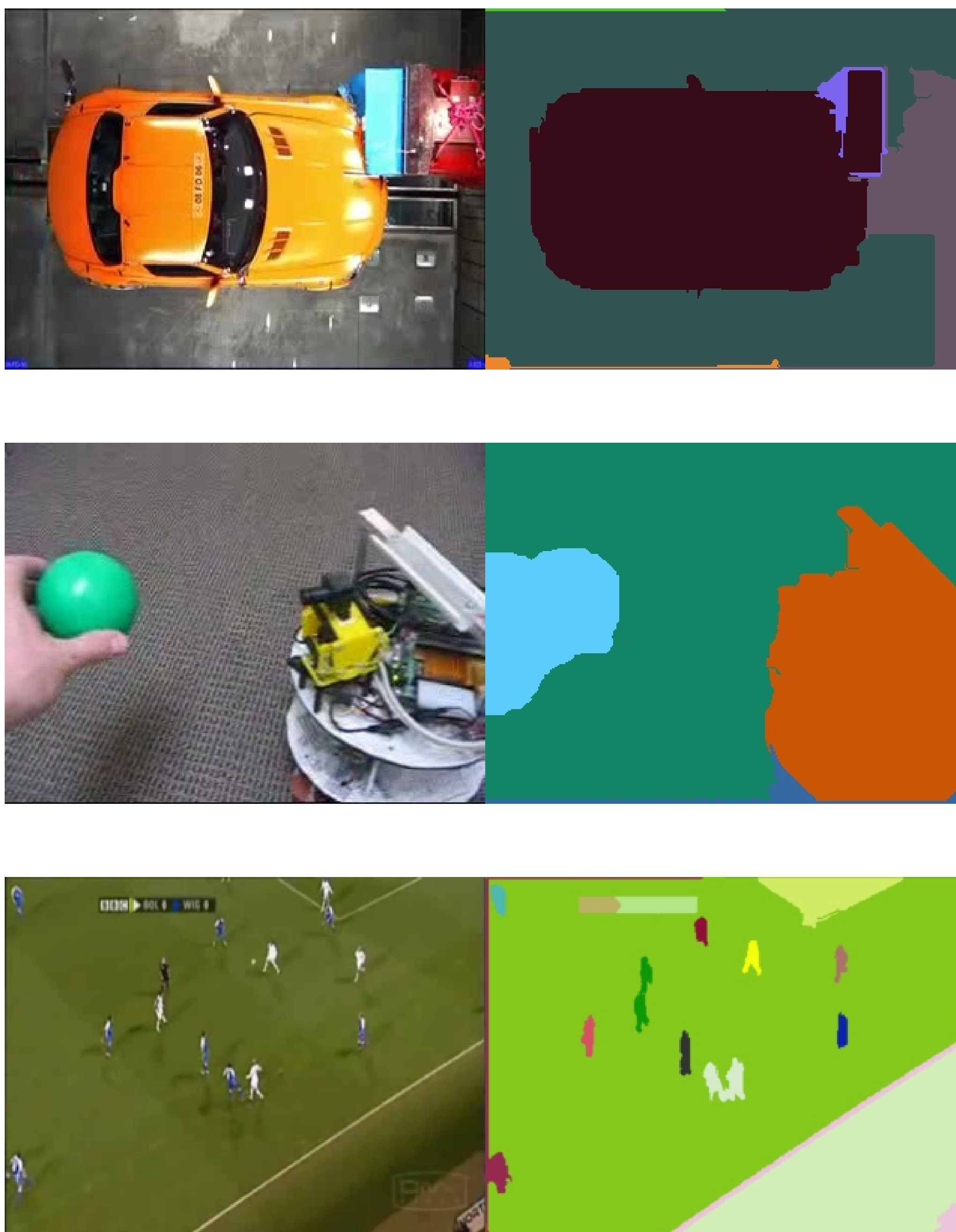


Figura 4.4: Esempio di segmentazione con $k = 2000$, $sigma = 1$ e $min = 400$

- il numero di match: se questo è minore di 10 (non si usa 0 per evitare possibili corrispondenze casuali) il nuovo frame viene considerato l'inizio di una nuova scena

- il numero di oggetti in *objsTrace*: se è pari a 0 significa che nessun oggetto che si stava osservando è più all'interno della scena corrente

In entrambi i casi di cui sopra viene riapplicata automaticamente la segmentazione d'immagine e viene reinizializzato il vettore *objsTrace*.

Nel passaggio da un frame al successivo, ogni *object* deve essere aggiornato con lo stato relativo al frame corrente; per trovare la nuova posizione dell'oggetto vengono cercate tutte le corrispondenze tra le feature associate all'oggetto nel frame precedente e tutte quelle estratte nel frame corrente, sulla base di queste corrispondenze vengono trovati i punti associati all'oggetto nel nuovo frame. Utilizzando questo insieme di feature viene innanzitutto calcolato il convex hull che definisce l'area in cui è contenuto l'oggetto, successivamente vengono calcolati anche il modulo e l'angolo del vettore di moto e viene infine aggiornato lo stato del filtro di Kalman con i nuovi valori. Se durante questa fase non vengono trovate corrispondenze con le feature di un oggetto, questo viene eliminato.

4.5 Calcolo e visualizzazione dei vettori di moto

In questa sezione vogliamo dare uno sguardo a come l'applicativo calcola la stima del vettore di moto di ogni singolo oggetto e della telecamera. Il concetto è di per sé molto semplice: per ogni oggetto, si indivuiduano tutte le feature, all'interno dell'area identificativa dell'oggetto stesso, di cui si è

trovata una corrispondenza con il frame precedente (gli elementi *movector* in *trj*) e dal moto di questi si calcola una approssimazione del moto globale dell'oggetto. Questa approssimazione è derivata dalla sottrazione del moto della telecamera al moto dell'oggetto relativo alla posizione nel frame video. Per quanto riguarda invece il moto della telecamera, lo si ricava considerando l'opposto del movimento del background ovvero quello di tutte le feature che non associate ad alcun oggetto e che quindi non sono all'interno di nessun contorno.

Le informazioni che ci interessano in questa fase sono la direzione e il modulo del vettore di moto che caratterizzano ogni oggetto nella scena e la scena stessa. Possiamo considerare il modulo come una rappresentazione molto approssimata della velocità, o meglio diciamo che è il rapporto tra spazio (quanto si è spostato l'oggetto) e tempo (un frame) il che non è altro che la velocità calcolata utilizzando la dimensione temporale discretizzata. Nel caso che il match sia stato trovato tra due feature a N frame di distanza, il modulo del vettore relativo a quel punto è pari alla distanza divisa per N, con N che rappresenta la sua "età". Per quanto riguarda questa stima del moto reale dell'oggetto è doveroso sottolineare che nello stato attuale l'applicazione calcola ogni cosa su coordinate 2D; questo implica che per ottenere una stima più realistica del moto rispetto alla scena si dovrebbe introdurre una fase preventiva di calibrazione della camera e calcolo del suo modello e successivamente una fase di mappatura delle coordinate dei punti del piano immagine in coordinate dello spazio 3D.

La direzione del vettore è espressa in gradi secondo lo schema di figura 4.5.

Per trovare la direzione globale di ogni oggetto si calcola la media pesata degli angoli di tutti i vettori di movimento associati ad ogni feature dell'og-

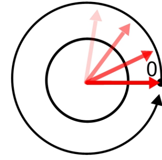


Figura 4.5: Angolo.

getto, utilizzando come peso il modulo di ogni stesso vettore. Per quanto riguarda invece il modulo, viene preso semplicemente il valore medio. Si sta già pensando, in un prossimo sviluppo dell'applicativo, di introdurre un metodo più efficiente per la determinazione di questi due parametri in modo da rendere più robusta la stima del valore reale.

Per la visualizzazione, se il vettore ha come origine O(x,y) il punto finale F(xf,yf) viene calcolato come $xf = modulo * cos(angolo)$ e $yf = modulo * sin(angolo)$.

In output il programma visualizza ad ogni nuovo frame due vettori per ogni oggetto posizionandoli nel centro dello stesso: il vettore di colore rosso indica il moto dell'oggetto rispetto all'immagine ovvero come la feature si è spostata da un frame al successivo, mentre quello di colore bianco rappresenta la stima del movimento dell'oggetto rispetto alla scena circostante ovvero rispetto al background. Questa stima è ricavata dalla somma vettoriale del vettore di movimento rispetto all'immagine e del vettore di movimento della camera. La stima del vettore di moto della camera è mostrata nell'angolo in basso a destra con una linea di colore viola. Ad ogni frame il programma mantiene la traccia del percorso di ogni oggetto, aggiornandola di volta in volta e la riporta in giallo sul frame di output (Figura 4.6).

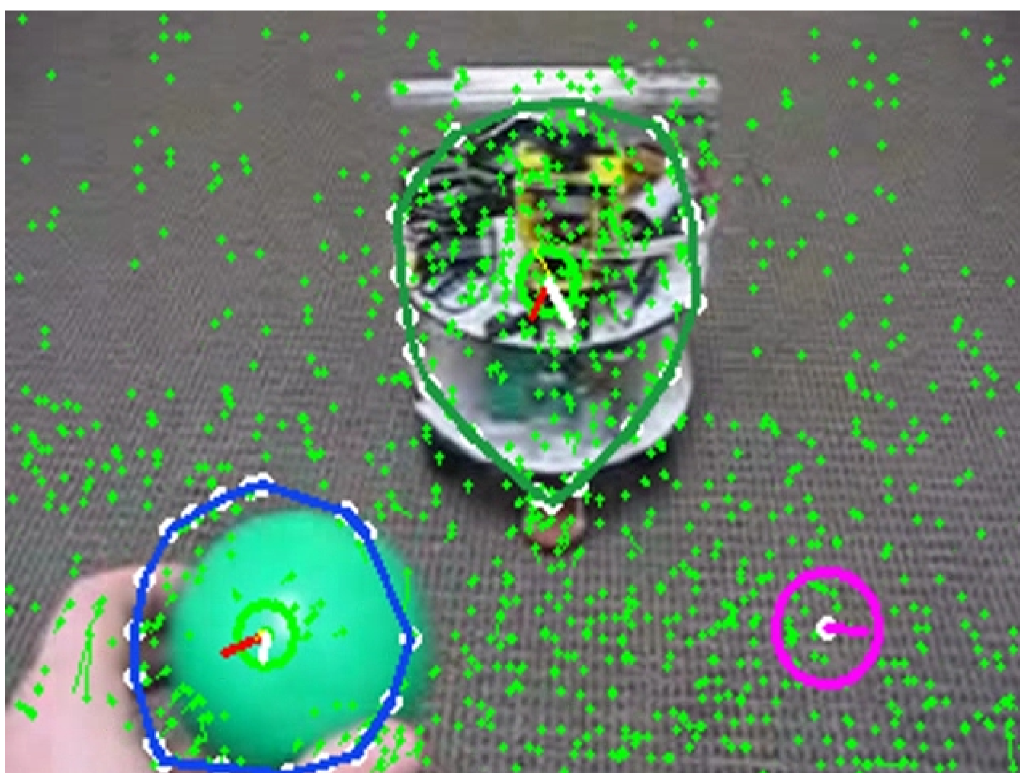


Figura 4.6: Output, le linee gialle rappresentano gli spostamenti.

Capitolo 5

Risultati sperimentali

L'output del programma consiste in una sequenza video in cui sono rappresentati i risultati del calcolo della stima dei vettori di moto. Prendiamo ad esempio la figura 5.1, sono visualizzati:

- **Le feature:** ogni singolo punto verde rappresenta la posizione di una feature del frame corrente

- **I vettori di moto delle feature:** per ogni feature visualizzata viene riportato il vettore di moto rappresentato da una piccola linea verde di lunghezza proporzionale al modulo del vettore

- **I contorni degli oggetti:** ogni oggetto di cui si sta tracciando la posizione e il moto è identificato da un insieme di feature, sulla base di queste viene calcolato il convex hull che viene poi disegnato sull'immagine e che rappresenta un'approssimazione del contorno dell'oggetto.

- **Il vettore di moto di ogni oggetto:** al centro di ogni oggetto sono riportati due vettori di moto: uno rosso che rappresenta il moto rispetto all'immagine ovvero quello calcolato sulla media del modulo del vettore

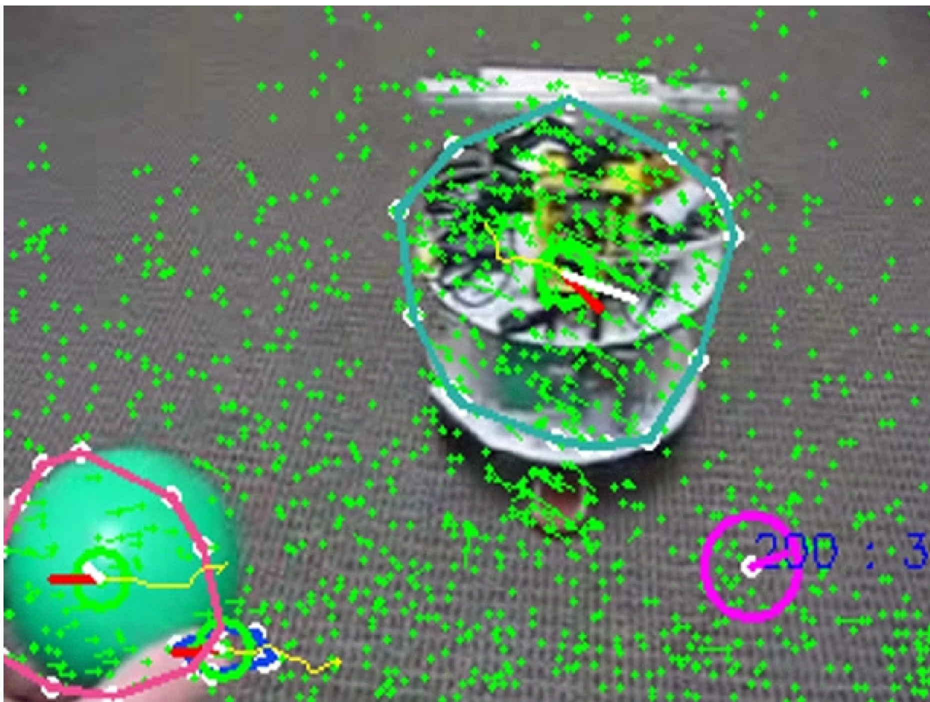


Figura 5.1: Un frame di una sequenza di output.

di ogni feature; uno bianco che rappresenta la stima approssimativa del moto rispetto alla scena reale e che viene calcolato come la somma del movimento di camera ed oggetto.

- **Il vettore di moto della camera:** è raffigurato nell'angolo in basso a destra con il colore viola e rappresenta la direzione in cui la camera si sta muovendo. Anche in questo caso la lunghezza della linea è proporzionale al modulo del vettore.

- **La traccia degli spostamenti degli oggetti:** per ogni oggetto viene mantenuta traccia della posizione dei centri calcolati di volta in volta e ogni punto viene disegnato sul frame di output andando a rappresentare il "percorso" fatto dall'oggetto sul piano immagine.

L'applicativo è stato testato su diverse sequenze video, in alcune si sono ottenuti buoni risultati mentre in altre si sono riscontrati alcuni problemi. Vediamo qualche esempio. Le immagini seguenti (Figura 5.2)sono relative alla stessa sequenza video di output e rappresentano i frame nell'ordine temporale dall'alto al basso:

Nella sequenza di sinistra si può notare che inizialmente l'oggetto in alto viene tracciato correttamente, man mano che la scena evolve, il robot si avvicina alla camera comportando a livello di immagine, un ingrandimento dell'oggetto stesso. L'applicazione mantiene agganciato l'oggetto e continua a tracciarne posizione e moto ma il problema consiste nel fatto che la grandezza dell'area in cui vengono cercate le corrispondenze con l'oggetto rimane la stessa dell'inizio mentre l'oggetto ha subito un ingrandimento.

Nella sequenza di destra viene invece mostrata un'altra sequenza in cui l'auto si sposta inizialmente a destra e poi facendo perno sulla zona azzurra ruota verso il basso. In questa sequenza sono stati riscontrati due problemi

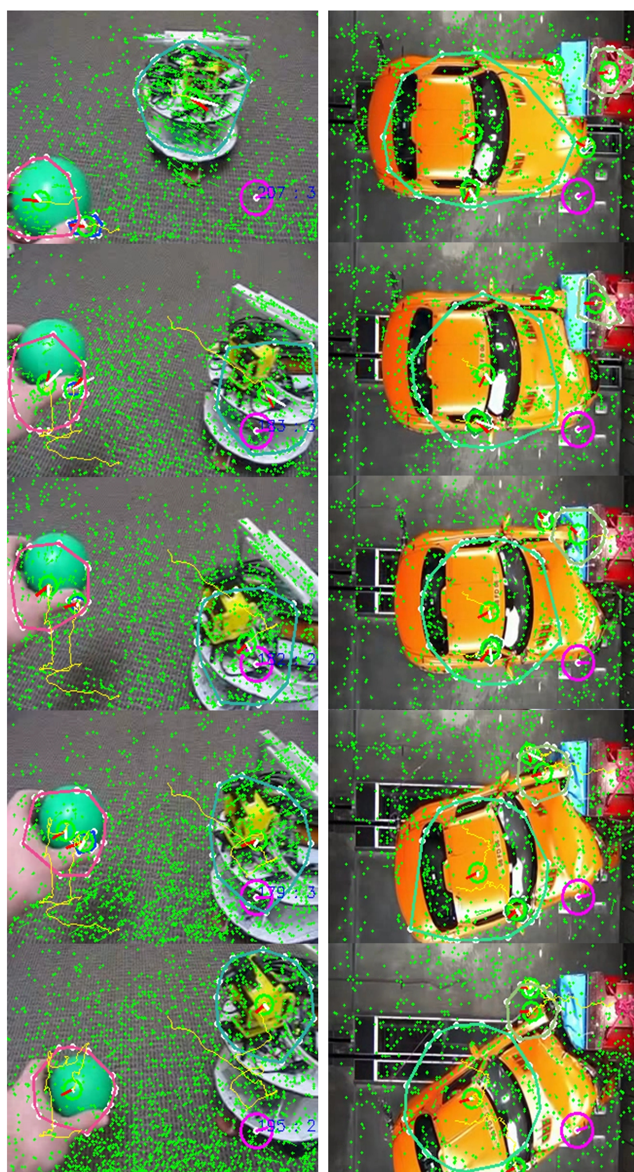


Figura 5.2: Alcuni risultati su sequenze video diverse.

nonostante l'applicazione non perda comunque la posizione dell'oggetto: la prima problematica è data dal rimpicciolimento dell'area dell'oggetto il che implica che per un certo numero di feature è stato perso il match andando così a perdere informazioni utili per il tracciamento nei frame successivi; un secondo problema è dato dal fatto che quando l'auto comincia ad uscire dalla scena, cominciano ad essere considerate come feature dell'oggetto alcune feature del background.

Nella maggior parte dei casi le applicazioni tipiche della computer vision sono applicazioni che vanno utilizzate in realtime, questo implica ovviamente che le funzioni implementate siano efficienti e che la mole di dati da elaborare sia limitata. Nello stato attuale questa implementazione richiede un tempo abbastanza lungo per l'elaborazione di ogni frame, per capire come il carico computazionale è distribuito tra le varie fasi l'applicazione è stata analizzata con un "profiler". Un "profiler", nel nostro caso *gprof* [7], è un programma che genera appunto un profilo del costo computazionale di un'applicazione, andando ad analizzare quante volte ogni singola funzione viene chiamata e quanto tempo, in termini di utilizzo della CPU, questa occupa nel corso dell'esecuzione.

I risultati ottenuti dal profiling dell'applicativo evidenziano chiaramente quanto la velocità di esecuzione sia influenzata dall'utilizzo delle SIFT, difatti le ripartizioni del tempo di esecuzione sono:

Ricerca delle corrispondenze tra feature: questa fase occupa in media circa l'*11%* del tempo totale e più in dettaglio è interessante notare che questa percentuale è suddivisa in 9% per la ricerca Best Bin First dei "nearest negihbor" all'interno del kd-tree e 2% per la costruzione dell'albero per l'indicizzazione delle feature.

Estrazione delle feature: questa fase è in assoluto la più intensiva e da sola è risultata utilizzare circa l'*81%* del totale tempo d'esecuzione. Questa percentuale è suddivisa come segue tra i passi che compongono l'estrazione delle feature:

Inizializzazione dello scale-space: il calcolo dell'operazione di convoluzione nella funzione di scale-space applicata alla *Difference of Gaussian* risulta occupare in totale circa il 42%

Calcolo dei descrittori delle feature: questa fase risulta occupare circa il 30% del totale

Calcolo delle orientazioni dei keypoint: circa il 7%

Ricerca degli estremi della funzione scale space: "solo" il 3%

Risulta quindi che le operazioni di estrazione e ricerca delle feature occupino insieme più del 92% percento del totale tempo di esecuzione del programma. Il rimanente 8% è ripartito tra tutte le altre fasi quali la segmentazione, l'aggiornamento dei filtri di Kalman, il calcolo dei vettori di moto e così via.

I risultati del profiling sono schematizzati nelle figure 5.3 e 5.4.

Per concludere riportiamo alcuni frame di video su cui l'applicativo è stato testato.

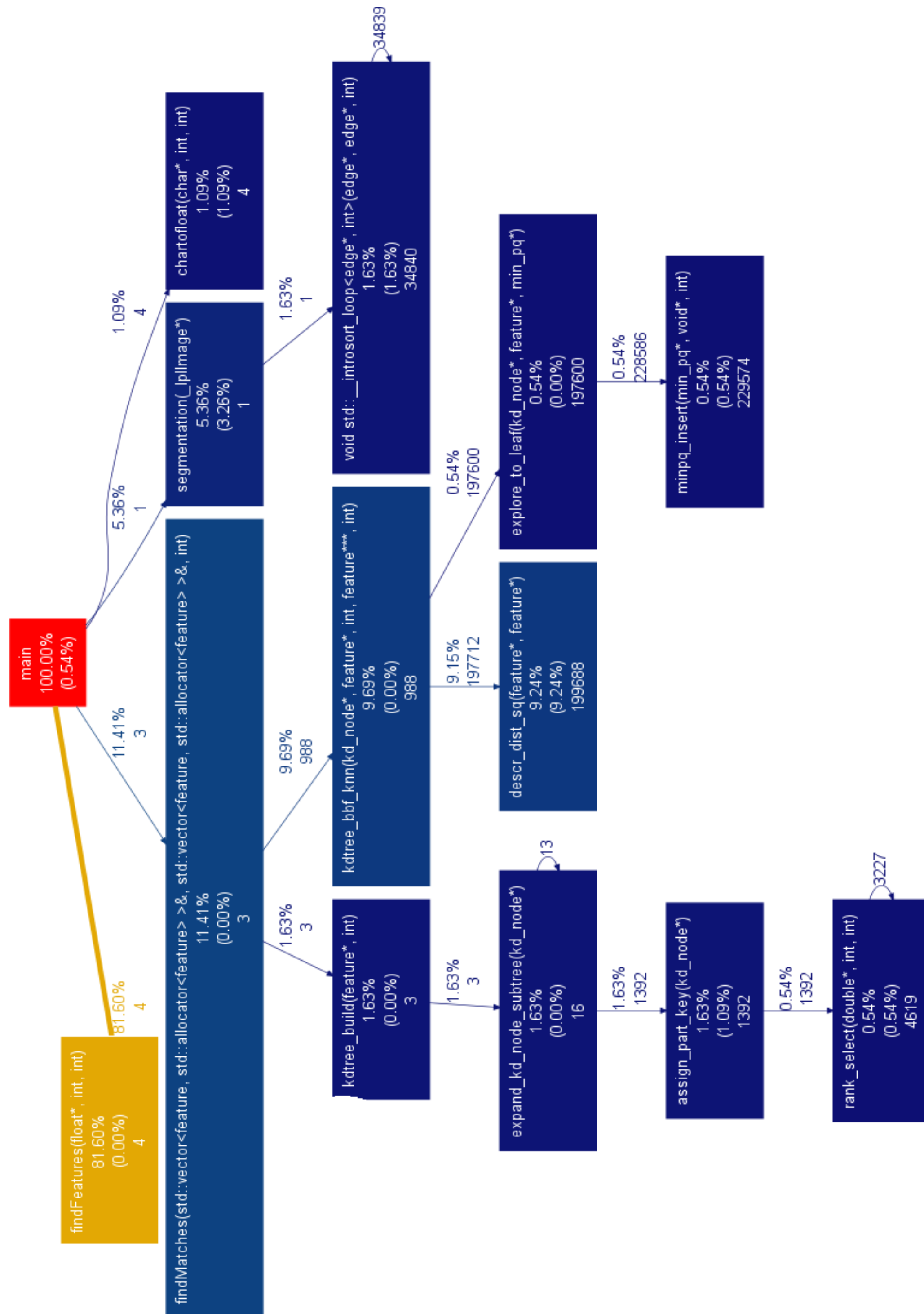


Figura 5.3: Risultato del profiling: ricerca delle corrispondenze e altre operazioni.

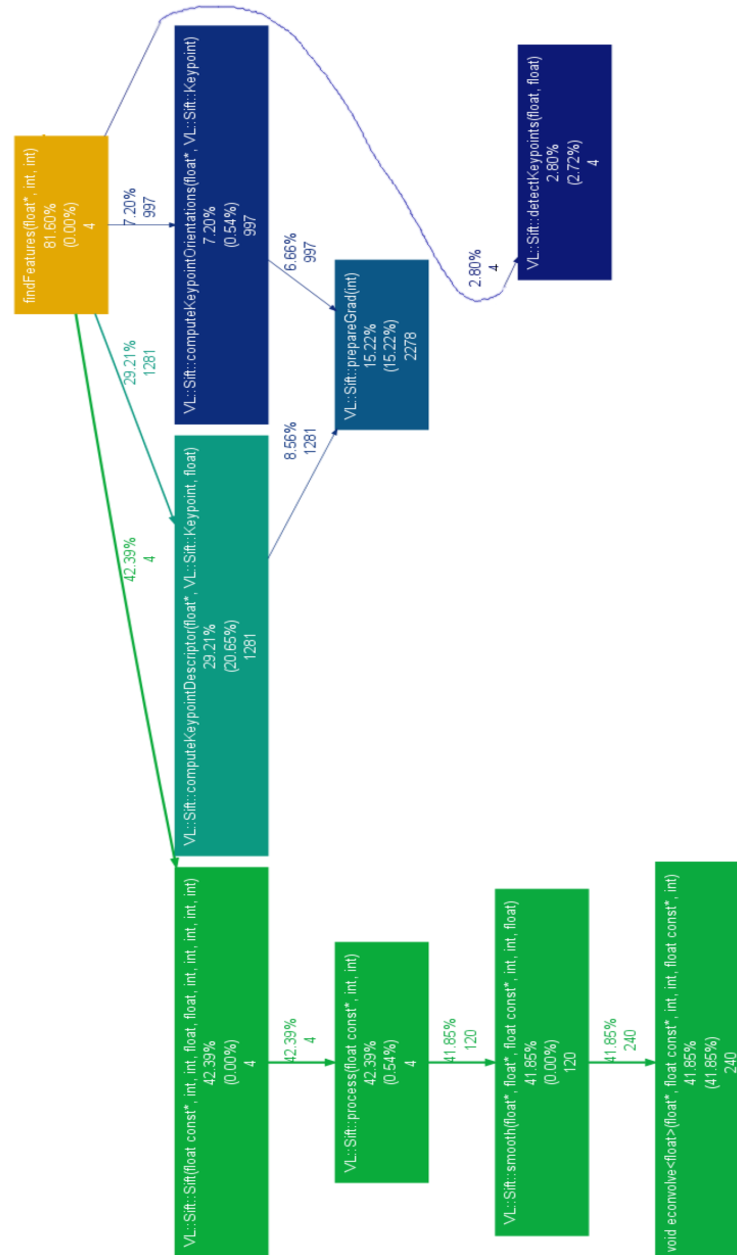
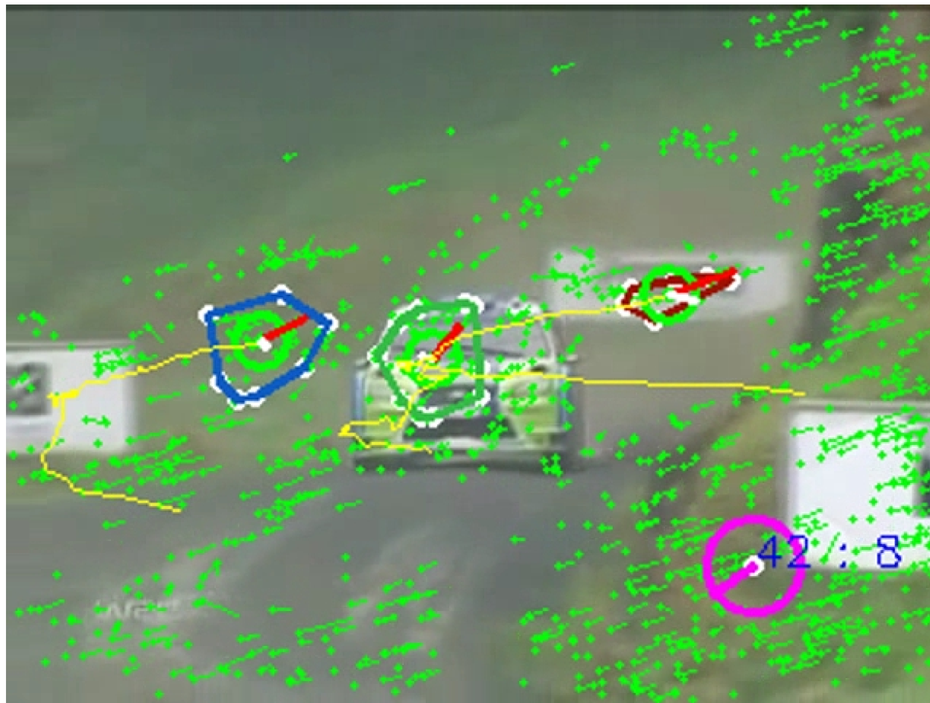


Figura 5.4: Risultato del profiling: estrazione delle feature.



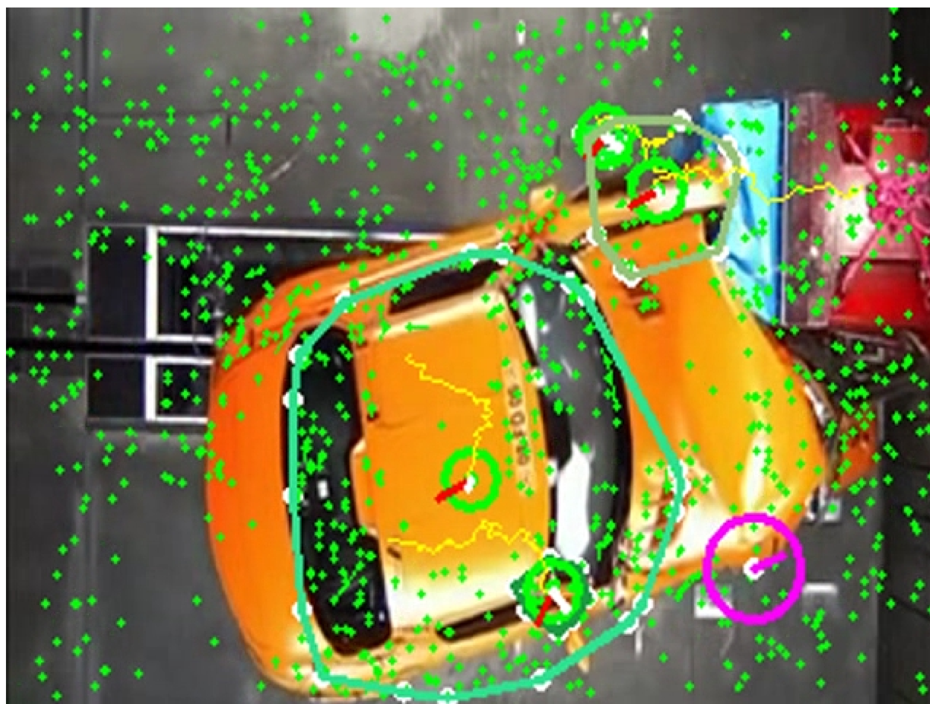
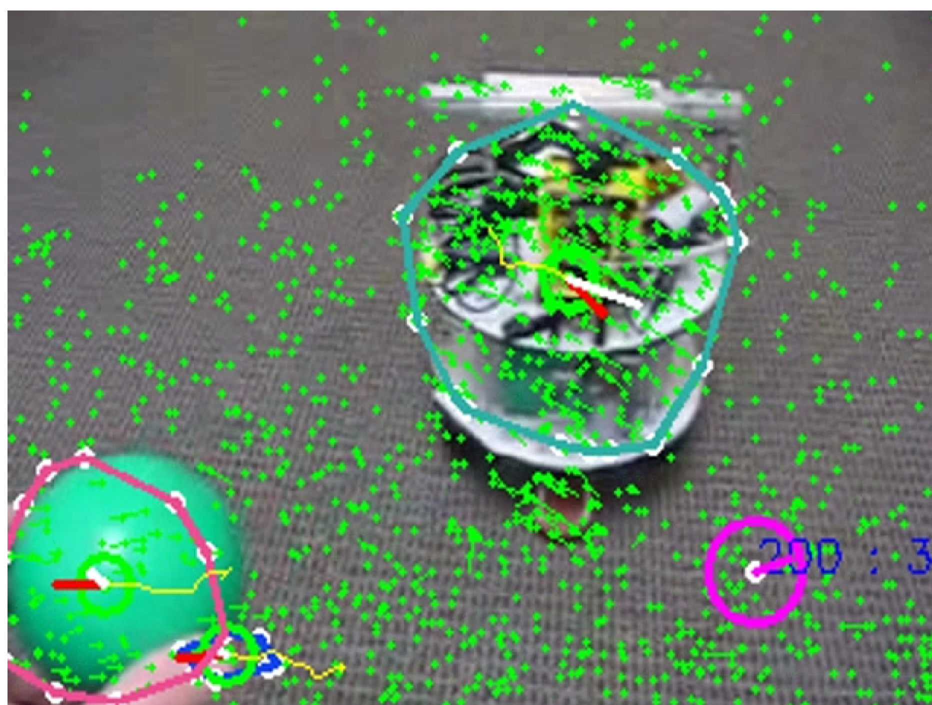


Figura 5.5: Alcuni risultati su sequenze video diverse.



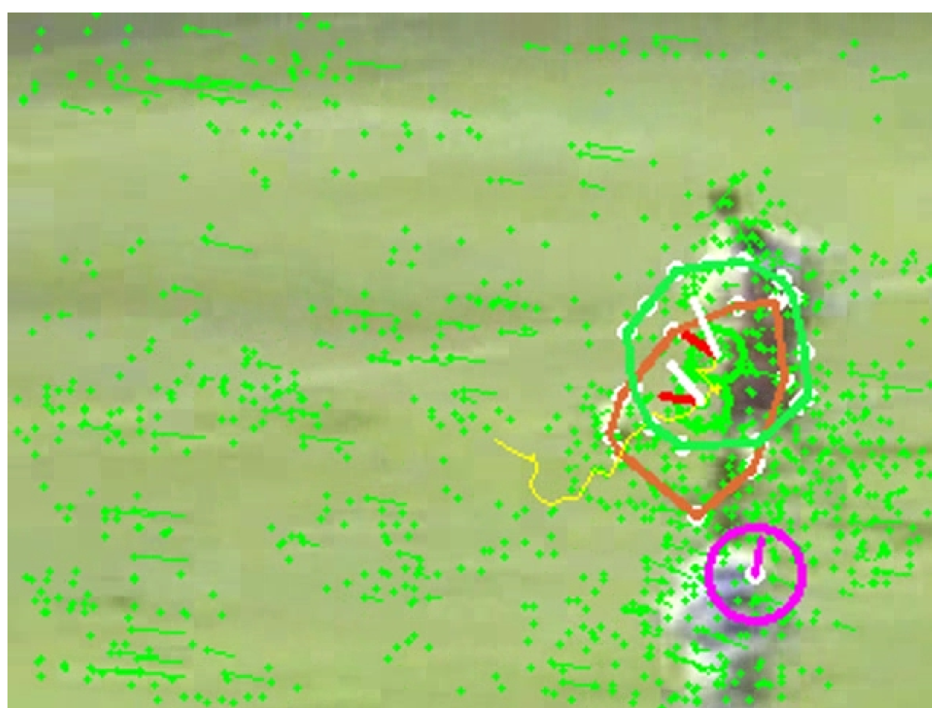


Figura 5.6: Alcuni risultati su sequenze video diverse.

Capitolo 6

Conclusioni e sviluppi futuri

Durante lo sviluppo di questa applicazione, l'idea originaria si è evoluta sotto diversi aspetti. Inizialmente si è cercato di utilizzare un algoritmo di clustering per separare i gruppi di feature degli oggetti dal background. L'approccio consisteva nel calcolare tutte le feature del frame corrente, cercarne le corrispondenze con quelli precedenti e calcolarne i vettori di movimento; una volta calcolati questi le problematiche erano principalmente due:

- capire cosa si stesse muovendo nella scena, oggetti e/o la telecamera e cosa fosse statico;
- individuare quali feature corrispondano a cosa, ovvero identificare gli oggetti e distinguerli dalla scena circostante.

Per individuare per quali feature fosse stato rilevato un movimento significativo, associabile con buona probabilità ad un movimento reale e non ad una semplice imprecisione di rilevamento, è stato utilizzato un algoritmo di clustering, il *kmeans*, ponendo come dati da processare le coordinate (x, y) del vettore di moto di ogni feature. Il risultato della clusterizzazione ottenuto cercando due insiemi ($k = 2$) era accettabile ma spesso poco preciso

con insiemi di feature poco stabili da cui risultava difficile estrarre insiemi ben definiti. Questo problema era probabilmente dovuto al fatto che la ricerca delle corrispondenze restituisce risultati che non sono stabili, ovvero che spesso una corrispondenza viene trovata tra due frame ma non nella coppia successiva facendo così cambiare anche di molto l'output di kmeans.

Le problematiche relative all'utilizzo di un algoritmo di clustering sono diverse, in primis la difficoltà di avere risultati "stabili" nel tempo e quindi cluster uniformi e sopratutto variazioni uniformi degli stessi, inoltre le feature estratte non hanno una ripetibilità elevata e questo porta ad avere insiemi di punti che variano, seppur di poco a livello globale, ma molto spesso localmente, portando così a variazioni nella clusterizzazione non prevedibili; Con "ripetibilità" si intende la probabilità di trovare corrispondenze tra feature che si susseguano tra più di due frame successivi, questo certo accade ma rispetto al numero totale di feature trovate, le feature più "longeve" non rappresentano una percentuale abbastanza elevata da rendere stabile la clusterizzazione.

Una possibile soluzione sarebbe da ricercare in algoritmi di clustering più complessi come ad esempio il *Quality Threshold Clustering* o lo *Spectral Clustering* che permettono entrambi di non specificare a priori il numero di cluster da ricercare ed inoltre consentono di configurare l'algoritmo in modo da limitare la grandezza geometrica dei cluster. Durante lo sviluppo è stato provato il *QTclustering*, utilizzando l'implementazione di in matlab, di cui è stato fatto il porting in C++, ma l'algoritmo non è efficiente e difatti già con un centinaio di punti i tempi di esecuzione crescevano esponenzialmente rendendone l'uso non idoneo per il nostro scopo. In un successivo sviluppo dell'applicativo sarebbe interessante provare ad implementare una versione più efficiente di un algoritmo di clustering (QTC o SC) e analizzare i risultati.

Ponendo come ipotesi di avere una fase di clusterizzazione efficiente, i casi possibili sono tre: la scena è completamente statica e si muove solo la camera, la camera è ferma ma alcuni oggetti si muovono, entrambe camera e oggetti si muovono. Nel primo caso le feature dovrebbero, a meno di errori di rilevamento, muoversi tutte nello stesso modo e avremo quindi un solo cluster il cui movimento rappresenta l'esatto opposto del movimento della camera; in pratica non succede e si trovano sempre corrispondenze non consistenti con la maggior parte delle altre feature ma si potrebbe pensare di prendere per buono il cluster con maggior numero di punti.

Nel secondo caso si dovrebbero ottenere due gruppi di feature distinti, uno appartenente a tutti gli oggetti e uno appartenente alla scena circostante; con la certezza che la camera è ferma si avrà quindi un cluster di feature con movimento nullo o quasi nullo e un cluster contenente tutte le feature in movimento. Il problema sorge in presenza di oggetti che si muovono lentamente, infatti può capitare che le feature dell'oggetto in questione vengano identificate erroneamente come background.

Nel terzo caso si avranno un numero variabile di cluster, teoricamente dovrebbero venire creati gruppi di feature con direzione e intensità del moto consistenti. Il primo problema da affrontare è quello di identificare l'insieme di feature associato alla scena, si può ipotizzare che il background sia il cluster con il maggior numero di punti ma anche questo non è sempre vero, in particolare se la scena è particolarmente omogenea (ad esempio un muro bianco), la SIFT troverebbe probabilmente un numero minore di feature nel background rispetto a quelle sugli oggetti. Supponendo di aver identificato il giusto cluster per il background rimane ora da distinguere i diversi oggetti in movimento, per fare questo si dovrebbe eseguire una nuova fase di clusterizzazione, stavolta però non più sulle coordinate del vettore di moto ma

sulla posizione delle stesse. Anche in questa situazione ottenere cluster ben distinti non è facile.

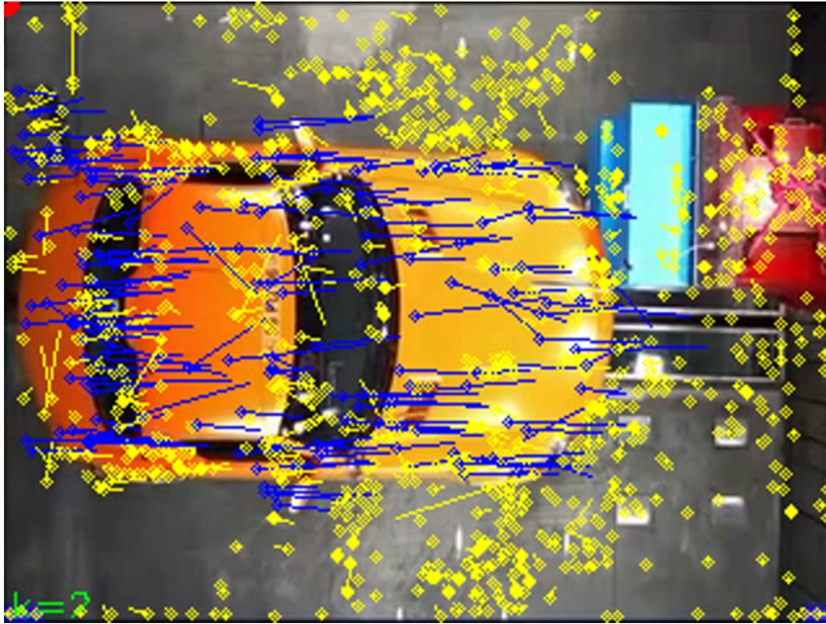


Figura 6.1: Prove di clustering.

Per migliorare questi aspetti si potrebbe provare a utilizzare una differente tipologia di feature come ad esempio le **SURF - Speeded Up Robust Features** che hanno le stesse caratteristiche di invarianza delle SIFT ma, a parità di immagine, sono meno e più stabili. Essendo inoltre più efficienti da calcolare, si otterrebbe un incremento nella velocità d'esecuzione.

Per velocizzare l'esecuzione si potrebbe provare a cambiare l'approccio con cui si effettua la ricerca delle corrispondenze tra feature. Nello stato attuale le feature vengono ricercate su tutta l'immagine e poi su questo insieme di feature vengono cercate le corrispondenze; essendo però nel nostro caso un frame video, per quanto gli spostamenti siano repentini, nella stragrande maggioranza dei casi lo spostamento dell'immagine da un frame al successivo è molto limitato. Si avrà così che le corrispondenze di una feature saranno

sempre in un certo intorno abbastanza ristretto della feature stessa. Detto questo basterà andare a cercare per ogni feature le corrispondenze solo in una regione limitata dell'immagine, centrata sulle coordinate della feature stessa.

In un prossimo sviluppo dell'applicativo è già stata considerata l'introduzione due migliorie:

- Ransac: l'utilizzo di Ransac come metodo per il calcolo dei valori di modulo e intensità dei vettori di moto; il problema riscontrato qui è dato dal fatto che, a causa delle errate corrispondenze che vengono trovate nella ricerca tra le feature di due oggetti, il valore del modulo e dell'angolo del vettore non risultano essere precisi come si vorrebbe ma sono invece soggetti a forte rumore. Questa imprecisione nasce dai valori errati del modulo e/o angolo delle singole feature e si presenta soprattutto quando il movimento dell'oggetto è poco marcato. L'uso di Ransac comporterebbe un notevole miglioramento della stabilità nel calcolo di questi valori.

- Mappatura nello spazio 3D: come già accennato precedentemente, per ottenere una stima realistica del moto di ogni oggetto rispetto alla scena, è necessario che le coordinate 2D dei punti sul piano immagine vengano mappate nello spazio 3D. Così facendo si potrebbero calcolare una stima del movimento nello spazio e tracciarne magari la traiettoria in un grafico in tre dimensioni. Per la mappatura è però necessario calcolare precedentemente il modello della camera.

Appendice A

Appendice

L'applicazione è stata implementata in C++ utilizzando come ambiente di sviluppo *Geany* [5] su piattaforma ArchLinux, un semplice editor con funzioni basilari di supporto per la programmazione; per il debugging durante lo sviluppo è stato utilizzato il GDB - Gnu Project Debugger con il supporto dell'interfaccia grafica DDD - Data Display Debugger.

A supporto del programma sono state utilizzate diverse librerie esterne; per quanto riguarda l'interfaccia grafica, l'estrazione dei frame video e l'elaborazione d'immagine ci si appoggia alle *OpenCv* - Open source Computer Vision [4] che forniscono una grande varietà di funzioni e algoritmi per la visione artificiale. Per le funzioni di motion tracking sempre parte delle OpenCv è l'implementazione del filtro di Kalman applicato nel calcolo della traiettoria del movimento di ogni singola feature e poi globalmente nella stima del movimento dei singoli oggetti e della camera andando a monitorare le variazioni di direzione e modulo del vettore di moto. La fase di individuazione degli oggetti si appoggia all'algoritmo per la segmentazione d'immagine proposto da Pedro F. Felzenszwalb in [8].

Per quanto invece riguarda la fase di estrazione delle feature, è stata uti-

lizzata l'implementazione open source in C++ della Scale Invariant Feature Transform [1] di Andrea Vedaldi[2] che permette una completa configurazione di tutti i parametri dell'algoritmo. Dall'implementazione originale è stato estrapolato il codice strettamente necessario e adattato all'interno dell'applicativo. Per la fase di ricerca delle corrispondenze tra feature è stato invece utilizzato il codice in C implementato da Rob Hess[3] e quindi tradotto poi in C++; le funzioni interessate sono quelle per l'indicizzazione dell'insieme di feature sulla base della versione modificata del *kd-tree* come illustrato da Lowe[1] e l'implementazione dell'algoritmo di ricerca ottimizzato, il *Best Bin First*.

Analizziamo qui come l'applicativo è strutturato e come sono definite le principali strutture dati.

A.1 File principali

- **main.cpp:** è il nucleo dell'applicazione ed è strutturato sulla base di un loop principale sull'acquisizione dei frame video e nel quale vengono, ad ogni iterazione, estratte le feature dall'immagine, cercate le corrispondenze tra il set di feature del frame corrente e le feature di cui si tiene traccia dalle corrispondenze trovate precedentemente. All'inizio e ogni qualvolta il programma rilevi un completo cambio di scena la fase di segmentazione e individuazione degli oggetti viene ripetuta. Infine viene aggiornata la traccia dei vettori di moto e gli stessi vengono disegnati nel frame del video di output.

- **support.cpp:** questo file contiene tutte le funzioni di appoggio dell'applicazione come ad esempio quella per l'estrazione delle SIFT feature, la funzione per l'interfacciamento con le funzioni di ricerca delle

corrispondenze che utilizzano il kd-tree e il metodo Best Bin First, e ancora le procedure di supporto per l'inizializzazione del filtro di Kalman e per il suo aggiornamento.

- **support.h:** in questo header sono contenute tutte le costanti globali, le dichiarazioni delle strutture dati e tutti i prototipi delle funzioni

Inoltre sono presenti altri tre insiemi di file contenenti le implementazioni della SIFT, del kdtree e della graph-based segmentation rispettivamente dai lavori di Vedaldi, Hess e Felzenszwalb.

A.2 Struct *feature*

Vediamo ora come è definita la struttura dati di una singola feature:

```
struct feature{
  double x;                       //x coord
  double y;                       //y coord
  double scl;                     //scale
  double ori;                     //orientation
  double descr[FEATURE_MAX_D];    //128D descriptor
  struct feature* fwd_match;      //match
  featureTrace * ft;              //used for kalman
};
```

I campi x e y sono le coordinate del keypoint nell'immagine, *scl* rappresenta la scala dello "scale-space" a cui il keypoint è stato trovato, *ori* è l'orientazione del keypoint calcolata come precedentemente illustrato nel capitolo 2, notare che la SIFT può generare feature con uguale posizione ma differente orientazione; *descr* è il descrittore delle SIFT ed consiste in

un vettore a 128 dimensioni che rappresenta univocamente la feature, è il campo più significativo perchè riassume tutte le informazioni riguardanti la feature e viene infatti utilizzato poi per la ricerca delle corrispondenze nella fase successiva. Durante la ricerca, ogni qualvolta una corrispondenza viene trovata tra la feature corrente e un'altra, il puntatore all'altra viene salvato nel campo *fwd_match*. Infine il campo *ft* contiene il puntatore ad un oggetto di tipo *featureTrace* utilizzato per mantenere le informazioni associate al filtro di Kalman applicato alla posizione del keypoint per poi tenerne traccia nei frame successivi.

A.3 Struct *featureTrace*

Il funzionamento del buffer contenente le feature degli N frame precedenti si basa sull'utilizzo di una struttura dati contenuta in ogni feature e così definita:

```
struct featureTrace{
    featureTrace(){ ... }
    ~featureTrace(){ ... }

    CvKalman * kalman;     //the kalman structure
    const CvMat * predict; //the predicted position
    const CvMat * correct; //the corrected position
    //first frame in which the feature has been found
    int firstFrame;
    //last frame in which the feature has been found
    int lastFrame;
};
```

I campi *kalman*, *predict* e *correct* servono per il calcolo della stima della posizione della feature con il filtro di kalman, *firstFrame* e *lastFrame* servono invece per dedurre l'"età" della feature in modo tale da poter eliminare la stessa dal buffer in caso sia troppo "vecchia" per essere mantenuta nella traccia.

A.4 Struct *object*

Per mantenere le informazioni relative ad ogni oggetto ci si appoggia alla struttura dati *object* definita come segue:

```
struct object{
object(){...}

vector<int> points;//vector of indexes to the trj vector
floatPoint center; //object center (from the contour moments)
float angle;       //motion direction(0-359.9999)
float module;      //movement vector module
vector<floatPoint> path; //movement trace

CvKalman * kalman; //track the angle and module
const CvMat * predict; //the predicted values by kalman
const CvMat * correct; //the corrected values by kalman

CvScalar color;    //color
CvSeq hull;        //the object's convex hull
};
```

- *points* è un vettore di indici degli elementi della traccia dei vettori di movimento *trj*, specifica ovvero quali feature sono associate all'oggetto e che verranno quindi usate per calcolarne la stima del moto;

- *center* è il centro del contorno e viene calcolato utilizzando i momenti (*cvMoments*) in questo modo: $center.x = m10/m00$; e $center.y = m01/m00$;.

- *angle,module* rappresentano la direzione e il modulo del vettore di moto

- *path* un vettore di punti che mantiene le precedenti posizioni dell'oggetto e che serve quindi a tracciare gli spostamenti dello stesso

- *kalman* è la struttura del filtro di kalman per la stima di angolo e modulo del vettore di moto

- *hull* è il convex hull dell'insieme dei punti dell'oggetto che serve per disegnarne il contorno

Bibliografia

[1] D. G. Lowe, *"Distinctive Image Features from Scale-Invariant Keypoints"*, International Journal of Computer Vision, 60, 2, pp. 91-110, 2004

[2] A. Vedaldi, *"SIFT++"*, University of California, 2006 `http://www.vlfeat.org/~vedaldi/code/siftpp.html`

[3] Rob Hess, *"SIFT feature detector"*, `http://web.engr.oregonstate.edu/~hess/`

[4] OpenCV, *"Open Source Computer Vision"*, Intel Corporation, `http://opencv.willowgarage.com/wiki/`

[5] Geany, `http://www.geany.org/`

[6] SVN, *"SubVersion"*, `http://subversion.apache.org/`

[7] gprof, *The GNU profiler*, `http://www.cs.utah.edu/dept/old/texinfo/as/gprof_toc.html`

[8] Pedro Felzenszwalb, *Efficient graph-based Image segmentation*, `http://people.cs.uchicago.edu/~pff/segment/`