



# UNIVERSITÀ DEGLI STUDI DI CATANIA

DIPARTIMENTO DI MATEMATICA E INFORMATICA

CORSO DI LAUREA MAGISTRALE IN INFORMATICA

---

*Rosario Scalia*  
(1000008648)

Progetto di Computer Vision

---

Anno Accademico 2019 - 2020

# Indice

<b>1</b>	<b>Introduzione</b>	<b>3</b>
1.1	Visual Tracking . . . . .	3
1.2	Difficoltà del Task . . . . .	3
1.3	Impostazione Tipica degli Algoritmi di Tracking . . . . .	4
1.4	Single-Object Tracking . . . . .	4
1.5	Multi-Object Tracking . . . . .	5
1.6	Scopo del Progetto . . . . .	6
<b>2</b>	<b>Algoritmi</b>	<b>7</b>
2.1	Single-Object Tracking . . . . .	7
2.1.1	TLD . . . . .	7
2.1.1.1	Modello del Target . . . . .	7
2.1.1.2	Tracking . . . . .	8
2.1.1.3	Learning . . . . .	8
2.1.1.4	Detection . . . . .	10
2.1.1.5	Pipeline generale dell'algoritmo. . . . .	11
2.1.2	CSRT . . . . .	12
2.1.2.1	Estrazione delle Feature . . . . .	12
2.1.2.2	Strategia di Localizzazione . . . . .	13
2.1.2.3	Pipeline Generale dell'Algoritmo. . . . .	17
2.1.3	LK Tracker . . . . .	17
2.1.3.1	Implementazione Piramidale . . . . .	18
2.1.3.2	Approccio Iterativo all'Ottimizzazione . . . . .	19
2.1.3.3	Pipeline Generale dell'Algoritmo . . . . .	22
2.2	Multi-Object Tracking . . . . .	22
2.2.1	Centroid Tracker . . . . .	22
2.2.2	SORT . . . . .	23
<b>3</b>	<b>Dataset</b>	<b>25</b>
3.1	MOT Challenge . . . . .	25
3.2	Formato Dati . . . . .	25
3.3	Dati Impiegati . . . . .	26
<b>4</b>	<b>Metriche di Valutazione</b>	<b>27</b>
4.1	Metriche Generali . . . . .	27
4.1.1	Metriche di Base . . . . .	27
4.1.2	Metriche Complesse . . . . .	28
4.2	Single-Object Tracking . . . . .	28

4.3	Multi-Object Tracking . . . . .	29
<b>5</b>	<b>Esperimenti Quantitativi</b>	<b>32</b>
5.1	Single-Object Tracking . . . . .	33
5.2	Multi-Object Tracking . . . . .	38
<b>6</b>	<b>Esperimenti Qualitativi</b>	<b>45</b>
<b>7</b>	<b>Conclusione</b>	<b>47</b>
	<b>Bibliografia</b>	<b>48</b>

# Capitolo 1

## Introduzione

### 1.1 Visual Tracking

Negli ultimi anni, il trend dell’acquisizione video sta crescendo sempre di più in una moltitudine di settori.

I motivi di tale crescita sono riconducibili sia ai bassi costi dei dispositivi di acquisizione che alla disponibilità di un’adeguata potenza di calcolo per elaborare i filmati.

In tale contesto, è sempre maggiore la richiesta di algoritmi che riescano ad estrapolare informazioni semanticamente significative dai filmati acquisiti.

In particolar modo, questa trattazione vuole affrontare il problema del Visual Tracking.

Tale task, consiste nel *rilevare* e *tracciare* la posizione di un oggetto di interesse (o *target*) all’interno di un filmato.

Il suddetto target è rappresentato attraverso dei valori di intensità luminosa; per tanto è importante che gli algoritmi di tracking riescano a modellare la relazione che sussiste fra il target e i suoi pixel.

### 1.2 Difficoltà del Task

In generale, il task del Tracking nasconde una serie di insidie che lo rendono un compito impegnativo.

In particolar modo, le difficoltà più frequenti sono riassunte di seguito:

- **Clutter**, spesso il target è *visivamente simile* a tanti altri oggetti presenti nella scena.

Tutto ciò rende più difficile la sua localizzazione.

- **Variazione dell’Aspetto del Target**, durante l’evoluzione del filmato, un target può cambiare molte volte il suo aspetto visivo per una serie di motivi.

I suddetti motivi possono essere ricondotti ad un cambio di posa oppure ad un cambio repentino dell’illuminazione ambientale.

Inoltre, le suddette variazioni possono essere causate anche dal rumore introdotto dal processo di acquisizione delle immagini.

Infine, un'altra causa di variazione dell'aspetto è *l'occlusione*; tale fenomeno si verifica quando un oggetto ostruisce la visibilità del target rispetto al punto di vista della camera.

### 1.3 Impostazione Tipica degli Algoritmi di Tracking

In letteratura, sono presenti una moltitudine di approcci diversi per affrontare il problema del tracking.

Ciò nonostante, esiste una struttura comune in tutti gli algoritmi di tracking, tale struttura è illustrata di seguito:

- **Rappresentazione dello Stato del Target**, questa fase permette di scegliere come rappresentare lo stato del target.

Spesso, tale scelta è frutto di un compromesso tra accuracy e persistenza del tracking.

- **Estrazione delle Feature**, questo processo permette di codificare in forma vettoriale le caratteristiche visive della regione dove è presente il target.

- **Localizzazione**, è un meccanismo che permette di propagare lo stato del target attraverso il tempo.

L'obiettivo è inferire lo stato del target ad ogni frame del filmato.

Per tale fase, è previsto l'impiego sia delle feature estratte che degli stati passati del target.

- **Gestione del Tracking**, questa fase implica l'utilizzo di un'apposita logica per gestire l'apparizione o la scomparsa di un target dal campo visivo della camera.

Inoltre, i Tracker vengono suddivisi in due grandi categorie:

- **Single-Object**, in questa casistica viene tracciato un solo target.
- **Multi-Object**, vengono tracciati target multipli.

### 1.4 Single-Object Tracking

Il problema del Single-Object Tracking può essere espresso attraverso la seguente notazione matematica:

- Sia  $\mathbf{I} = \{I_k \in E_I : k \in \mathbb{N}\}$  l'insieme di immagini appartenenti al filmato preso in esame.

Ogni immagine  $I_k$  appartiene allo spazio di tutte le immagini possibili  $E_I$ .

Inoltre, l'indice  $k$  enumera i frame presenti nel filmato.

- A partire dall'immagine  $I_k$ , Il Single-Object Tracking prevede la stima della seguente serie temporale caratterizzata dallo stato del target ad ogni frame del filmato:

$$\mathbf{x} = \{x_k \in E_s : k \in \mathbb{N}\} \quad (1.1)$$

La serie temporale appena citata viene chiamata *traiettoria* in letteratura. Inoltre, ogni vettore della serie appartiene allo spazio degli stati del target  $E_s$ .

- In generale, il mapping fra immagine e stato del target non è diretto. Del resto, vi è di mezzo una trasformazione dello spazio delle immagini  $E_I$  in un nuovo spazio  $E_o$ .

L'obiettivo di questo processo è evidenziare maggiormente le caratteristiche utili per il tracking.

Inoltre, è frequente avere anche una riduzione della dimensionalità quando si effettua la suddetta trasformazione.

Per inciso, il processo appena raccontato coincide con l'estrazione delle feature descritta nella sezione precedente.

Dal punto di vista matematico, lo spazio delle feature è descritto dal seguente insieme:

$$E_o = \{z_k \in \mathbb{R}^n : k \in \mathbb{N} \wedge \phi(I_k) = z_k\} \quad (1.2)$$

$\phi(\cdot)$  = Feature Extractor

Ogni vettore  $z_k$  di tale insieme viene chiamato *Osservazione*, inoltre l'insieme delle osservazioni presenti in un video viene indicato con la seguente notazione:

$$\mathbf{Z} = \{z_k \in E_o : k \in \mathbb{N}\} \quad (1.3)$$

**Digressione 1** (Informazioni contenute nello Stato del Target) *Solitamente, la tipologia di informazioni contenute nello stato del target sono legate all'applicativo che si vuole sviluppare.*

*Ciò nonostante, esistono una serie di informazioni che vengono utilizzate frequentemente in letteratura:*

- *Informazioni sulla Posizione e sulle caratteristiche visive del target.*
- *Informazioni sulla variazione delle caratteristiche visive del target.*

## 1.5 Multi-Object Tracking

Il Multi-Object Tracking prevede il tracciamento di target multipli per ogni frame del filmato.

La sua trasposizione matematica è fortemente basata su quella presentata per il Single-Object Tracking.

Le uniche modifiche sono le seguenti:

- Stati dei Target:

$$\begin{aligned}\mathbf{x} &= \{x_k \in \mathbb{W}(E_s) : k \in \mathbb{N}\} \\ x_k &= \{x_{k,1}, x_{k,2}, \dots, x_{k,n},\} \\ \mathbb{W}(E_s) &= \text{Collezione finita di tutti i sottoinsiemi di } E_s\end{aligned}\tag{1.4}$$

- Osservazioni:

$$\begin{aligned}\mathbf{z} &= \{z_k \in \mathbb{F}(E_o) : k \in \mathbb{N}\} \\ z_k &= \{z_{k,1}, z_{k,2}, \dots, z_{k,n},\} \\ \mathbb{F}(E_o) &= \text{Collezione finita di tutti i sottoinsiemi di } E_o\end{aligned}\tag{1.5}$$

## 1.6 Scopo del Progetto

Questa trattazione pone l'obiettivo di confrontare le prestazioni di 5 algoritmi di Tracking attraverso un benchmark noto in letteratura, ovvero il *MOT Benchmark* [2].

Nello specifico, gli algoritmi scelti sono elencati di seguito:

- **Lucas-Kanade Tracker** [12] - Single-Object Tracking.
- **CSRT** [8] - Single-Object Tracking
- **TLD** [3] - Single-Object Tracking.
- **SORT** [14] - Multi-Object Tracking.
- **Centroid Tracker** - Multi-Object Tracking.

# Capitolo 2

## Algoritmi

### 2.1 Single-Object Tracking

#### 2.1.1 TLD

L'algoritmo *Tracking-Learning-Detection (TLD)* [3] presentato da Kalai et. al. nel 2010 è uno fra gli algoritmi più interessanti prodotti nell'ambito della ricerca sul Visual Tracking.

L'importanza di tale algoritmo non è legata solamente alle sue prestazioni.

Del resto, quest'ultimo si configura come un ottimo framework per sviluppare ulteriori algoritmi di tracking.

L'algoritmo TLD è basato su tre fasi distinte e concorrenti:

- **Tracking**, in questa fase viene stimato il movimento del target *frame-per-frame*.

Inoltre, si assume il fatto che il tracker *fallisca frequentemente* quando il target esce dal FOV della camera.

Infine, lo stato del target viene rappresentato attraverso una bounding-box ,di aspect ratio fisso, avente il seguente formato:  $(x, y, x_{max}, y_{max})$ .

- **Detector**, analizza l'intera immagine sfruttando una sua suddivisione in patch.

Il suo obiettivo è stimare ,in ogni patch, la presenza o meno del target.

Per portare a termine i suoi scopi, il detector viene aiutato da una fase di *online learning*; quest'ultima gli permette di guadagnare *una certa invarianza prestazionale* rispetto alle variazioni di aspetto del target lungo il trascorrere del flusso video.

- **Learning**, rileva gli errori del Detector (*Falsi Positivi e Falsi Negativi*), incrementa il numero di osservazioni del training-set ed effettua gli addestramenti del detector.

##### 2.1.1.1 Modello del Target

Durante l'esecuzione dell'algoritmo TLD, viene mantenuto un Modello del Target.

Quest'ultimo consiste in una struttura dati (sia essa  $M$ ) contenente una lista finita di patch positive (appartenenti al target) e negative (non appartenenti al target) inferite lungo il flusso video.

$$M = \{p_1^+, p_2^+, \dots, p_m^+ | p_1^-, p_2^-, \dots, p_n^-\} \quad (2.1)$$

Le patch sono ottenute campionando la bounding-box selezionata dal tracker.

Tale processo restituisce una serie di patch che poi verranno ricampionate ad una risoluzione  $15 \times 15$ .

La similarità fra due patch è definita dalla seguente formula:

$$S(p_i, p_j) = 0.5 \cdot (NCC(p_i, p_j) + 1) \quad (2.2)$$

La funzione  $NCC$  rappresenta l'operatore di Correlazione Normalizzato.

Dalla similarità appena mostrata è possibile ricavarne altre che saranno utili durante la computazione di TLD:

- $S^+(p, M) = \max_{p_i^+ \in M} S(p, p_i^+)$
- $S^-(p, M) = \max_{p_i^- \in M} S(p, p_i^-)$
- $S_{50\%}^+(p, M) = \max_{[p_i^+ \in M \wedge i < \frac{m}{2}]} S(p, p_i^+)$
- $S^r(p, M) = \frac{S^+}{S^+ + S^-}$
- $S^c(p, M) = \frac{S_{50\%}^+}{S_{50\%}^+ + S^-}$

**Aggiornamento del Modello.** La similarità  $S^r$  fra due patch viene sfruttata per definire un classificatore  $1\text{-NN}$ .

Una patch  $p$  viene classificata come positiva se  $S^r(p, M) > \theta_{NN}$ , altrimenti viene classificata come negativa.

Detto ciò, una patch  $p$  viene aggiunta al modello solamente se il classificatore  $1\text{-NN}$  e le funzioni  $P$  e  $N$  (*che verranno illustrate di seguito*) divergono nella stima dell'etichetta della patch.

### 2.1.1.2 Tracking

Nell'implementazione proposta, è stato sfruttato l'algoritmo *KCFC* [6].

È da sottolineare che quest'ultimo è diverso da quello originario impiegato dagli autori, che per inciso era MedianFlow [7].

### 2.1.1.3 Learning

Il processo di learning raccontato nelle sezioni precedenti sfrutta una particolare tecnica di *Self-Supervised Learning*: il *P-N Learning* [4].

Quest'ultima esibisce le seguenti caratteristiche:

- Sia  $X$  lo spazio delle feature.

- Sia  $x \in X$  un vettore di feature associate ad una generica patch di una generica immagine.

- Sia  $Y$  lo spazio delle etichette, nel nostro caso  $y \in Y \wedge y \in \{0, 1\}$ .

Quando l'etichetta assume il valore 1 significa che la patch attualmente analizzata appartiene al target (patch positiva).

Viceversa, significa che la patch attualmente analizzata non vi appartiene (patch negativa).

- Sia  $L = (x^{(i)}, y^{(i)})$  il training-set.

- Sia  $U = (x^{(i)})$  un set di dati non etichettato.

- Sia  $h_\theta : X \rightarrow Y$  l'object detector, nello specifico quest'ultimo è un classificatore binario di patch.

Per tanto nel resto della trattazione verranno usati indistintamente entrambi i termini.

- Come accennato nella sezione precedente, il P-N Learning ha il compito di rilevare gli errori del detector.

Quest'ultimi possono essere suddivisi in due categorie: Falsi Positivi e Falsi Negativi.

Il P-N Learning ,a sua volta, si basa sulle seguenti due funzioni per la rilevazione degli errori:

$$\begin{aligned} P : U &\longrightarrow Y \\ N : U &\longrightarrow Y \end{aligned} \tag{2.3}$$

La funzione  $P$  ha l'obiettivo di rilevare i falsi negativi restituiti dal detector (etichetta restituita pari a 0 mentre il ground-truth era 1).

Una volta rilevato un Falso Negativo ,sia esso  $x^{(k)} \in U$ , viene aggiunta una nuova tupla con etichetta cambiata al training-set: ovvero  $(x^{(k)}, y^{(k)} = 1) \in L$ .

La funzione  $N$  ha l'obiettivo di rilevare i falsi positivi restituiti dal detector (etichetta restituita pari ad 1 mentre il ground-truth era 0).

Una volta rilevato un Falso Positivo ,sia esso  $x^{(k)} \in U$ , viene aggiunta una nuova tupla con etichetta cambiata al training-set: ovvero  $(x^{(k)}, y^{(k)} = 0) \in L$ .

- La pipeline completa della fase di learning del TLD consiste in un processo iterativo caratterizzato dai seguenti passi:

1. Inferire le etichette di  $U$  sfruttando  $h_\theta$ .
2. Stimare gli errori del classificatore sfruttando le funzioni  $P$  e  $N$ .
3. Incrementare la cardinalità di  $L$  aggiungendo *opportunamente* le osservazioni di  $U$  classificate in maniera erronea dal detector.
4. Addestrare il classificatore sfruttando il *nuovo training-set*.
5. Cicla ad 1 fino a quando non si verifica un criterio di arresto.

Per quanto detto, possiamo dire che la funzione  $P$  incrementa la recall del classificatore mentre la funzione  $N$  incrementa la sua precision.

In ultima analisi, resta da illustrare la logica che seguono le due funzioni.

Nello specifico, la funzione  $P$  richiede che le patch *vicine* alla traiettoria stimata siano di classe positiva.

Invece, la funzione  $N$  richiede che le patch *lontane* alla traiettoria stimata siano di classe negativa.

Per tanto, possiamo concludere che le funzioni  $P$  e  $N$  sono dei vincoli che affondano le loro radici nella dipendenza spazio-temporale presente nel task del video tracking.

#### 2.1.1.4 Detection

Il detector dell'algoritmo TLD si basa su due entità di base:

- **Costruzione di un set di Ancore**, in tale fase vengono costruite una serie di patch predefinite nella regione di immagine contenente la bounding-box iniziale data dall'utente.  
Tali patch sono frutto di un processo incrementale di shifting e variazione di scala della già citata bounding-box iniziale.
- **Cascata di Classificatori**, in tale fase viene sfruttata una successione di classificatori per ottenere la lista di patch che con maggior probabilità contengono il target.

La suddetta cascata di classificatori è composta da 3 componenti:

1. **Patch Variance**, in tale fase vengono rigettate tutte le patch che hanno una varianza in scala di grigio minore del 50 % di quella della bounding-box selezionata per il tracking.
2. **Ensable di Classificatori**, i classificatori dell'ensamble (nello specifico l'ensamble è una Random Forest [5]) prendono in input tutte le patch che non sono state rigettate dal Patch Variance.

**Inferenza.** Per ogni patch e per ogni classificatore dell'ensamble viene effettuata la seguente computazione:

- (a) Genera il codice binario  $x$  a partire da una serie di confronti fra alcuni pixel della patch attualmente in esame.
- (b) Stima la probabilità a posteriori  $P_i(y|x)$   $y \in \{0, 1\}$ .  
Tale distribuzione è una distribuzione a  $2^k$  stati, dove  $k$  è il numero di confronti assegnati ad ogni classificatore.
- (c) Effettua una media della distribuzione  $P_i(y|x)$ , sia essa  $P_i^{(AVG)}(y|x)$ .

A questo punto, la patch in esame viene classificata come positiva se la media delle  $P_i^{(AVG)}(y|x)$  è maggiore di 0.5.

Come accennato nella precedente descrizione, ogni classificatore dell'ensamble sfrutta una serie di confronti fra pixel all'interno della patch attualmente analizzata.

I risultati di tali confronti vengono concatenati in un codice binario che verrà impiegato per la stima delle probabilità a posteriori.

La scelta di quali confronti effettuare è demandata ad una strategia portata a termine in maniera random ed offline.

Inoltre, la generazione e distribuzione dei confronti fra i classificatori dell'ensamble viene effettuata rispettando l'indipendenza fra i classificatori; tale requisito è fondamentale per le prestazioni e viene soddisfatto assegnando in maniera esclusiva i confronti ai classificatori.

L'ultimo punto mancante alla descrizione dell'ensamble è la strategia di stima delle probabilità a posteriori.

Nello specifico, la stima della distribuzione  $P_i(y|x)$  viene demandata alla seguente formula:

$$P_i(y|x) = \frac{\#p}{\#p + \#n} \quad (2.4)$$

$\#p \Rightarrow$  Numero di Patch positive assegnate al codice  $x$ .  
 $\#n \Rightarrow$  Numero di Patch negative assegnate al codice  $x$ .

**Learning.** Il learning dell'ensamble di classificatori segue la seguente logica:

- Addestrare l'ensamble sfruttando il paradigma del *P-N Learning*.
  - (a) Stimare gli errori del classificatore ,sulle patch attuali, sfruttando le funzioni  $P$  e  $N$ .
  - (b) Incrementare la cardinalità di  $L$  aggiungendo *opportunamente* le osservazioni classificate in maniera erronea dal detector.
  - (c) Inferenza del classificatore sul *nuovo training-set aumentato*.
  - (d) Incremento opportuno di  $\#n$  e  $\#p$  quando l'ensamble sbaglia ad inferire la classe di una patch del training-set.
  - (e) Aggiornamento distribuzioni a Posteriori.
  - (f) Cicla ad (a) fino a quando non si verifica un criterio di arresto.

3. **Classificatore NN**, questo componente prende in input le patch rimaste dopo il filtraggio effettuato dall'ensamble di classificatori.

Nello specifico, l>NN-classifier classifica le patch restanti sfruttando il criterio di similarità  $S^r$  e la soglia  $\theta_{NN} = 0.6$ .

Le patch restanti rappresentano la risposta *nel suo insieme* del detector.

#### 2.1.1.5 Pipeline generale dell'algoritmo.

La pipeline esaustiva dell'algoritmo TLD viene riassunta di seguito:

- Inizialmente, l'utente fornisce una bounding-box iniziale rappresentante il target.

Da quest'ultima, viene generato il training-set sfruttando una serie di patch estratte dalla regione di immagine contenente la bounding-box.

Inoltre, viene inizializzata la struttura dati contenente il Modello del Target.

- Per ogni frame successivo a quello di inizializzazione, avviene la seguente computazione:

1. Il **Tracker** effettua la sua elaborazione ed eventualmente restituisce in output la bounding-box che dovrebbe contenere il target.
2. Il **Detector**, attraverso la sua computazione, restituisce in output una serie di bounding-box che sono candidate a contenere il target.  
Chiaramente, può accadere che nessuna bounding-box venga restituita dal detector; in quel caso significa che il detector non è riuscito a localizzare il target nella scena in base a quanto appreso in passato (magari per via di una copiosa alterazione del suo aspetto).
3. Viene calcolata la misura  $S^c$  sfruttando sia le bounding-box del detector che quelle del tracker.  
Fatto ciò, la bounding-box finale restituita dall'algoritmo sarà quella che massimizza la misura  $S^c$ .
4. Il Componente di **Learning** valuta le prestazioni del detector sulle patch attuali sfruttando le funzioni  $P$  e  $N$ .  
Fatto ciò, vengono aggiunte nuove osservazioni al training-set impiegando la logica raccontata nelle sezioni precedenti.  
Successivamente, viene avviato un nuovo addestramento del detector utilizzando il training.set.
5. Eventuale aggiornamento del **Modello del Target** sfruttando le correzioni apportate dalle funzioni  $P$  e  $N$ .

### 2.1.2 CSRT

L'algoritmo CSRT (Channel and Spatial Releability Tracker) è un single-object tracker basato sui *Filtri Discriminativi di Correlazione*.

Tale algoritmo, in sintesi, applica degli opportuni filtri di correlazione su di una regione di immagine e attraverso le risposte dei filtri permette di individuare la posizione più verosimile del target all'interno del frame.

#### 2.1.2.1 Estrazione delle Feature

Il primo passo per applicare il CSRT è estrarre una serie di mappe 2D contenenti le feature necessarie alla successiva correlazione.

Le mappe estratte vengono ricavate da una regione di immagine centrata sulla posizione della bounding-box che rappresenta il target al frame precedente a quello attuale.

*Osservazione 1.* Nel caso del frame iniziale, è previsto l'inserimento della bounding-box dall'utente.

Le feature estratte appartengono a tre tipologie differenti, ovvero:

- HoG (Histograms of Gradient)
- Colornames
- Valori a Tono di Grigio

### 2.1.2.2 Strategia di Localizzazione

Una volta estratte le mappe di feature, è possibile applicarvi una lista di filtri di correlazione 2D e prelevare la posizione con risposta massima come nuova location del target ovvero:

$$\begin{aligned}
 c_w \times c_h &= \text{Shape delle Mappe di Feature e dei Filtri di Correlazione} \\
 f &= \text{Lista di Mappe di feature} \\
 h &= \text{Lista di Filtri di Correlazione} \\
 N_m &= \text{Numero di Mappe/Filtri di Feature/Correlazione} \\
 \tilde{g}(f, h) &= \text{Convoluzione Circolare dei filtri sulle mappe} \\
 \tilde{g}(f, h) &= \sum_{d=1}^{N_m} f_d \star h_d \\
 \text{new\_position} &= \text{Posizione con massima risposta di } \tilde{g}(f, h)
 \end{aligned}
 \tag{2.5}$$

A questo punto, la lista di filtri *ottima* viene ottenuta minimizzando la seguente funzione costo:

$$\begin{aligned}
 J(h) &= \sum_{d=1}^{N_m} \| [f_d \star h_d] - g \|^2 + \lambda \| h_d \|^2 \\
 g &= \text{Gaussiana 2D di Ground-Truth centrata nella posizione del Target} \\
 g &\in c_w \times c_h
 \end{aligned}
 \tag{2.6}$$

Inoltre, è importante sottolineare che il potere discriminativo tra le varie feature (HoG, Toni di Grigio ecc...) è diverso; tutto ciò porta la necessità di modificare la funzione di convoluzione circolare  $\tilde{g}(f, h)$  in modo tale da tenere conto di tale caratteristica:

$$\tilde{g}(f, h) = \sum_{d=1}^{N_m} f_d \star h_d \cdot \tilde{w}_d
 \tag{2.7}$$

Il vettore di pesi  $\tilde{w}$  permette di pesare *opportunamente* la capacità discriminativa delle mappe di feature (o canali); per tanto i pesi del vettore vengono denominati *Channel Releability Weights*.

**Learning dei Filtri di Correlazione.** Il learning dei filtri di correlazione è indipendente al livello del singolo filtro.

Per tale motivo ,nel resto della trattazione, verrà illustrato il learning di un singolo filtro  $h = h_d \quad \forall d \in \{1, 2, \dots, N_m\}$ .

Detto ciò, un concetto di fondamentale utilità per il processo di learning è la *Spatial Releability Map*; quest'ultima è una mappa binaria che indica quali pixel della regione di interesse verosimilmente appartengono al target (valore 1) oppure non vi appartengono (valore 0).

Ai fini della spiegazione della sezione corrente, si supponga che venga data *esogenatamente* una Spatial Releability Map; sia essa  $m$ .

Allora, il filtro  $h$  che si vuole apprendere è data dalla seguente espressione:

$$\begin{aligned}
 h &= m \odot h \\
 \odot &= \text{Prodotto punto-punto fra matrici.}
 \end{aligned}
 \tag{2.8}$$

Quest'ultima espressione del filtro  $h$ , implicitamente, introduce un vincolo al processo di learning/ottimizzazione nel suo insieme.

Inoltre, l'aggiunta del suddetto vincolo implica un processo di learning *iterativo*; in particolar modo verrà preso spunto dalla tecnica illustrata in [9].

Detto ciò, il passo successivo da effettuare è introdurre la variabile duale  $h_c$  ed imporre il vincolo:

$$h_c - m \odot h \equiv 0 \quad (2.9)$$

A questo punto, è possibile definire una *nuova funzione costo* denominata *Lagrangiana Aumentata*:

$$\begin{aligned} \mathcal{L}(\hat{h}_c, h, \hat{I}|m) = & \| \text{diag}(\hat{f}) \overline{\hat{h}_c} - \hat{g}(f, h) \|^2 + \frac{\lambda}{2} \cdot \| h_m \|^2 + \\ & + [\hat{I}(\hat{h}_c - \hat{h}_m) + \overline{\hat{I}(\hat{h}_c - \hat{h}_m)}] + \mu \cdot \| \hat{h}_c - \hat{h}_m \|^2 \end{aligned}$$

$\overline{(\cdot)}$  = Complesso coniugato

$\hat{x}$  = Trasformata di Fourier di  $x$  e conseguente reshape in un vettore colonna.

$\text{diag}(\hat{x})$  = Matrice diagonale avente come valori la trasformata di Fourier di  $x$

$\text{diag}(\hat{x}) \in \mathbb{C}^{D \times D} : D = c_w \times c_h$

$h_m = m \odot h$

$f$  = Mappa di feature da sfruttare per il learning.

$\hat{I}$  = Moltiplicatore Lagrangiano Complesso

$\mu$  = Scalare maggiore di 0.

(2.10)

La Loss Function appena mostrata esibisce una caratteristica notevole, ovvero l'impiego della trasformata di fourier.

Tale trasformata viene impiegata per via della maggior efficienza di calcolo che possiede il dominio frequenziale rispetto a quello temporale per tale tipologia di computazione.

Detto ciò, la Lagrangiana mostrata può essere minimizzata attraverso un processo iterativo.

Ad ogni iterazione del processo, vengono minimizzati i seguenti due oggetti:

$$\hat{h}_c^{i+1} = \underset{h_c}{\text{argmin}} \mathcal{L}(\hat{h}_c, h^i, \hat{I}^i|m) \quad (2.11)$$

$$h^{i+1} = \underset{h}{\text{argmin}} \mathcal{L}(\hat{h}_c^{i+1}, h, \hat{I}^i|m) \quad (2.12)$$

Il moltiplicatore lagrangiano complesso ,ad ogni iterazione, viene aggiornato dalla seguente espressione:

$$\hat{I}^{i+1} = \hat{I}^i + \mu \cdot (\hat{h}_c^{i+1} - \hat{h}^{i+1}) \quad (2.13)$$

Le equazioni (2.11) e (2.12) possono essere minimizzate ,rispettivamente, in forma chiusa sfruttando le seguenti due formule:

$$\begin{aligned} \hat{h}_c^{i+1} &= (\hat{f} \odot \overline{\hat{g}} + (\mu \hat{h}_m^i - \hat{I}^i)) \odot^{-1} (\overline{\hat{f}} \odot \hat{f} + \mu^i) \\ \mu^{i+1} &= \beta \cdot \mu^i \\ \odot^{-1} &= \text{Divisione punto-punto fra matrici} \end{aligned} \quad (2.14)$$

$$\hat{h}^{i+1} = m \odot \frac{\mathcal{F}^{-1}[\hat{I}^i + \mu^i \cdot \hat{h}_c^{i+1}]}{\frac{\lambda}{2D} + \mu^i} \quad (2.15)$$

$\mathcal{F}^{-1}$  = Trasformata di Fourier Inversa

**Costruzione della Spatial Releability Map.** Il prerequisito per la costruzione della Spatial Releability Map è la localizzazione del target nell'immagine.

Da tale localizzazione, viene ricavata una regione di training; quest'ultima sarà utile sia per costruire la mappa che per apprendere i filtri di correlazione.

Inoltre, dopo aver estratto la regione di training si provvederà a definire il modello di colore del Background e del Foreground, per fare ciò si impiegano gli Istogrammi di colori.

Dal punto di vista della costruzione della mappa, la training region rappresenta una lista di osservazioni colore-posizione  $y_i = (y_i^c, y_i^x)$ ; inoltre la variabile che rappresenta il valore di un generico pixel della mappa  $m_i$  viene vista come una Variabile Aleatoria.

Detto ciò, per costruire la Spatial Releability Map viene sfruttato un particolare framework matematico, ovvero il *Markov Random Field*.

In tale contesto le probabilità a priori  $P(m_i = 0)/P(m_i = 1)$  e a posteriori  $P(m_i = 0 | y_i)/P(m_i = 1 | y_i)$  sono trattate come Variabili Aleatorie.

Inoltre, è possibile visualizzare le probabilità a priori di ogni pixel della mappa come un vettore  $\pi_i = [\pi_{i0}, \pi_{i1}]$ .

Fatto ciò, verranno portate a compimento le seguenti azioni:

- Approssimazione della pdf:

$$\sum \pi_i \sim \prod P(\pi_i | \pi_{N_i}) \quad (2.16)$$

L'oggetto  $\pi_{N_i}$  rappresenta un Mixture Model dei  $\pi_z$  appartenenti ai pixel vicini al pixel  $i$ .

- L'oggetto  $P(\pi_i | \pi_{N_i})$  viene chiamato *Potenziale* nel MRF e viene calcolato con la seguente formula:

$$\begin{aligned} & \exp^{-\frac{1}{2} \cdot E(\pi_i, \pi_{N_i})} \\ & E(\pi_i, \pi_{N_i}) = D(\pi_i || \pi_{N_i}) + H(\pi_i) \\ & D(x || y) = \text{Disuguaglianza di Kullback, misura differenza fra distribuzioni} \\ & H(x) = \text{Entropia di } x \end{aligned} \quad (2.17)$$

- Ottimizzazione della seguente funzione energia:

$$F = \sum_{i=1}^{N_{pixel}} \left[ \log p(y_i) - \frac{1}{2} \cdot (E(\pi_i, \pi_{N_i}) + E(p_i, p_{N_i})) \right] \quad (2.18)$$

L'ottimizzazione dell'energia viene portata a termine sfruttando il metodo presentato da Diplaros et. al. [10].

Inoltre, le variabili da ottimizzare sono le seguenti:

- Probabilità a Priori su ogni pixel della mappa ( $\pi$ ).
  - Probabilità a Posteriori su ogni pixel della mappa ( $p$ ).
- Inoltre, ogni probabilità a posteriori di ogni pixel della mappa viene indicata con la seguente espressione:

$$p_i = [p_{i0} = P(m_i = 1 | y_i), p_{i1} = P(m_i = 1 | y_i)] \quad (2.19)$$

Infine, il termine  $p_{N_i}$  come nel caso delle probabilità a priori rappresenta il Mixture Model delle probabilità associate ai pixel vicini al pixel attualmente analizzato dalla probabilità a posteriori.

- Sogliatura delle probabilità a posteriori  $p$  a 0.5, tale processo restituirà i valori finali della mappa  $m$ .

**Stima della Channel Releability.** Il processo di stima della Channel Releability confluisc nel calcolo dei pesi da assegnare ad ogni mappa di feature estratta, ovvero la stima del vettore  $\tilde{w}$ .

Entrando nel dettaglio, possiamo dire che il peso della generica mappa (o canale)  $d$  viene calcolato dalla seguente espressione:

$$\begin{aligned} \tilde{w}_d &= \tilde{w}_d^{(lrn)} \cdot \tilde{w}_d^{(loc)} \\ \tilde{w}_d^{(lrn)} &= \text{Peso stimato in fase di learning} \\ \tilde{w}_d^{(loc)} &= \text{Peso stimato in fase di Localizzazione} \end{aligned} \quad (2.20)$$

Per la stima dei pesi in fase di learning viene impiegata la seguente strategia:

- L'ottimizzazione (2.10) include un *effetto collaterale* utile per la stima di  $\tilde{w}_d^{(lrn)}$ : ovvero il fatto che la risposta dei filtri appresi su feature con basso potere discriminativo sono sempre più basse rispetto alle stesse risposte su feature con elevato potere discriminativo.
- Per quanto detto, è possibile stimare il Channel Releability Weight di learning ,per la mappa  $d$ , impiegando la seguente formula:

$$\tilde{w}_d^{(lrn)} = \max(f_d \star h_d) \quad (2.21)$$

Invece, per la stima dei pesi in fase di localizzazione viene utilizzata la seguente ulteriore strategia:

- L'idea dei pesi di localizzazione è codificare *quanto ogni canale voti per una singola posizione del target*.

Per la stima dei suddetti pesi, viene impiegato il rapporto fra i due più alti picchi non adiacenti ricavati dalla risposta del filtro.

Nello specifico, la formula impiegata per la stima del peso di localizzazione associato alla mappa  $d$  è la seguente:

$$\tilde{w}_d^{(loc)} = \max\left(1 - \frac{\rho_d^{max^2}}{\rho_d^{max^1}}, 0.5\right) \quad (2.22)$$

### 2.1.2.3 Pipeline Generale dell'Algoritmo.

La pipeline generale dell'algoritmo CSRT viene delineata di seguito:

- **Prerequisiti**

1. Immagine al tempo attuale  $t$ , ovvero:  $I_t$
2. Posizione del target al tempo precedente  $t - 1$ , ovvero:  $p_{t-1}$ .
3. Lista di filtri di correlazione legata al tempo precedente  $t - 1$ , ovvero:  $h_{t-1}$ .
4. Modello di Colore  $c_{t-1}$  legata al tempo precedente  $t - 1$ .
5. Pesi di Channel Releability legati al tempo precedente  $t - 1$ , ovvero:  $w_{t-1}$ .

- **Localizzazione**

1.  $p_t =$  Posizione con massima risposta di  $\tilde{g}(f^t, h^{t-1})$
2. Calcolo della Localization Channel Releability attraverso la (2.22) e le risposte dei filtri calcolate in  $I$ .

- **Learning**

1. Estrazione della training region sfruttando la localizzazione attuale  $p_t$
2. Aggiornamento del modello di Background/Foreground sfruttando la nuova training region.
3. Calcolo della Releability Map.
4. Apprendimento dei Nuovi Filtri  $h_t$ .
5. Calcolo dei pesi della Learning Channel Releability. (2.21)
6. Calcolo dei Pesi della Channel Releability. (2.20)

### 2.1.3 LK Tracker

L'algoritmo di tracking proposto da Lucas e Kanade [12] nei primi anni 80 prevede il tracciamento di una serie di *keypoints* lungo una serie di immagini.

Nello specifico, in questo progetto è stata impiegata una versione *Iterativa e Piramidale* del suddetto algoritmo [11].

L'idea di base dell'algoritmo di Lucas e Kanade viene riassunta dai seguenti punti:

1. Sia  $I$  un'immagine Sorgente a scala di grigi.
2. Sia  $J$  un'immagine Destinazione a scala di grigi.
3. Sia  $n_x$  l'altezza delle immagini sorgente e destinazione.
4. Sia  $n_y$  la larghezza delle immagini sorgente e destinazione.
5. Sia  $u = [u_x, u_y]$  un punto nell'immagine sorgente  $I$ .

6. L'obbiettivo dell'algoritmo è stimare la posizione del punto  $u$  nell'immagine destinazione  $J$  (sia esso  $v = [v_x, v_y]$ ) in modo tale che le quantità  $I(u)$  e  $J(v)$  siano *simili*.
7. Il punto di destinazione  $v$  viene modellato sfruttando un'affinità ed una traslazione:  $v = Au + d$ .
8. La matrice affine  $A$  possiede la seguente formulazione:

$$A = \begin{pmatrix} 1 + d_{xx} & d_{xy} \\ d_{yx} & 1 + d_{yy} \end{pmatrix} \quad (2.23)$$

9. Il vettore di traslazione  $d$  possiede la seguente formulazione matematica:

$$d = [d_x, d_y] \quad (2.24)$$

Tale vettore viene spesso denominato *Optical Flow*.

10. Dal punto di vista matematico, la similarità richiesta al punto  $\theta$  viene garantita dalla minimizzazione del seguente obbiettivo:

$$E(d, A) = \sum_{x=-w_x}^{w_x} \sum_{y=-w_y}^{w_y} (I(x+u) - J(u + Ax + d))^2 \quad (2.25)$$

Gli scalari  $(w_x, w_y)$  rappresentano rispettivamente l'altezza e la larghezza di una regione di ricerca su cui opera l'algoritmo.

Del resto, un'ulteriore caratteristica che contraddistingue il metodo è la ricerca del target su di una regione di immagine limitata.

#### 2.1.3.1 Implementazione Piramidale

L'LK Tracker implementato sfrutta un approccio piramidale.

Nello specifico, a partire da un'immagine sorgente viene costruita una piramide contenente varie versioni sottocampionate dell'immagine originale.

Tutto ciò, conferisce maggior robustezza all'algoritmo nel suo insieme.

In generale, la computazione che sfrutta le Piramidi di Immagini sfrutta la seguente logica:

- Dal livello più profondo (sia esso  $L_m$ ) a quello meno profondo (sia esso 0) della piramide:

1. Il livello superiore  $L + 1$  passa al livello attuale  $L$  un suggerimento *iniziale* in merito alla soluzione del problema, sia esso:

$$\text{guess} = [g^{L+1}, G^{L+1}] \quad (2.26)$$

2. Minimizzazione del seguente obbiettivo sfruttando un approccio iterativo:

$$\begin{aligned} I^L(x) &= I^L(u^L + x) \\ J^L(x) &= u^L + G^{L+1}x + g^{L+1} \\ E^L(d^L, A^L) &= \sum_{x=-w_x}^{w_x} \sum_{y=-w_y}^{w_y} (I^L(x) - J(A^L x + d^L))^2 \end{aligned} \quad (2.27)$$

3. Costruzione del nuovo suggerimento per il livello  $L - 1$ :

$$\begin{aligned} guess &= [g^L, G^L] \\ g^L &= 2(g^{L+1} + G^{L+1}d^L) \\ G^L &= G^{L+1}A^L \end{aligned} \quad (2.28)$$

4. Cicla ad 1 fin quando non si raggiungono le fondamenta della piramide di immagini.

- Alla fine delle iterazioni, la soluzione finale del problema sarà data dalle seguenti formule:

$$\begin{aligned} d &= g^1 + G^1d^0 \\ G &= G^1A^0 \\ v &= Gu + d \end{aligned} \quad (2.29)$$

### 2.1.3.2 Approccio Iterativo all'Ottimizzazione

Come accennato nella precedente sezione, ad ogni livello della piramide viene attuato un processo di ottimizzazione iterativo.

Nel resto della trattazione, verrà affrontato in prima battuta la *logica di ottimizzazione* ed in seconda battuta l'integrazione di quest'ultima nel processo iterativo.

**Logica di Ottimizzazione.** Il processo di ottimizzazione coinvolto implica la minimizzazione del seguente obiettivo:

$$\begin{aligned} I(x) &= I(u + x) \\ J(x) &= u + Gx + g \\ E(d, A) &= \sum_{x=-w_x}^{w_x} \sum_{y=-w_y}^{w_y} (I(x) - J(Ax + d))^2 \end{aligned} \quad (2.30)$$

Per poter portare a termine l'ottimizzazione, è necessario che il gradiente si annulli, ovvero:

$$\begin{aligned} D &= \left[ \frac{\partial E}{\partial v_x}, \frac{\partial E}{\partial v_y}, \frac{\partial E}{\partial v_{xx}}, \frac{\partial E}{\partial v_{xy}}, \frac{\partial E}{\partial v_{yx}}, \frac{\partial E}{\partial v_{yy}} \right] \\ D &= -2 \sum_{x=-w_x}^{w_x} \sum_{y=-w_y}^{w_y} (I(x) - J(Ax + d))D_2(x) \end{aligned} \quad (2.31)$$

$$D_2(x) = \left[ \frac{\partial J}{\partial v_x}, \frac{\partial J}{\partial v_y}, x \frac{\partial J}{\partial v_x}, y \frac{\partial J}{\partial v_x}, x \frac{\partial J}{\partial v_y}, y \frac{\partial J}{\partial v_y} \right]$$

$$D_{opt} = [0, 0, 0, 0, 0, 0]$$

A questo punto, il passo cruciale per ottimizzare l'obbiettivo è applicare l'approssimazione di Taylor sulla funzione  $J$ :

$$J(Ax + d) \sim J(x) + D_2(x) \cdot \bar{v}$$

$$\bar{v} = \begin{bmatrix} v_x, v_y, v_{xx}, v_{xy}, v_{yx}, v_{yy} \end{bmatrix} \quad (2.32)$$

Sostituendo quest'ultima sulla derivata dell'obbiettivo si ottiene:

$$D = -2 \sum_{x=-w_x}^{w_x} \sum_{y=-w_y}^{w_y} (\underbrace{I(x) - J(x)}_{\delta I(x)} - \underbrace{D_2(x) \bar{v}}_{\nabla I} \underbrace{D_2(x)}_{\nabla I}) \quad (2.33)$$

$\delta I(x)$  = Derivata temporale dell'immagine sorgente al punto  $x$ .

$$\nabla I \sim D_2(x)$$

*Osservazione 2* (Approssimazione di  $D_2$ ). La matrice  $D_2$  viene definita in (2.31) a partire dall'immagine di destinazione  $J$ .

Ciò nonostante, è possibile approssimare tale oggetto sfruttando anche l'immagine sorgente.

Tale processo implica un'errore numerico che diventa accettabile se e solamente se i punti target hanno subito una piccola variazione.

In tutti i casi, è stata scelta tale strategia per facilitare l'approccio iterativo che si andrà a raccontare nel seguito della trattazione.

Alla luce di quanto detto, è possibile rimaneggiare la derivata dell'obbiettivo nel seguente modo:

$$\frac{1}{2} D^T = \sum_{x=-w_x}^{w_x} \sum_{y=-w_y}^{w_y} \left( \underbrace{G}_{(\nabla I \cdot \nabla I)} \cdot \bar{v} - \underbrace{b}_{\nabla I \cdot \delta I} \right) \quad (2.34)$$

A questo punto, il passo conclusivo dell'ottimizzazione è il seguente:

$$\overline{v_{opt}} = G^{-1} \cdot b \quad (2.35)$$

L'espressione (2.35) vale se  $G$  è invertibile; dal punto di vista semantico significa che l'immagine sorgente deve contenere abbastanza informazioni derivate lungo entrambi gli assi.

In conclusione, la (2.35) permette di ottenere il vettore di coefficienti che parametrizza la matrice  $A$  e il vettore di traslazione  $d$  e che contestualmente minimizza l'obbiettivo (2.30).

È da sottolineare che l'algoritmo appena presentato ,a causa dell'approssimazione di Taylor, funziona bene esclusivamente quando i punti target si muovono con bassa velocità.

Per ottenere una maggior robustezza, è necessario iterare nello schema appena presentato.

Quest'ultimo approccio verrà presentato nella prossima sezione.

**Strategia Iterativa.** La logica di ottimizzazione appena raccontata viene integrata nella seguente strategia iterativa:

- Cicla fin quando non si verifica un criterio d'arresto

1. All'iterazione  $k : k \geq 1$  viene presentato un guess iniziale sia per il vettore di traslazione che per la matrice affine:

$$guess = [g^{k-1}, G^{k-1}] \quad (2.36)$$

2. Una volta ottenuto il guess, viene avviato il processo di ottimizzazione del seguente obiettivo in modo tale da ottenere i coefficienti per il vettore residuo di traslazione  $\eta^k = [\eta_x^k, \eta_y^k]$  e per la matrice residua affine  $\mathcal{M}^k = \begin{pmatrix} 1 + \eta_{xx}^k & \eta_{xy}^k \\ \eta_{yx}^k & 1 + \eta_{yy}^k \end{pmatrix}$

$$\begin{aligned} I(x) &= I(u + x) \\ J(x) &= u + G^{k-1}x + g^{k-1} \\ E(\eta^k, \mathcal{M}^k) &= \sum_{x=-w_x}^{w_x} \sum_{y=-w_y}^{w_y} (I(x) - J(\mathcal{M}^k x + \eta^k))^2 \end{aligned} \quad (2.37)$$

Per ottimizzare il suddetto obiettivo viene impiegata la stessa logica illustrata nella sezione precedente:

$$\begin{bmatrix} \eta_x^k \\ \eta_y^k \\ \eta_{xx}^k \\ \eta_{xy}^k \\ \eta_{yx}^k \\ \eta_{yy}^k \end{bmatrix} = G^{-1} \cdot b^k \quad (2.38)$$

$$b^k = \sum_{x=-w_x}^{w_x} \sum_{y=-w_y}^{w_y} \begin{bmatrix} I_x(x) \delta I^k(x) \\ I_y(x) \delta I^k(x) \\ x I_x(x) \delta I^k(x) \\ y I_x(x) \delta I^k(x) \\ x I_y(x) \delta I^k(x) \\ y I_y(x) \delta I^k(x) \end{bmatrix} \quad (2.39)$$

$$\delta I^k(x) = I(x) - J_k(x) \quad (2.40)$$

*Osservazione 3* (Aspetti Computazionali). Nello schema iterativo appena presentato, è utile sottolineare come la matrice  $G$  e le derivate lungo entrambi gli assi ( $I_x(x)$  e  $I_y(x)$ ) vengono calcolati una sola volta.

Invece, è necessario calcolare ad ogni iterazione il vettore che cattura il movimento residuo  $b^k$ .

3. Una volta terminato il processo di ottimizzazione viene generato un nuovo guess per la prossima iterazione:

$$\begin{aligned} g^k &= \eta^{k-1} + G^{k-1} \eta^k \\ G^k &= G^{k-1} \cdot \mathcal{M}^k \end{aligned} \quad (2.41)$$

### 2.1.3.3 Pipeline Generale dell'Algoritmo

L'algoritmo implementato sfrutta quanto detto sin'ora assieme ad altre strategie aggiunte dal sottoscritto per renderlo utile al task del tracking di bounding-box.

La pipeline dell'algoritmo viene mostrata di seguito:

- L'utente seleziona una bounding-box iniziale contenente il target.
- Per ogni frame successivo alla selezione:
  1. Estrai dalla **bounding-box Selezionata**, al frame precedente, una serie di keypoints sfruttando [13].
  2. Per ogni keypoint estratto: applica l'algoritmo di Lucas-Kanade descritto nelle sezioni precedenti.
  3. Ottenuti i keypoints aggiornati, calcola una media (lungo entrambi gli assi) dello scostamento dei keypoints rispetto alla **bounding-box Selezionata**.
  4. A questo punto, genera una Nuova Nounding-box sommando gli scostamenti medi lungo entrambi gli assi alla **Bounding-box Selezionata**.
  5. Fatto ciò, confronta la nuova bounding-box con una lista di bounding-box restituite da un detector al frame attuale.  
In particolar modo, il confronto implica una sogliatura su tre livelli (0.5, 0.6, 0.9) sfruttando la similarità di Jaccard.  
Se la sogliatura restituisce l'insieme vuoto, significa che il target è stato perso dal tracker.  
Altrimenti, viene costruita la **Nuova Bounding-Box Selezionata** prelevando la bounding-box più vicina alla Nuova Nounding-box dal punto di vista della Distanza Euclidea.

## 2.2 Multi-Object Tracking

### 2.2.1 Centroid Tracker

Il *Centroid Tracker* è un'algoritmo di tracking multi-oggetto che si basa fortemente sulle bounding-box restituite da un object-detector.

Detto ciò, l'algoritmo sfrutta la seguente logica per portare a termine i suoi scopi:

- Ad ogni frame vengono analizzate le bounding-box elaborate da un detector.
- **Track Iniziale.** All'avvio del processo di tracking, vengono registrate tutte le bounding-box correnti andando a salvare le seguenti informazioni per ognuna di esse: {Track ID, Centroide}.
- **Track NON Iniziale.** Per ogni bounding-box restituita dal detector:
  1. Estrapolo il Centroide dalla bounding-box.

- Calcolo e Salvo le distanze euclidee fra il centroide estrapolato al punto precedente e i centrodi appartenenti ai target già salvati.

Fatto ciò, aggiorno ogni target con la boundin-box avente il centroide più vicino dal punto di vista della distanza euclidea.

- Le eventuali bounding-box rimaste fuori dall'assegnazione presente al punto precedente vengono inizializzate come nuovi target.
- I target che non vengono matchati da nessuna ipotesi per un certo numero di frame verranno eliminati.

### 2.2.2 SORT

L'algoritmo SORT (Simple Online and Realtime Tracker) [14] è un algoritmo di tracking multi-oggetto che sfrutta l'approccio *tracking-by-detection*.

Inoltre, SORT si contraddistingue per l'impiego del *Filtro di Kalman* per il tracciamento delle bounding-box; nello specifico il filtro traccia le seguenti grandezze associate alle bounding-box:

- Posizione Lungo le Ascisse -  $x$ .
- Posizione Lungo le Ordinate -  $y$
- Area -  $A = \text{base} \times \text{altezza}$
- Aspect Ratio -  $\frac{\text{larghezza}}{\text{altezza}}$
- Derivata della Posizione Lungo le Ascisse -  $\frac{\partial x}{\partial t}$ .
- Derivata della Posizione Lungo le Ordinate -  $\frac{\partial y}{\partial t}$ .
- Derivata rispetto all'area -  $\frac{\partial A}{\partial t}$ .

Le caratteristiche salienti del suddetto algoritmo sono riassunte di seguito:

- L'algoritmo prende in input una lista di bounding-box nel formato:

$$[x, y, x_{max}, y_{max}, score] \quad (2.42)$$

- Per ogni tracker precedentemente registrato: Ottieni l'output sfruttando la fase di predizione del filtro di Kalman.
- Utilizzando l'Algoritmo Ungherese e la Similarità di Jaccard, effettua un *Assegnamento* fra le bounding-box restituite dai tracker al punto precedente (ammesso che fossero presenti) e le bounding-box restituite dal detector.

Una volta applicato l'algoritmo Ungherese, viene effettuata una sogliatura delle assegnazioni sfruttando una soglia pari a 0.3.

Le bounding-box associate ai tracker/detection che non hanno superato la soglia vengono inserite in delle rispettive liste.

Inoltre, anche le coppie tracker-detection che hanno superato la sogliatura vengono inserite in un'apposita lista.

*Osservazione 4.* Se al punto 2 non fosse presente alcun tracker, verranno inizializzati un numero di tracker pari al numero di bounding-box restituite dal detector.

- Per ogni tracker che ha assegnata una bounding-box, viene avviato l'aggiornamento del filtro di Kalman sfruttando la bounding-box stessa.
- Per ogni detection non assegnata ad alcun tracker, viene inizializzato un nuovo Tracker basato sul Filtro di Kalman.
- Inoltre, i tracker che non hanno ricevuto detection per  $z$  volte vengono cancellati dalla lista di tracker considerata dall'algoritmo.

# Capitolo 3

## Dataset

### 3.1 MOT Challenge

I dati impiegati per questo progetto sono forniti dal *MOT Benchmark* [2]. Quest'ultimo è un recente benchmark per il task del Multi-Object Tracking. Le caratteristiche salienti di tale benchmark sono riassunte di seguito:

- Viene presentato un set di metriche *standard* per valutare i metodi.
- Vengono fornite una serie di sequenze di addestramento, quest'ultime sono contraddistinte dalle detection e dagli *ID* di ground-truth.  
È da sottolineare come le traiettorie di ground-truth non siano mai frammentate.
- Vengono fornite una serie di sequenze di evaluation, quest'ultime si differenziano da quelle di addestramento per l'assenza degli *ID* di ground-truth.
- Alcune sequenze possiedono i file di calibrazione della camera.
- Le scene riprese contengono le seguenti variabili:
  1. Camera fissa o mobile
  2. Meteo Sereno o Nuvoloso
  3. Visuale dall'alto o in prima persona

### 3.2 Formato Dati

I dati allegati ad ogni sequenza video sono immagazzinati in un file CSV avente il seguente formato:

- Frame Number
- ID (se il file è di una sequenza di addestramento) altrimenti "-1"
- Posizione nelle Ascisse della Bounding-box
- Posizione nelle Ordinate della Bounding-box

- Larghezza della Bounding-box
- Altezza della Bounding-box
- Confidenza (sempre 1.0 per i dati di ground-truth)
- (3D) Posizione Asse *x* Bounding-Box
- (3D) Posizione Asse *y* Bounding-Box
- (3D) Posizione Asse *z* Bounding-Box

### 3.3 Dati Impiegati

I dati provengono dalle sequenze del 2015, del 2016 e del 2020.

Di seguito vengono mostrate una serie di tabelle che riassumono le caratteristiche salienti dei dati impiegati:

Nome	FPS	Risoluzione	Durata	Track	Boxes
ADL-Rundle-6	30	1920x1080	00:18	24	5009
ETH-Sunnyday	14	640x480	00:25	30	1858
PETS09-S2L1	7	768x576	01:54	19	4476

**Tabella 3.1:** Sequenze 2015

Nome	FPS	Risoluzione	Durata	Track	Boxes
MOT16-11	30	1920x1080	00:30	69	9174
MOT16-10	30	1920x1080	00:22	54	12318
MOT16-04	30	1920x1080	00:35	83	47557

**Tabella 3.2:** Sequenze 2016

Nome	FPS	Risoluzione	Durata	Track	Boxes
MOT20-02	25	1920x1080	01:51	296	202215
MOT20-03	25	1173x880	01:36	735	356728

**Tabella 3.3:** Sequenze 2020

# Capitolo 4

# Metriche di Valutazione

Come accennato in precedenza, la scopo di questo progetto è *confrontare* una serie di algoritmi di tracking.

Per portare a termine tale confronto, vi è la necessità di utilizzare alcune misure di valutazione.

In questa sede, sono state sfruttate le metriche presenti nel MOT Benchmark [2].

Nello specifico, le metriche per i Multi-Object Tracker sono prese nella loro interezza dal suddetto Benchmark.

Invece, le metriche per i Single-Object Tracker sono frutto di un riadattamento delle metriche di [2] al caso del tracking ad oggetti singoli.

## 4.1 Metriche Generali

### 4.1.1 Metriche di Base

Il primo passo per valutare un tracker ,indipendentemente dal numero di oggetti che traccia, è determinare se un'*ipotesi* generata da quest'ultimo è corretta o meno.

Nello specifico, gli errori che può commettere un tracker ,su di una data ipotesi, sono due: Falso Positivo (rilevazione erronea) o Falso Negativo (mancata rilevazione).

Di seguito sono presentate le metriche *di base* utilizzate per tutti gli algoritmi di Tracking del progetto:

$$True\_Positive(box, gt) = \begin{cases} \textbf{SI} & jaccard(box, gt) \geq 0,5 \\ \textbf{NO} & \text{altrimenti} \end{cases}$$

$$False\_Positive(box, gt) = \begin{cases} \textbf{SI} & jaccard(box, gt) < 0,5 \\ \textbf{NO} & \text{altrimenti} \end{cases}$$

$$False\_Negative(track, ground\_truth) = \{\textbf{ground\_truth} \neq \emptyset \wedge \textbf{track} = \emptyset\} \quad (4.1)$$

### 4.1.2 Metriche Complesse

Le metriche *Complesse* sfruttano quelle di base per definire delle nuove metriche che hanno come raggio di azione l'intera traiettoria del un target.

Nello specifico, sono state impiegate le seguenti metriche:

$$\begin{aligned}
 False\_Positive\_Rate &= \frac{\sum_{i=1}^{num\_false\_pos} \mathbf{1}}{\text{num\_frame\_tracking}} \\
 Precision &= \frac{\text{TP\_filmato}}{\text{TP\_filmato} + \text{FP\_filmato}} \\
 Recall &= \frac{\text{TP\_filmato}}{\text{TP\_filmato} + \text{FN\_filmato}} \\
 OTP &= \frac{\sum_{i=start\_track}^{end\_track} \text{jaccard}(\text{box}_i^{(TP)}, \text{gt}_i)}{\sum_{i=start\_track}^{end\_track} \text{match\_number}_i}
 \end{aligned} \tag{4.2}$$

La *Precision* quantifica la capacità del tracker di individuare i target correttamente.

La *Recall* quantifica le capacità dei tracker nella rilevazione dei target.

La *Object-Tracking Precision (OTP)* misura la *precisione* di localizzazione inerentemente alle ipotesi correttamente assegnate ai target.

## 4.2 Single-Object Tracking

Le metriche appositamente pensate per i Single-Object Tracker sono elencate di seguito:

$$SOTA = \mathbf{1} - \left[ \frac{\sum_{i=start\_track}^{end\_track} \mathbf{FN}_i + \mathbf{FP}_i}{\text{num\_frame\_track}} \right] \tag{4.3}$$

**Algoritmo 4.1:** Track Consistency (Single-Target)

```

1 def track_consistency (video, tracker, ground_truth, target):
2
3     target_track      = []
4     target_lost       = []
5
6     last_status       = None
7     fragment_counter = 0
8
9     n_frame_track    = ground_truth.target_interval(target)
10    start_frame      = ground_truth.start_frame (target)
11    video.select (start_frame)
12
13    for frame in video:
14        status          = tracker.update(frame)
15
16        if (status == "Lost"):
17            target_lost.append(frame.number)
18            last_status = "Lost"
19        else:
20            target_track.append(frame.number)
21
22        if (last_status == "Lost"):
23            fragment_counter += 1
24            last_status       = None
25
26
27    T =  $\frac{\text{target\_track\_LENGTH}}{\text{n\_frame\_track}} \cdot 100$ 
28
29    L =  $\frac{\text{target\_lost\_LENGTH}}{\text{n\_frame\_track}} \cdot 100$ 
30
31    return (T , L , fragment_counter)

```

La *Single-Object Tracking Precision (SOTA)* è una metrica costruita dal sottoscritto prendendo ispirazione dalla metrica *MOTA (Multi-Object Tracking Precision)* presente nel MOT Benchmark.

Sostanzialmente, SOTA è una fra le metriche più riassuntive inerentemente le performance di un tracker.

Un'ulteriore metrica impiegata è la *Track Consistency*, quest'ultima valutata sia la costanza di tracciamento dei target che il numero di *buchi* nelle traiettorie generate.

Nel dettaglio, un target viene considerato *Molto Tracciato* se viene seguito dal tracker per almeno l'80 % della sua vita.

Invece, un target viene considerato *Poco Tracciato* se viene seguito dal tracker per non più del 20 % della sua vita.

### 4.3 Multi-Object Tracking

$$IDSW = \{gt_i^{(t)} \rightarrow box_j^{(t)} \wedge gt_i^{(t+1)} \rightarrow box_k^{(t+1)} \quad k \neq j\} \quad (4.4)$$

$$total\_IDSW = \sum_{i=start\_track}^{end\_track} IDSW_i$$

$$weighted\_IDSW = \frac{total\_IDSW}{\text{Recall}} \quad (4.5)$$

$$MOTA = 1 - \left[ \frac{\sum_{i=start\_track}^{end\_track} FN_i + FP_i + IDSW_i}{\text{num\_target\_track}} \right]$$

**Algoritmo 4.2:** Track Consistency (Multi-Target)

```

1 def track_consistency (video, tracker, ground_truth):
2
3     target_track      = []
4     target_lost       = []
5
6     last_status       = {}
7     fragment_counter = 0
8
9     n_frame_track    = ground_truth.target_interval()
10    start_frame      = ground_truth.start_frame ()
11    video.select (start_frame)
12
13    for frame in video:
14        trackers       = tracker.update(frame)
15
16        for tk in trackers:
17            status = tk.status
18
19            if (status == "Lost"):
20                target_lost.append(frame.number)
21                last_status[tk.ID] = "Lost"
22            else:
23                target_track.append(frame.number)
24
25            if (last_status[tk.ID] == "Lost"):
26                fragment_counter += 1
27                last_status[tk.ID] = None
28
29
30    T = target_track_LENGTH / n_frame_track * 100
31
32    L = target_lost_LENGTH / n_frame_track * 100
33
34    return (T , L , fragment_counter)

```

Per le metriche Track Consistency e MOTA, valgono le stesse considerazioni fatte per i Single-Object Tracker.

Invece, il conteggio degli ID Switch (*IDSW*) merita un discorso a parte.

Nello specifico, un ID Switch avviene quando un target subisce un cambiamento di *Identificativo* fra due computazioni in tempi diversi dell'algoritmo di tracking.

Tale fenomeno, è chiaramente da minimizzare.

Resta il fatto che è *inevitabile* che al crescere della Recall aumentino gli ID Switch; per tanto ,come proposto dagli autori del benchmark, è stata impiegata anche la metrica *Weighted ID Switch*.

# Capitolo 5

## Esperimenti Quantitativi

Nella sezione corrente verranno confrontate le performance degli algoritmi proposti.

Il confronto impiegherà le metriche illustrate nei capitoli precedenti ed è diviso in due fasi distinte e separate:

- Confronto fra i Single-Object Tracker (CSRT, TLD, Lucas-Kanade Tracker).
- Confronto fra i Multi-Object Tracker (SORT, Centroid Tracker).

Di seguito vengono riassunte le caratteristiche salienti dei dati impiegati per la comparativa:

Nome	FPS	Risoluzione	Durata	Track	Boxes
ADL-Rundle-6	30	1920x1080	00:18	24	5009
ETH-Sunnyday	14	640x480	00:25	30	1858
PETS09-S2L1	7	768x576	01:54	19	4476
MOT16-11	30	1920x1080	00:30	69	9174
MOT16-10	30	1920x1080	00:22	54	12318
MOT16-04	30	1920x1080	00:35	83	47557
MOT20-02	25	1920x1080	01:51	296	202215
MOT20-03	25	1173x880	01:36	735	356728

**Tabella 5.1:** Dati di Test

Inoltre, è di utilità sottolineare che i valori di ogni metrica mostrati di seguito sono frutto della media aritmetica di quest'ultima su tutti i video di test.

Infine, è importante illustrare ulteriormente le metodologie di test per i Single-Object Tracker.

Nello specifico, per ogni Video di Test vengono portate a termine le seguenti azioni:

- Vengono scelti 3 Target da tracciare per tutti gli algoritmi del lotto (CSRT, LK Tracker, TLD).

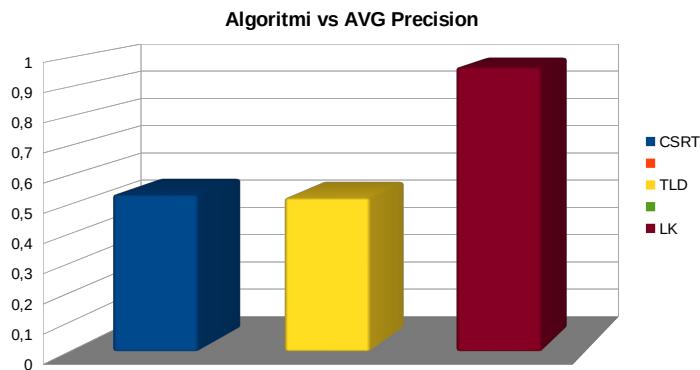
- Per ogni coppia (Algoritmo, Target) vengono valutate le performance secondo le metriche proposte.
- Infine, per ogni algoritmo e per ogni metrica: viene fatta una Media Aritmetica sulle misure ottenute sui Target.

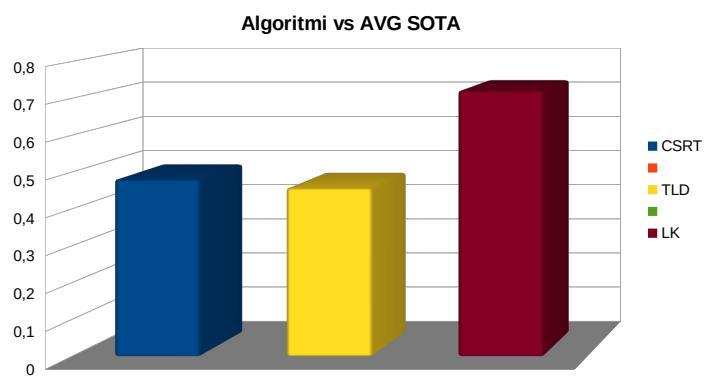
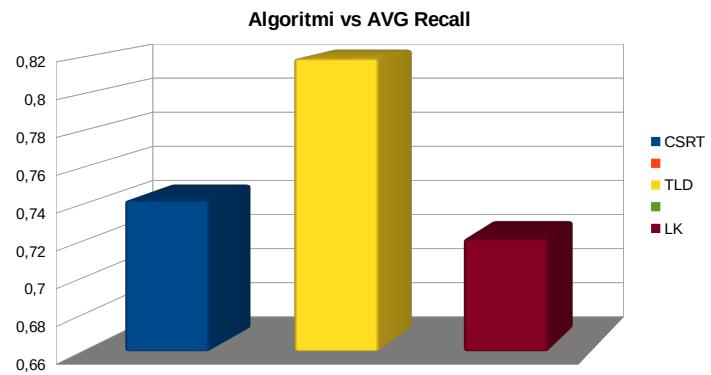
Chiaramente, per ogni algoritmo e per ogni metrica verrà effettuata anche la media su tutti i video di test.

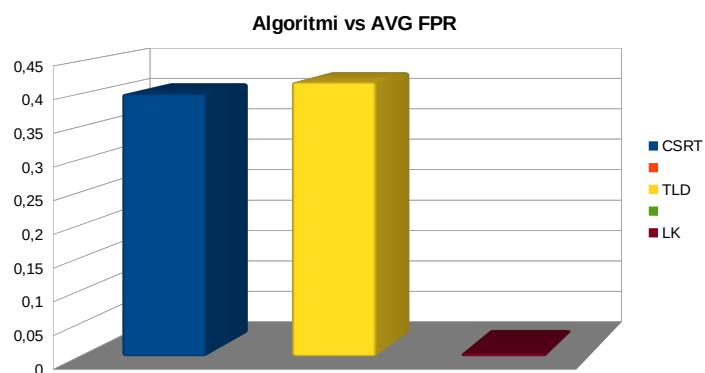
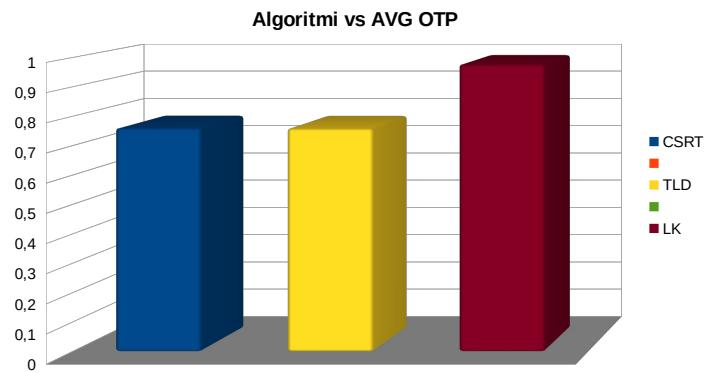
## 5.1 Single-Object Tracking

La sezione corrente si prefigge l'obbiettivo di mostrare le performance dei Single-Object Tracker proposti.

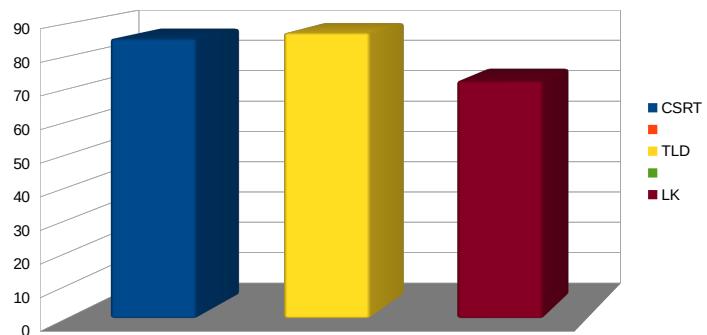
Di seguito, vengono mostrati una serie di grafici che evidenziano la *performance Media su Tutti i Video di Test* dei suddetti algoritmi:



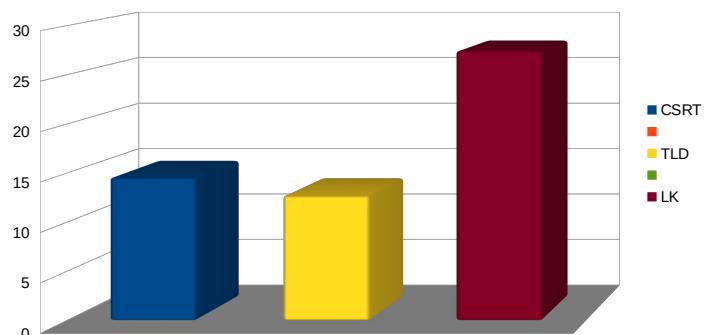


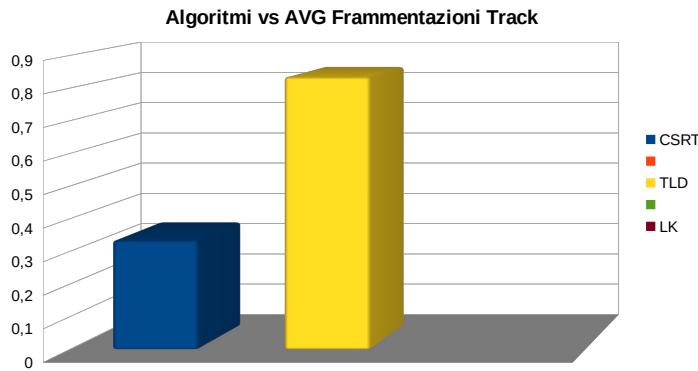


**Algoritmi vs AVG % Track**



**Algoritmi vs % Lost**





I grafici mostrati portano alle seguenti conclusioni:

- L'algoritmo Lucas-Kanade porta in dote un eccellente valore di Precision ed un discreto valore di Recall.

Questo significa che i target rilevati saranno con elevate probabilità corretti.

Nonostante ciò, il discreto valore di Recall evidenzia alcune difficoltà nella rilevazione dei target.

- Gli algoritmi TLD e CSRT esibiscono un comportamento simile e per tanto verranno analizzati insieme.

È da sottolineare che entrambi gli algoritmi sfruttano le caratteristiche visive dell'immagine anziché impiegare le informazioni restituite da un object-detector.

Tutto ciò, unito al copioso clutter presente nei filmati di test, induce delle prestazioni di Precision inferiori rispetto al Lucas-Kanade Tracker.

Dal punto di vista della Recall, l'algoritmo TLD esibisce delle performance superiori rispetto agli altri algoritmi.

Tale prerogativa era preventivabile dato che è stato impiegato un sistema di learning che analizza l'intera immagine.

A tal proposito, è interessante sottolineare che il valore di Recall dell'algoritmo CSRT è sensibilmente inferiore rispetto a quello di TLD.

Quanto detto, lascia intuire che le feature e il processo di learning di TLD sono sensibilmente più efficaci dal punto di vista della rilevazione dei target.

- Dal punto di vista della precisione di localizzazione (OTP), c'è da dire che tutti gli algoritmi si comportano bene.

Infatti, il risultato minimo di Object Tracking Accuracy si attesta su 0,74.

- Dal punto di vista della frammentazione della traiettoria generata, nessun algoritmo mostra frammentazioni di traiettoria copiose.

Nonostante ciò, l'LK Tracker spicca fra tutti per l'assenza di frammentazione su tutti i video di test.

- Dal punto di vista della consistenza del Track, c'è da dire che l'LK Tracker offre una percentuale media di Track del 70 % e una di Lost del 35 %.

Gli altri due algoritmi del lotto offrono una Consistenza di Track Simile fra loro e superiore rispetto al Lucas-Kanade Tracker

Quanto detto lascia intuire una difficoltà da parte del Lucas-Kanade Tracker nel mantenere il track di un target quando avviene un'occlusione marcata oppure quando quest'ultimo esce e rientra dal campo visivo della camera.

Tale risultato è riconducibile all'assenza di un meccanismo di *reidentification* nell'implementazione dell'LK Tracker proposta.

In realtà, neanche TLD e CSRT possiedono un meccanismo esplicito di *reidentification*.

Ciò nonostante, entrambi possono contare su un processo di *online-learning* che gli permette di individuare più facilmente il target nella scena; compreso il caso in cui quest'ultimo esce e rientra dal campo visivo della camera.

In conclusione di questa sezione, si andrà ad analizzare la metrica SOTA (Single-Object Tracking Accuracy) restituita dagli algoritmi testati; tale metrica racchiude un pò tutte le altre e per tale motivo è stata scelta per le considerazioni finali.

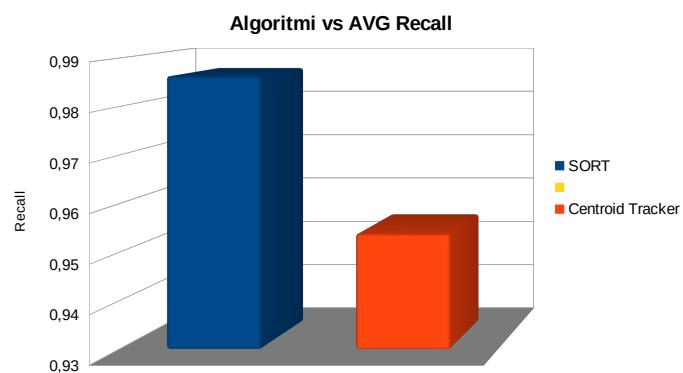
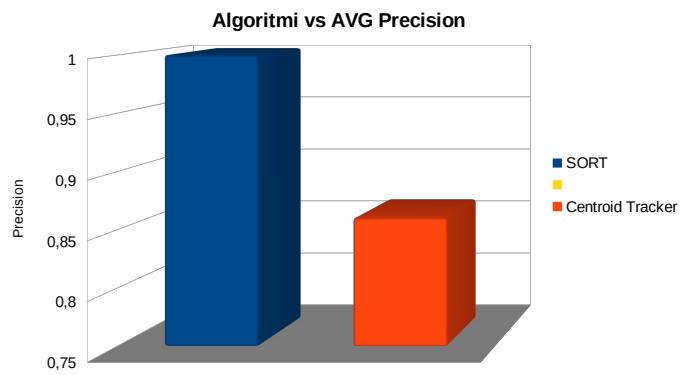
A tal proposito, la metrica SOTA evidenzia una superiorità generale dell'algoritmo di Lucas e Kanade.

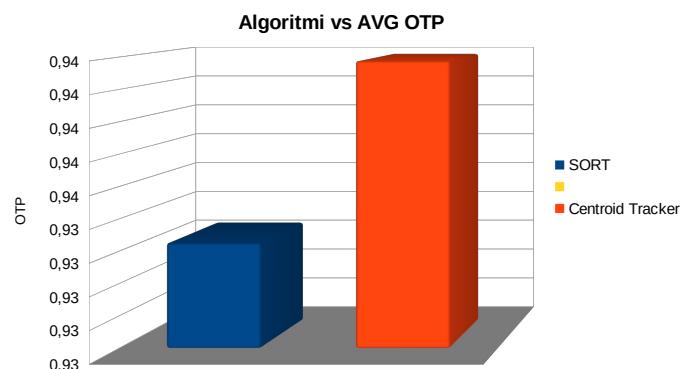
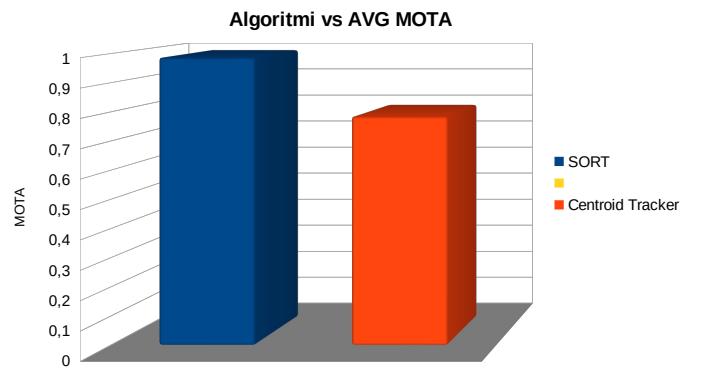
Invece, gli algoritmi TLD e CSRT hanno risultati simili reciprocamente e inferiori rispetto all'LK Tracker.

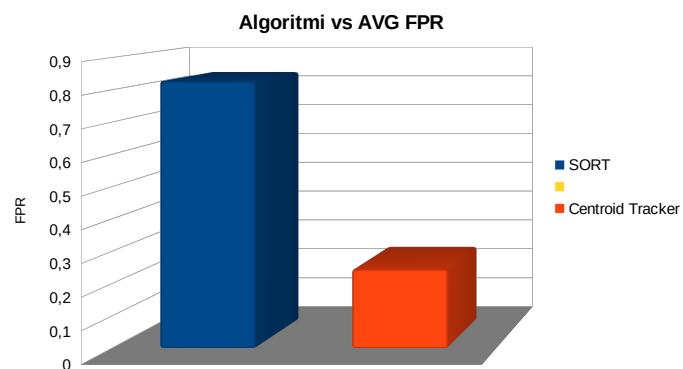
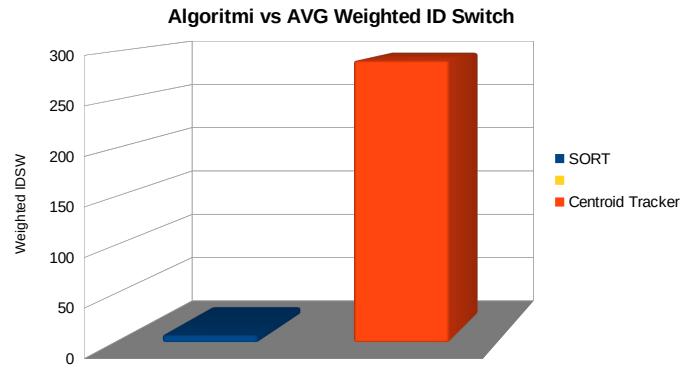
## 5.2 Multi-Object Tracking

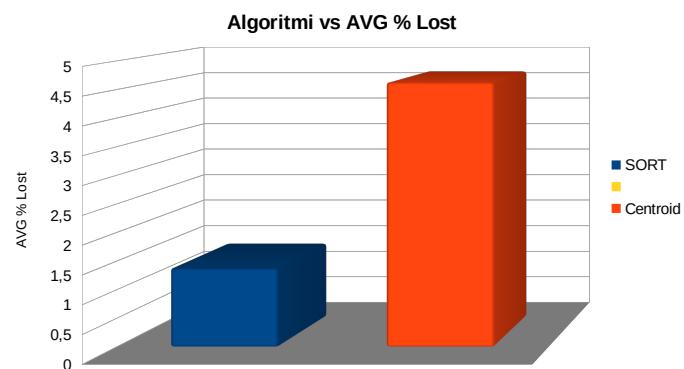
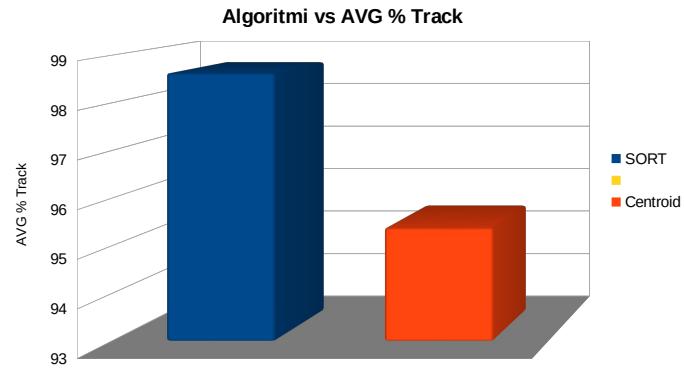
La sezione corrente si prefigge l'obiettivo di mostrare le performance dei Multi-Object Tracker proposti.

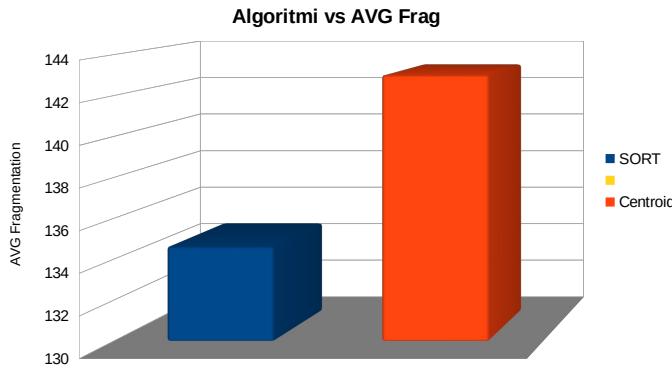
Di seguito, vengono mostrati una serie di grafici che evidenziano la *performance Media su Tutti i Video di Test* dei suddetti algoritmi:











I grafici mostrati portano alle seguenti conclusioni:

- Per entrambi gli algoritmi, i valori di Precision e Recall si attestano a dei valori ottimi.

Inoltre, è da evidenziare le prestazioni *prossime alla perfezione* dell'algoritmo SORT.

- Dal punto di vista della precisione di localizzazione, entrambi gli algoritmi si comportano bene.

Inoltre, nessuno dei due mostra una superiorità marcata rispetto all'altro.

- Dal punto di vista della precisione di localizzazione (OTP), c'è da dire che tutti gli algoritmi si comportano bene.

Infatti, il risultato minimo di Object Tracking Accuracy si attesta su 0,93.

- Dal punto di vista degli ID Switch, il Centroid Tracker fa abbastanza peggio rispetto a SORT.

Tutto ciò è riconducibile all'impiego della strategia nearest neighbors da parte del Centroid Tracker.

Del resto, tale euristica ,in presenza di occlusione marcata, può portare ad un copioso fenomeno di ID Switch.

- I valori restituiti per la metrica FPR lasciano trasparire una superiorità da parte del Centroid Tracker.

Del resto, l'algoritmo SORT impiega un Kalman Filter per stimare la nuova posizione di ogni target.

Tale stima è un processo statistico e per tanto è più prone ad errori rispetto alla strategia *1-NN* impiegata dal Centroid Tracker.

- Dal punto di vista della consistenza del Track, entrambi gli algoritmi si comportano bene.

Discorso analogo per la Frammentazione delle traiettorie.

In conclusione di questa sezione, si andrà ad analizzare la metrica MOTA (Multi-Object Tracking Accuracy) restituita dagli algoritmi testati.

La suddetta metrica *racchiude* un pò tutte le altre e per tale motivo è stata scelta per le considerazioni finali.

A tal proposito, la metrica MOTA evidenzia una superiorità dell'algoritmo SORT rispetto al Centroid Tracker.

Tale superiorità si può attribuire alla maggiore capacità garantita dal Filtro di Kalman rispetto alla strategia 1 – *NN* del Centroid Tracker.

# Capitolo 6

## Esperimenti Qualitativi

L'ultimo tassello del progetto svolto è stato la costruzione di un tool *Grafico* con l'obbiettivo di mostrare in maniera *visiva* le prestazioni degli algoritmo confrontati.

Nello specifico, il tool costruito esibisce le seguenti caratteristiche:

- Player Video con Bottoni Play, Stop, Reset e Applicazione Algoritmi di Tracking.
- Selettore Algoritmo di Tracking.
- Selettore della ROI per i Single-Object Tracker.
- Ogni ipotesi restituita dai tracker avrà assegnata una bounding-box di colore univoco in modo tale da poter apprezzare in maniera *visiva* il processo di tracking.
- Le detection impiegate per le bounding-box sono quelle fornite dal MOT Benchmark.

Tale scelta ha ristretto i filmati impiegati a quelli del benchmark.

Chiaramente, per un tool di produzione sarà necessario addestrare un object-detector, ad esempio *Faster R-CNN* [15].

Per la costruzione del Tool, è stato impiegato il linguaggio di programmazione Python e la libreria grafica *wxPython*.

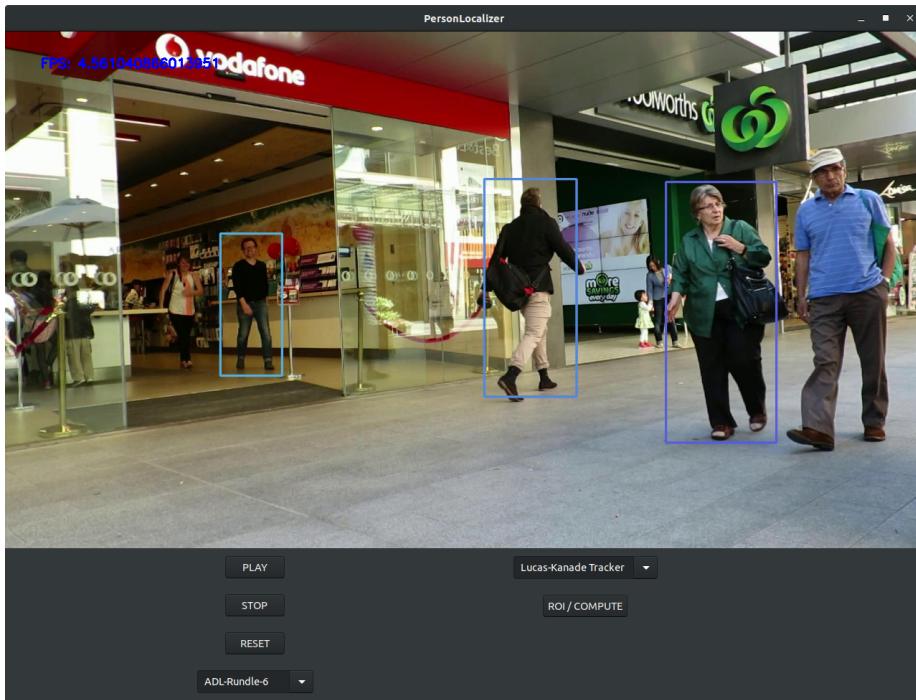
Inoltre, gli algoritmi disponibili sono i seguenti: Lucas-Kanade Tracker, CSRT Tracker, SORT, Centroid Tracker.

Come si può apprezzare, manca l'algoritmo TLD; quest'ultima prerogativa è stata dettata dal linguaggio di programmazione impiegato per il tool.

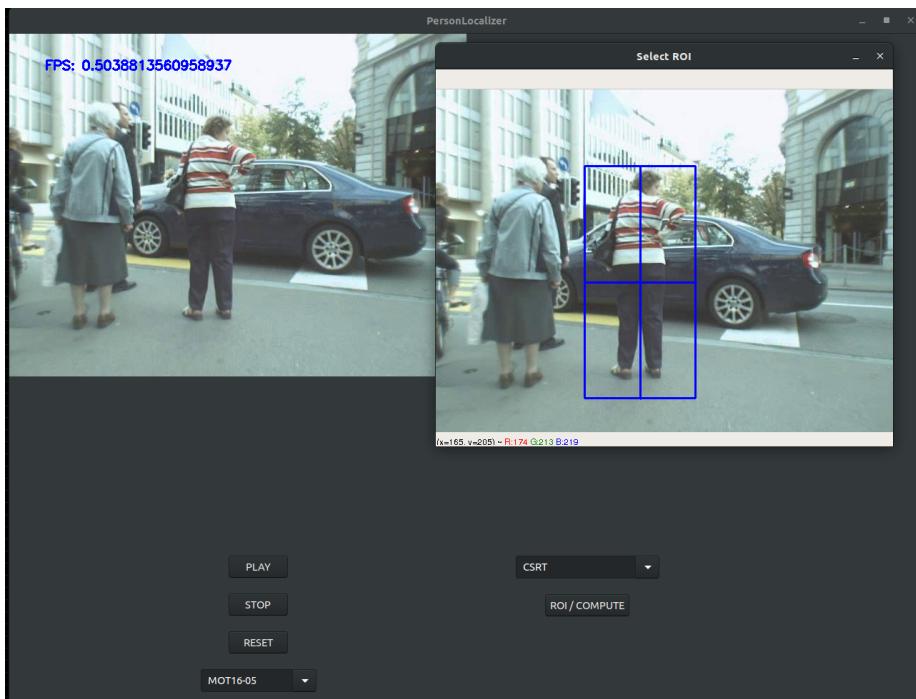
Del resto, l'implementazione proposta per TLD è stata scritta in C++, mentre le altre sono state scritte in Python.

Per inciso, la necessità del C++ per TLD è stata dettata dalla maggior richiesta di efficienza computazionale caratterizzante il suddetto algoritmo.

Nel resto del capitolo, vengono mostrati una serie di screenshot del tool sviluppato:



**Figura 6.1:** Tracciamento di tre persone sfruttando l'LK Tracker e il tool costruito.



**Figura 6.2:** Per i Single-Object Tracker, è prevista la selezione del target attraverso un'apposita GUI.

# Capitolo 7

## Conclusione

In conclusione di questo progetto, posso dire di aver approfondito una fra le tematiche più interessanti e degne di nota della Computer Vision.

Inoltre, l'esperienza fatta lascia trasparire una tematica trasversale a tutti i test svolti dal sottoscritto: le prestazioni di Runtime.

In effetti, tutti gli algoritmi che hanno impiegato l'intera immagine (o parti di essa) come input hanno restituito dei valori di FPS davvero bassi per poter essere utilizzati in Real-Time.

Per inciso, tale problematica è stata appresa durante la costruzione del tool grafico.

Per quanto detto, se il focus è sulle prestazioni in real-time è di utilità evitare algoritmi come CSRT, Lucas-Kanade e TLD.

Il prezzo da pagare è una perdita di generalità della soluzione, soprattutto rispetto a TLD e CSRT.

Del resto, SORT, Lucas-Kanade e Centroid Tracker hanno bisogno delle bounding-box restituite da un object-detector, che nella maggior parte dei casi deve essere addestrato su una quantità di dati che per forza di cose deve avere un set di etichette finito e ben congegnato.

Tale prerogativa non è presente in TLD e CSRT, infatti tali algoritmi, attraverso il loro processo di online-learning, permettono di tracciare oggetti sconosciuti in un filmato.

# Bibliografia

- [1] E.Maggio, A.Cavallaro - Video Tracking Theory and Practice
- [2] Leal-Taixe et. al. - MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking
- [3] Kalal et. al. - Tracking-Learning-Detection
- [4] Kalai et. al. - P-N Learning: Bootstrapping Binary Classifiers by Structural Constraints
- [5] Fua et. al. - Fast Keypoint Recognition in Ten Lines of Code
- [6] Henriques et. al. - High-Speed Tracking with Kernelized Correlation Filters
- [7] Kalai et. al. - Forward-Backward Error: Automatic Detection of Tracking Failures
- [8] Matas et. al. - Discriminative Correlation Filter Tracker with Channel and Spatial Reliability
- [9] Galoogahi et. al. - Correlation Filters with Limited Boundaries
- [10] Dipiropoulos et. al. - A Spatially Constrained Generative Model and an EM Algorithm for Image Segmentation
- [11] Bouguet - Pyramidal Implementation of the Lucas Kanade Feature Tracker  
Description of the algorithm
- [12] Lucas, Kanade - An Iterative Image Registration Technique with an Application to Stereo Vision
- [13] Shi et. al. - Good features to Track
- [14] Bewley et. al. - Simple Online And Realtime Tracking
- [15] Sun et. al. - Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks