



**Università
di Catania**

Università degli Studi di Catania

Corso di Laurea Magistrale in Informatica

Sistema di Tracciamento Oggetti in Video con Analisi mediante Metriche Full-Reference

Progetto di Multimedia e Laboratorio

Studente:

Michele Guglielmino
Matricola:1000031551

Docente:

Prof. Dario Allegra
Prof. Filippo Stanco

Anno Accademico 2025/2026

Indice

1	Introduzione	4
1.1	Contesto e Motivazioni	4
1.2	Obiettivi del Progetto	4
2	Riferimenti Teorici	5
2.1	Il Movimento nei Video	5
2.1.1	Rappresentazione del Movimento	5
2.1.2	Background Subtraction e MOG2	5
2.2	Tracciamento Oggetti	6
2.2.1	Tracciamento basato su Distanza Euclidea	6
2.2.2	Feature-based Tracking con ORB	6
2.3	Stabilizzazione Video	7
2.3.1	Stabilizzazione Feature-based	7
2.3.2	Trasformazione Affine	8
2.4	Operazioni Morfologiche	8
2.4.1	Closing (Chiusura)	8
2.4.2	Opening (Apertura)	8
2.5	Non-Maximum Suppression (NMS)	8
2.6	Coordinate Smoothing	9
2.7	Metriche di Qualità per Video	9
2.7.1	Mean Squared Error (MSE)	9
2.7.2	Peak Signal-to-Noise Ratio (PSNR)	10
2.7.3	Structural Similarity Index (SSIM)	10
3	Architettura del Sistema	11
3.1	Panoramica dell'Architettura	11
3.2	Dual-Pipeline Architecture	11
3.3	Pipeline Principale	12
3.3.1	Fase 1: Inizializzazione e Configurazione	12
3.3.2	Fase 2: Selezione della Region of Interest (ROI)	12
3.3.3	Fase 3: Inizializzazione Background Subtractor	13
3.3.4	Fase 3.5: Kernel Morfologico	13
3.3.5	Fase 4: Loop di Elaborazione Fotogrammi	14
3.3.6	Fase 5: Rilevamento Contorni, NMS e Tracking Dual-Pipeline	16
3.3.7	Fase 7: Applicazione Algoritmi e Memorizzazione	17
3.3.8	Fase 8: Visualizzazione Real-Time Dual-Pipeline	18
3.3.9	Fase 9: Calcolo Metriche di Qualità	19
4	Algoritmo di Tracking	20
4.1	Classe Tracciamento	20
4.1.1	Struttura della Classe	20
4.2	Metodo di Aggiornamento con Feature Matching	21
4.3	Coordinate Smoothing	23
4.4	Gestione Tracce Obsolete	23
4.5	Non-Maximum Suppression (NMS)	24

4.6	Limitazioni e Miglioramenti	25
4.6.1	Limitazioni	25
5	Algoritmi di Valutazione della Qualità	26
5.1	Video Stabilizer	26
5.1.1	Architettura della Classe	26
5.2	Moving Average Spatial Filter	28
5.3	Compressione JPEG	29
5.3.1	Implementazione	29
5.4	Sfocatura Gaussiana	29
5.4.1	Implementazione	30
5.5	Unsharp Masking	30
5.5.1	Implementazione	31
5.6	Conclusioni	31
6	Calcolo Metriche di Qualità	32
6.1	Architettura della Funzione	32
6.1.1	Validazione Input	32
6.2	Calcolo MSE	33
6.3	Calcolo PSNR	33
6.4	Calcolo SSIM	33
6.5	Restituzione Risultati	34
7	Risultati Sperimentali	34
7.1	Setup Sperimentale	34
7.1.1	Configurazione Parametri	34
7.2	Tracciamento Oggetti: Confronto Dual-Pipeline	35
7.2.1	Risultati Quantitativi su traffico1.mp4	35
7.2.2	Analisi Qualitativa: Riduzione Falsi Positivi	35
7.2.3	Analisi Maschere MOG2: Riduzione Rumore	36
7.2.4	Conclusioni Comparative	37
7.2.5	Risultati Quantitativi su traffico2.mp4	38
7.2.6	Analisi Visiva: Impatto Camera Shake	38
7.2.7	Confronto Maschere MOG2: Efficacia Stabilizzazione + Blurring	39
7.2.8	Confronto Comparativo traffico1 vs traffico2	40
7.3	Metriche Qualitative	41
7.3.1	Risultati traffico1.mp4 (1280x720)	41
7.3.2	Risultati traffico2.mp4 (1920x1080)	41
7.3.3	Analisi Metriche - traffico1.mp4	41
7.3.4	Analisi Metriche - traffico2.mp4	42
7.4	Conclusioni	44

Sommario

Il presente progetto si inserisce nell'ambito dell'elaborazione e analisi di contenuti multimediali, concentrandosi specificamente sul tracciamento automatico di oggetti in movimento all'interno di sequenze video e sulla valutazione quantitativa della qualità delle trasformazioni applicate. L'obiettivo principale consiste nello sviluppo di un sistema integrato capace di identificare, seguire e contrassegnare entità dinamiche presenti nei fotogrammi mediante l'impiego di algoritmi di *background subtraction* basati su modelli statistici, e di valutare oggettivamente l'impatto di diverse operazioni di post-processing attraverso l'applicazione di metriche *full-reference*.

Il sistema implementato si articola in quattro moduli funzionali principali: un modulo di orchestrazione del flusso video (`main.py`), un algoritmo di tracciamento basato su distanza euclidea (`tracciamento.py`), un insieme di filtri di post-processing (`filtri.py`), e un modulo di calcolo metriche qualitative (`Calcolo_metriche.py`). L'architettura modulare adottata consente una chiara separazione delle responsabilità e facilita l'estensibilità del sistema.

La metodologia di tracciamento si basa sull'algoritmo MOG2 (*Mixture of Gaussians 2*) per la segmentazione dinamica dello sfondo, seguito da un'associazione temporale degli oggetti rilevati mediante calcolo della distanza euclidea tra i centroidi. Le metriche di qualità implementate includono MSE (*Mean Squared Error*), PSNR (*Peak Signal-to-Noise Ratio*) e SSIM (*Structural Similarity Index*), calcolate confrontando l'intera sequenza video originale con le sequenze risultanti dall'applicazione di tre differenti algoritmi di elaborazione: compressione JPEG lossy, sfocatura gaussiana e sharpening mediante unsharp masking.

I risultati ottenuti dimostrano l'efficacia del sistema nel tracciamento accurato di oggetti multipli in scenari dinamici, nonché la capacità delle metriche implementate di quantificare oggettivamente le alterazioni introdotte dai diversi algoritmi di elaborazione. L'analisi comparativa mostra come algoritmi di smoothing conservino maggiormente la similarità strutturale rispetto a trasformazioni aggressive come la compressione lossy, e come lo sharpening possa migliorare la nitidezza percepita pur alterando il segnale originale.

1 Introduzione

1.1 Contesto e Motivazioni

L'elaborazione di contenuti video rappresenta uno dei pilastri fondamentali dell'informatica moderna, con applicazioni che spaziano dalla sorveglianza automatizzata alla realtà aumentata, dall'analisi sportiva ai sistemi avanzati di assistenza alla guida. Nel contesto dell'era digitale, la capacità di estrarre informazioni semantiche ad alto livello da sequenze di fotogrammi costituisce una competenza essenziale per numerosi domini applicativi.

Il movimento rappresenta una caratteristica intrinseca e distintiva dei contenuti video, differenziandoli dalle immagini statiche. La capacità di rilevare, tracciare e analizzare entità in movimento fornisce informazioni preziose relative alla dinamica della scena, consentendo di implementare sistemi intelligenti capaci di reagire agli eventi osservati. Applicazioni pratiche includono il monitoraggio del traffico veicolare, il riconoscimento di azioni umane, la videosorveglianza intelligente, e l'analisi comportamentale.

Parallelamente, la valutazione oggettiva della qualità video assume rilevanza crescente in contesti dove le trasformazioni applicate ai contenuti multimediali devono essere quantificate e validate. Operazioni di filtraggio, compressione, restauro e miglioramento possono introdurre degradazioni percettive che necessitano di essere misurate mediante metriche standardizzate. Le metriche *full-reference* permettono confronti diretti tra contenuti originali e trasformati, fornendo indicatori numerici della fedeltà e della similarità strutturale.

1.2 Obiettivi del Progetto

Il progetto persegue i seguenti obiettivi principali:

1. **Implementazione di un sistema di tracciamento robusto:** sviluppare un algoritmo capace di identificare e seguire automaticamente oggetti multipli in movimento attraverso sequenze video, assegnando identificatori univoci e persistenti nel tempo.
2. **Applicazione di trasformazioni di post-processing:** implementare una collezione di filtri rappresentativi delle principali categorie di operazioni di elaborazione video.
3. **Valutazione quantitativa mediante metriche oggettive:** calcolare metriche standard (MSE, PSNR, SSIM) per confrontare sistematicamente le sequenze originali con quelle trasformate, quantificando l'impatto di ciascun filtro applicato.
4. **Sperimentazione su contenuti reali:** applicare il sistema sviluppato a video realistici contenenti oggetti in movimento, validando l'efficacia delle tecniche implementate in scenari pratici.

2 Riferimenti Teorici

2.1 Il Movimento nei Video

I video digitali rappresentano sequenze temporali di immagini bidimensionali (fotogrammi) acquisite a frequenze standard (tipicamente 24, 25, 30 o 60 fotogrammi al secondo). Il movimento percepito dal sistema visivo umano durante la riproduzione video emerge dalla rapida successione di fotogrammi leggermente differenti, fenomeno noto come persistenza retinica.

2.1.1 Rappresentazione del Movimento

Quando si analizza il movimento in ambito computazionale, è necessario distinguere tra movimento tridimensionale reale nella scena e movimento bidimensionale proiettato sul piano immagine. La proiezione prospettica di un punto $P = (X, Y, Z)$ nello spazio tridimensionale sul piano immagine con coordinate (x, y) è governata dalle relazioni:

$$x = F \frac{X}{Z}, \quad y = F \frac{Y}{Z} \quad (1)$$

dove F rappresenta la lunghezza focale della telecamera e Z la profondità del punto rispetto al centro ottico.

La differenza tra la posizione di un punto \mathbf{x} nel fotogramma al tempo t e la sua posizione nel fotogramma al tempo $t + \Delta t$ definisce il **vettore di movimento** (motion vector):

$$\mathbf{d}(\mathbf{x}) = \mathbf{x}_{t+\Delta t} - \mathbf{x}_t \quad (2)$$

L'insieme dei vettori di movimento per tutti i punti dell'immagine costituisce il **campo di movimento** (motion field). In pratica, si adottano rappresentazioni semplificate del campo di movimento:

- **Rappresentazione basata su blocchi:** l'immagine viene suddivisa in blocchi regolari (tipicamente 8×8 o 16×16 pixel) e si assume che tutti i pixel di un blocco condividano lo stesso vettore di movimento.
- **Rappresentazione basata su regioni:** si segmenta l'immagine in regioni semanticamente omogenee e si assegna un vettore di movimento per regione.
- **Rappresentazione basata su feature:** si individuano punti salienti (corner, edge) e si traccia il loro movimento.

2.1.2 Background Subtraction e MOG2

La sottrazione di sfondo rappresenta una tecnica fondamentale per l'identificazione di oggetti in movimento. Il principio consiste nel confrontare ciascun fotogramma $I_t(\mathbf{x})$ con un modello dello sfondo statico $B(\mathbf{x})$, identificando come foreground i pixel che si discostano significativamente dal modello:

$$F_t(\mathbf{x}) = \begin{cases} 1 & \text{se } |I_t(\mathbf{x}) - B(\mathbf{x})| > \tau \\ 0 & \text{altrimenti} \end{cases} \quad (3)$$

dove τ è una soglia predefinita e $F_t(\mathbf{x})$ è la maschera binaria di foreground.

L'algoritmo **MOG2** (Mixture of Gaussians 2) rappresenta un'evoluzione dell'approccio classico MOG, introducendo un numero adattivo di distribuzioni gaussiane per modellare ciascun pixel dello sfondo. Ogni pixel viene modellato mediante una miscela di K distribuzioni gaussiane:

$$P(I_t(\mathbf{x})) = \sum_{k=1}^K w_{k,t}(\mathbf{x}) \cdot \mathcal{N}(I_t(\mathbf{x}) \mid \mu_{k,t}(\mathbf{x}), \sigma_{k,t}^2(\mathbf{x})) \quad (4)$$

dove $w_{k,t}$ sono i pesi della miscela, $\mu_{k,t}$ le medie e $\sigma_{k,t}^2$ le varianze. I parametri vengono aggiornati dinamicamente mediante un processo di apprendimento online che incorpora ogni nuovo fotogramma osservato.

MOG2 introduce inoltre un criterio di selezione automatica del numero ottimale di gaussiane per ciascun pixel, migliorando l'adattabilità a variazioni di illuminazione e riducendo i falsi positivi causati da ombre e riflessi.

2.2 Tracciamento Oggetti

Una volta individuate le regioni di foreground mediante background subtraction, è necessario associare tali regioni attraverso i fotogrammi successivi per costruire traiettorie temporali coerenti. Il tracciamento multi-oggetto richiede la risoluzione del problema di *data association*: dato un insieme di rilevamenti al tempo t e un insieme di tracce esistenti, determinare quali rilevamenti corrispondono a quali tracce.

2.2.1 Tracciamento basato su Distanza Euclidea

L'approccio più semplice per il tracciamento consiste nel calcolare la distanza euclidea tra i centroidi degli oggetti rilevati in fotogrammi consecutivi. Dato un oggetto con centroide $\mathbf{c}_t = (x_t, y_t)$ al tempo t e un insieme di rilevamenti con centroidi $\{\mathbf{r}_i\}$ al tempo $t + 1$, si associa l'oggetto al rilevamento più vicino:

$$\mathbf{r}^* = \arg \min_{\mathbf{r}_i} \|\mathbf{c}_t - \mathbf{r}_i\|_2 \quad (5)$$

dove la norma euclidea è definita come:

$$\|\mathbf{c}_t - \mathbf{r}_i\|_2 = \sqrt{(x_t - x_i)^2 + (y_t - y_i)^2} \quad (6)$$

L'associazione viene accettata solo se la distanza è inferiore a una soglia predefinita d_{\max} , garantendo che non vengano associate entità troppo distanti spazialmente.

2.2.2 Feature-based Tracking con ORB

ORB (Oriented FAST and Rotated BRIEF) rappresenta un algoritmo di feature detection e description efficiente e robusto. ORB combina il detector FAST (Features

from Accelerated Segment Test) per l'identificazione di keypoints con il descriptor BRIEF (Binary Robust Independent Elementary Features) orientato per garantire invarianza alla rotazione.

Il feature matching consente di associare oggetti basandosi non solo sulla posizione geometrica, ma anche su caratteristiche visuali locali (bordi, texture, pattern). Dati due oggetti O_t e O_{t+1} rilevati in fotogrammi consecutivi, si estraggono feature ORB dalle rispettive regioni e si calcolano le corrispondenze mediante **BFMatcher (Brute Force Matcher)** con distanza Hamming.

Lo score combinato di similarità integra:

- **Similarità feature:** numero di feature corrispondenti (threshold distanza Hamming minore di 50)
- **Distanza geometrica:** distanza euclidea tra centroidi normalizzata

$$S_{\text{match}} = 0.7 \cdot \frac{N_{\text{matches}}}{\max(N_{kp1}, N_{kp2})} + 0.3 \cdot \left(1 - \frac{d_{\text{eucl}}}{d_{\text{max}}}\right) \quad (7)$$

dove N_{matches} è il numero di feature corrispondenti, N_{kp1}, N_{kp2} sono i numeri di keypoints estratti dai due oggetti, d_{eucl} è la distanza euclidea tra centroidi e d_{max} è la soglia massima configurabile.

2.3 Stabilizzazione Video

La stabilizzazione video mira a compensare i movimenti indesiderati della camera (jitter, shake) che causano spostamenti globali dell'intera scena. Questi movimenti globali generano falsi positivi nella background subtraction, poiché pixel statici della scena appaiono in movimento relativo al frame di riferimento.

2.3.1 Stabilizzazione Feature-based

L'approccio feature-based stima la trasformazione geometrica tra fotogrammi consecutivi mediante:

1. **Estrazione feature:** identificazione di keypoints salienti (ORB detector con 2000 features)
2. **Feature matching:** corrispondenza di feature tra frame corrente e frame di riferimento (primo frame del video)
3. **Stima trasformazione:** calcolo della trasformazione affine ottimale mediante RANSAC
4. **Smoothing:** applicazione di moving average sulle trasformazioni per evitare movimenti bruschi
5. **Warping:** applicazione della trasformazione per allineare il frame corrente al riferimento

2.3.2 Trasformazione Affine

Una trasformazione affine 2D preserva parallelismo e rapporti di distanze lungo linee parallele, ed è definita da 6 parametri:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad (8)$$

La matrice 2×3 parametrizza rotazione, scaling, shearing e traslazione. L'algoritmo `estimateAffinePartial2D` di OpenCV stima i parametri ottimali minimizzando l'errore di riproiezione delle feature corrispondenti, utilizzando RANSAC per robustezza agli outliers.

2.4 Operazioni Morfologiche

Le operazioni morfologiche matematiche operano su immagini binarie modificando la forma delle regioni mediante operazioni di set theory con elementi strutturanti (kernel).

2.4.1 Closing (Chiusura)

L'operazione di closing è definita come dilatazione seguita da erosione:

$$A \bullet B = (A \oplus B) \ominus B \quad (9)$$

dove A è l'immagine binaria, B è l'elemento strutturante, \oplus denota dilatazione e \ominus erosione. Il closing **riempie piccoli buchi** e **connette regioni vicine**, utile per fondere detection frammentate dello stesso oggetto (es. tetto, cofano, finestrini di un'auto).

Nel progetto si utilizza un kernel ellittico 7×7 applicato con 2 iterazioni per garantire fusione efficace di regioni adiacenti.

2.4.2 Opening (Apertura)

L'opening è definito come erosione seguita da dilatazione:

$$A \circ B = (A \ominus B) \oplus B \quad (10)$$

L'opening **rimuove piccole regioni isolate** (rumore) preservando le regioni più grandi. Viene applicato dopo il closing (1 iterazione) per eliminare falsi positivi residui senza degradare le detection valide.

2.5 Non-Maximum Suppression (NMS)

Il Non-Maximum Suppression elimina detection duplicate dello stesso oggetto selezionando la bounding box più rappresentativa tra quelle sovrapposte. La sovrapposizione è quantificata mediante **Intersection over Union (IoU)**:

$$\text{IoU}(B_1, B_2) = \frac{\text{Area}(B_1 \cap B_2)}{\text{Area}(B_1 \cup B_2)} \quad (11)$$

dove B_1, B_2 sono due bounding box. L'algoritmo NMS implementato procede:

1. Ordinamento delle detection per area decrescente
2. Selezione della detection con area massima
3. Rimozione di tutte le detection con $\text{IoU} > \tau$ rispetto alla detection selezionata
4. Iterazione fino ad esaurimento delle detection

Il threshold IoU configurato è $\tau = 0.3$, bilanciando eliminazione di duplicati e preservazione di oggetti distinti parzialmente sovrapposti.

2.6 Coordinate Smoothing

Il coordinate smoothing riduce il jitter (instabilità) delle bounding box che può derivare da piccole variazioni nel rilevamento dei contorni frame-by-frame. Si applica un **exponential moving average (EMA)** sulle coordinate del centroide:

$$\mathbf{c}_t^{\text{smooth}} = \alpha \cdot \mathbf{c}_t + (1 - \alpha) \cdot \mathbf{c}_{t-1}^{\text{smooth}} \quad (12)$$

dove $\alpha = 0.3$ è il fattore di smoothing. Valori bassi di α producono traiettorie più fluide ma con maggiore latency; $\alpha = 0.3$ bilancia reattività e stabilità.

2.7 Metriche di Qualità per Video

La valutazione della qualità video può essere condotta secondo approcci soggettivi o oggettivi. Le metriche soggettive (MOS - Mean Opinion Score, DSCQS - Double Stimulus Continuous Quality Scale) richiedono la partecipazione di osservatori umani e risultano onerose in termini di tempo e costi. Le metriche oggettive automatizzano il processo mediante formule matematiche.

Le metriche oggettive si classificano in base alla disponibilità del segnale di riferimento:

- **Full-Reference (FR)**: richiedono la disponibilità completa del segnale originale non distorto.
- **Reduced-Reference (RR)**: utilizzano informazioni parziali estratte dal segnale originale.
- **No-Reference (NR)**: operano senza alcuna informazione sul segnale originale.

Nel presente progetto si adottano metriche full-reference, confrontando fotogramma per fotogramma le sequenze originali con quelle trasformate.

2.7.1 Mean Squared Error (MSE)

L'errore quadratico medio quantifica la divergenza media tra i valori dei pixel di due immagini. Date due immagini I_1 e I_2 di dimensioni $M \times N$, l'MSE è definito come:

$$\text{MSE}(I_1, I_2) = \frac{1}{MN} \sum_{i=1}^M \sum_{j=1}^N [I_1(i, j) - I_2(i, j)]^2 \quad (13)$$

Per sequenze video, si calcola l'MSE medio su tutti i fotogrammi corrispondenti:

$$\text{MSE}_{\text{video}} = \frac{1}{T} \sum_{t=1}^T \text{MSE}(I_{\text{orig}}^{(t)}, I_{\text{filt}}^{(t)}) \quad (14)$$

dove T è il numero totale di fotogrammi. Valori bassi di MSE indicano elevata similarità.

2.7.2 Peak Signal-to-Noise Ratio (PSNR)

Il PSNR esprime la qualità in termini logaritmici del rapporto tra il segnale massimo possibile e il rumore di distorsione, fornendo una metrica in decibel:

$$\text{PSNR}(I_1, I_2) = 10 \log_{10} \left(\frac{\text{MAX}_I^2}{\text{MSE}(I_1, I_2)} \right) = 20 \log_{10} \left(\frac{\text{MAX}_I}{\sqrt{\text{MSE}(I_1, I_2)}} \right) \quad (15)$$

dove MAX_I rappresenta il valore massimo assumibile dai pixel (255 per immagini a 8 bit). Maggiore è il suo valore maggiore sarà la “somiglianza” con l’originale.

2.7.3 Structural Similarity Index (SSIM)

L’SSIM supera le limitazioni di MSE e PSNR considerando la percezione umana della similarità strutturale. L’ipotesi fondamentale è che il sistema visivo umano sia particolarmente sensibile alle informazioni strutturali dell’immagine.

Date due finestre locali \mathbf{x} e \mathbf{y} estratte dalle immagini da confrontare, l’SSIM è definito come:

$$\text{SSIM}(\mathbf{x}, \mathbf{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \quad (16)$$

dove:

- μ_x, μ_y sono le medie locali
- σ_x^2, σ_y^2 sono le varianze locali
- σ_{xy} è la covarianza
- $C_1 = (K_1 L)^2, C_2 = (K_2 L)^2$ sono costanti di stabilizzazione ($K_1 = 0.01, K_2 = 0.03, L = 255$ per immagini a 8 bit)

L’indice SSIM assume valori nell’intervallo $[-1, 1]$, dove 1 indica identità perfetta e -1 massima dissimilarità. Per l’intera immagine, si calcola la media degli SSIM locali calcolati su finestre scorrevoli.

3 Architettura del Sistema

3.1 Panoramica dell'Architettura

Il sistema implementato adotta un'architettura modulare organizzata in quattro componenti principali, ciascuno responsabile di funzionalità specifiche:

1. **Modulo di orchestrazione** (`main.py`): coordina il flusso di esecuzione, gestisce l'acquisizione video, la selezione della ROI, implementa la **dual-pipeline architecture** (tracciamento parallelo su frame originali e frame stabilizzati+filtrati), applica stabilizzazione video, operazioni morfologiche, NMS, invoca gli algoritmi di tracking e filtraggio, e presenta i risultati.
2. **Modulo di tracciamento** (`tracciamento.py`): implementa la classe `Tracciamento` che gestisce l'assegnazione di identificatori univoci agli oggetti rilevati mediante **ORB feature matching**, calcolo score combinato (feature similarity + distanza euclidea), **coordinate smoothing** (exponential moving average), e **NMS con IoU** per eliminazione detection duplicate.
3. **Modulo di post-processing e stabilizzazione** (`filtri.py`): fornisce la classe `VideoStabilizer` per stabilizzazione feature-based mediante ORB (2000 features), trasformazione affine e smoothing, oltre alle funzioni per algoritmi di qualità ai fotogrammi (compressione JPEG, gaussian blurring, unsharp masking) e moving average spaziale.
4. **Modulo di valutazione** (`Calcolo_metriche.py`): implementa le metriche di qualità full-reference per il confronto quantitativo tra sequenze video.

La separazione modulare favorisce la mantenibilità, il testing e l'estensibilità del codice. Ciascun modulo può essere sviluppato, testato e modificato indipendentemente dagli altri, purché vengano rispettate le interfacce definite.

3.2 Dual-Pipeline Architecture

Il sistema implementa una **architettura a due pipeline parallele** per confrontare l'efficacia della stabilizzazione e del filtering:

1. **Pipeline Baseline**: elabora i frame originali senza preprocessing
 - Frame originale → MOG2 → Contour detection → Filtering (area) → Tracking
2. **Pipeline Stabilized+Filtered**: applica preprocessing avanzato prima del tracking
 - Frame originale → **Video Stabilization** → **Moving Average Spatial** → MOG2 → **Morphological Operations** → Contour detection → NMS → Tracking

Entrambe le pipeline operano sullo stesso video in parallelo, consentendo confronto visivo diretto tra i risultati. La pipeline stabilized+filtered riduce significativamente i falsi positivi dovuti a movimento camera e frammentazione oggetti.

3.3 Pipeline Principale

Il file `main.py` implementa la dual-pipeline di elaborazione video, orchestrando l'interazione tra i vari moduli. Il flusso di esecuzione si articola nelle seguenti fasi:

3.3.1 Fase 1: Inizializzazione e Configurazione

La fase di inizializzazione comprende:

```

1 # Inizializzazione due sistemi di tracciamento indipendenti
2 sistema_tracciamento = Tracciamento()                      # Pipeline
   baseline
3 sistema_tracciamento_stabilized_ma = Tracciamento()      # Pipeline
   stabilized
4
5 cattura_video = cv2.VideoCapture('video/traffico1.mp4')
6 if not cattura_video.isOpened():
7     print("Impossibile trovare il video!")
8     exit()
9
10 stato lettura, fotogramma_iniziale = cattura_video.read()
11 if not stato lettura:
12     print("Acquisizione fotogramma iniziale fallita!")
13     exit()
14
15 altezza_fotogramma, larghezza_fotogramma =
   fotogramma_iniziale.shape[:2]
16
17 # Inizializzazione Video Stabilizer con frame di riferimento
18 stabilizer = VideoStabilizer()
19 stabilizer.set_reference_frame(fotogramma_iniziale)

```

Listing 1: Inizializzazione dual tracking system e caricamento video

Vengono istanziati **due oggetti Tracciamento** indipendenti: uno per la pipeline baseline (frame originali) e uno per la pipeline stabilized+filtered. Il video viene caricato mediante la classe `VideoCapture` di OpenCV. Il primo fotogramma viene utilizzato come **frame di riferimento** per la stabilizzazione video.

3.3.2 Fase 2: Selezione della Region of Interest (ROI)

Il sistema offre tre modalità per la definizione della ROI:

1. **Modalità manuale:** l'utente seleziona interattivamente un'area rettangolare mediante interfaccia grafica
2. **Modalità automatica:** il sistema definisce automaticamente una ROI centrata contenente il 60% dell'area totale
3. **Fotogramma completo:** l'intera area del fotogramma viene considerata come ROI

```

1 if scelta_modalita == "2":
2     percentuale_larghezza_roi = 0.6
3     percentuale_altezza_roi = 0.6
4
5     larghezza_roi = int(larghezza_fotogramma *
6                         percentuale_larghezza_roi)
7     altezza_roi = int(altezza_fotogramma *
8                         percentuale_altezza_roi)
9     pos_x_roi = (larghezza_fotogramma - larghezza_roi) // 2
10    pos_y_roi = (altezza_fotogramma - altezza_roi) // 2

```

Listing 2: Configurazione automatica della ROI

La possibilità di limitare l'elaborazione a una ROI consente di ridurre il carico computazionale concentrandosi sulle aree di interesse, particolarmente utile in scenari dove il movimento si concentra in regioni specifiche del fotogramma.

3.3.3 Fase 3: Inizializzazione Background Subtractor

Il rilevatore di foreground viene inizializzato con l'algoritmo MOG2:

```

1 # MOG2 per pipeline baseline (frame originali)
2 rilevatore_entita = cv2.createBackgroundSubtractorMOG2(
3     history=200,
4     varThreshold=40
5 )
6
7 # MOG2 per pipeline stabilized+filtered
8 rilevatore_entita_stabilized_ma =
9     cv2.createBackgroundSubtractorMOG2(
10        history=200,
11        varThreshold=40
11 )

```

Listing 3: Inizializzazione MOG2 per entrambe le pipeline

I parametri configurati sono:

- **history=200**: numero di fotogrammi considerati per la costruzione del modello di background
- **varThreshold=40**: soglia sulla distanza di Mahalanobis al quadrato per classificare un pixel come foreground

Vengono istanziati **due background subtractor indipendenti** per evitare contaminazione tra le due pipeline, permettendo confronto equo.

3.3.4 Fase 3.5: Kernel Morfologico

Si definisce il kernel per le operazioni morfologiche:

```

1 kernel_morph = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7,
7))

```

Listing 4: Definizione kernel ellittico

Un kernel ellittico 7×7 garantisce operazioni morfologiche isotrope (uniformi in tutte le direzioni), adatto per fusione di regioni di oggetti con forma irregolare.

3.3.5 Fase 4: Loop di Elaborazione Fotogrammi

Il ciclo principale elabora sequenzialmente tutti i fotogrammi del video implementando le due pipeline parallele:

```

1 in_pausa = False # Flag per pausa video
2
3 while True:
4     # Gestione controlli tastiera
5     tasto = cv2.waitKey(1) & 0xFF
6     if tasto == ord('q'):
7         break
8     elif tasto == ord(' '): # SPAZIO per pausa/resume
9         in_pausa = not in_pausa
10
11    # Loop di attesa durante pausa
12    while in_pausa:
13        tasto = cv2.waitKey(100) & 0xFF
14        if tasto == ord(' '):
15            in_pausa = False
16        elif tasto == ord('q'):
17            break
18
19    if tasto == ord('q'):
20        break
21
22    stato_acquisizione, fotogramma_attuale = cattura_video.read()
23    if not stato_acquisizione:
24        break
25
26    # Estrazione ROI
27    regione_interesse = fotogramma_attuale[
28        pos_y_roi:pos_y_roi+altezza_roi,
29        pos_x_roi:pos_x_roi+larghezza_roi
30    ]
31
32    # ===== PIPELINE STABILIZED+FILTERED =====
33    # 1. Stabilizzazione video
34    frame_stabilizzato =
35        stabilizer.stabilize(fotogramma_attuale.copy())
36
# 2. Moving Average Spaziale (riduzione rumore)

```

```

37     frame_ma = moving_average_filter(frame_stabilizzato)
38
39     # Estrazione ROI dal frame processato
40     roi_stabilized_ma = frame_ma[
41         pos_y_roi:pos_y_roi+altezza_roi,
42         pos_x_roi:pos_x_roi+larghezza_roi
43     ]
44
45     # 3. Background subtraction
46     maschera_stabilized_ma =
47         rilevatore_entita_stabilized_ma.apply(
48             roi_stabilized_ma
49         )
50     _, maschera_stabilized_ma = cv2.threshold(
51         maschera_stabilized_ma, 254, 255, cv2.THRESH_BINARY
52     )
53
54     # 4. Operazioni morfologiche
55     # Closing: fusione regioni frammentate
56     # (tetto+cofan+finestrini)
57     maschera_stabilized_ma = cv2.morphologyEx(
58         maschera_stabilized_ma, cv2.MORPH_CLOSE, kernel_morph,
59         iterations=2
60     )
61     # Opening: rimozione rumore
62     maschera_stabilized_ma = cv2.morphologyEx(
63         maschera_stabilized_ma, cv2.MORPH_OPEN, kernel_morph,
64         iterations=1
65     )
66
67     # ===== PIPELINE BASELINE (frame originale) =====
68     maschera_rilevamento =
69         rilevatore_entita.apply(regione_interesse)
70     _, maschera_rilevamento = cv2.threshold(
71         maschera_rilevamento, 254, 255, cv2.THRESH_BINARY
72     )

```

Listing 5: Loop principale con dual-pipeline

Il sistema implementa:

- **Controlli tastiera:** Q per uscita, SPAZIO per pausa/resume
- **Stabilizzazione** del frame completo mediante ORB feature matching
- **Moving average spaziale** (kernel 5×5) per smoothing
- **Background subtraction** indipendente per ciascuna pipeline
- **Operazioni morfologiche** solo su pipeline stabilized: closing (2 iterazioni) + opening (1 iterazione)

3.3.6 Fase 5: Rilevamento Contorni, NMS e Tracking Dual-Pipeline

Sulla maschera binaria si individuano i contorni degli oggetti in foreground per entrambe le pipeline:

```

1 # ===== PIPELINE BASELINE =====
2 contorni_rilevati, _ = cv2.findContours(
3     maschera_rilevamento,
4     cv2.RETR_EXTERNAL,    # Solo contorni esterni
5     cv2.CHAIN_APPROX_SIMPLE
6 )
7
8 rilevamenti_validi = []
9 for singolo_contorno in contorni_rilevati:
10     superficie_contorno = cv2.contourArea(singolo_contorno)
11     if superficie_contorno > 500:  # Soglia area aumentata
12         coord_x, coord_y, dimensione_w, dimensione_h = \
13             cv2.boundingRect(singolo_contorno)
14         rilevamenti_validi.append([coord_x, coord_y,
15                                     dimensione_w, dimensione_h])
16
17 # Tracking pipeline baseline
18 risultati_tracciamento =
19     sistema_tracciamento.update(rilevamenti_validi)
20
21 # ===== PIPELINE STABILIZED+FILTERED =====
22 contorni_stabilized, _ = cv2.findContours(
23     maschera_stabilized_ma,
24     cv2.RETR_EXTERNAL,
25     cv2.CHAIN_APPROX_SIMPLE
26 )
27
28 rilevamenti_stabilized_ma = []
29 for contorno in contorni_stabilized:
30     area = cv2.contourArea(contorno)
31     if area > 500:
32         x, y, w, h = cv2.boundingRect(contorno)
33         rilevamenti_stabilized_ma.append([x, y, w, h])
34
35 # 5. NMS: eliminazione detection duplicate
36 rilevamenti_stabilized_ma = filtra_box_sovrapposti(
37     rilevamenti_stabilized_ma,
38     iou_threshold=0.3
39 )
40
41 # Tracking pipeline stabilized+filtered
42 risultati_stabilized_ma =
43     sistema_tracciamento_stabilized_ma.update(
44         rilevamenti_stabilized_ma
45 )

```

Listing 6: Rilevamento contorni, NMS e tracking parallelo

Le modifiche chiave rispetto alla versione baseline:

1. **RETR_EXTERNAL**: estrae solo contorni esterni, evitando contorni gerarchici interni
2. **NMS con IoU threshold 0.3**: elimina bounding box sovrapposte mantenendo quella con area maggiore
3. **Tracking parallelo**: due sistemi di tracciamento indipendenti per confronto

La funzione `filtra_box_sovrapposti` implementa l'algoritmo NMS:

```

1 def filtra_box_sovrapposti(boxes, iou_threshold=0.5):
2     if len(boxes) == 0:
3         return []
4
5     # Ordinamento per area decrescente
6     boxes_sorted = sorted(boxes,
7                           key=lambda b: b[2] * b[3],
8                           reverse=True)
9
10    filtered_boxes = []
11    while boxes_sorted:
12        # Selezione box con area massima
13        current_box = boxes_sorted.pop(0)
14        filtered_boxes.append(current_box)
15
16        # Rimozione box sovrapposti
17        boxes_sorted = [
18            box for box in boxes_sorted
19            if calcola_iou(current_box, box) < iou_threshold
20        ]
21
22    return filtered_boxes

```

Listing 7: Algoritmo NMS con IoU

3.3.7 Fase 7: Applicazione Algoritmi e Memorizzazione

Per ciascun fotogramma si applicano tre algoritmi di elaborazione distinti, memorizzando le sequenze risultanti:

```

1 # Compressione JPEG
2 fotogramma_solo_jpeg = jpeg_compression(
3     fotogramma_attuale.copy(), quality=50
4 )
5 fotogrammi_jpeg.append(fotogramma_solo_jpeg)
6

```

```

7 # Sfocatura Gaussiana
8 fotogramma_solo_blurred = blurring(
9     fotogramma_attuale.copy(), kernel_size=(5, 5), sigma=0
10)
11 fotogrammi_blurred.append(fotogramma_solo_blurred)
12
13 # Unsharp Masking (Sharpening)
14 fotogramma_solo_sharpened = unsharp_masking(
15     fotogramma_attuale.copy(), amount=1.5
16)
17 fotogrammi_sharpened.append(fotogramma_solo_sharpened)

```

Listing 8: Applicazione algoritmi di qualità

È fondamentale passare copie dei fotogrammi alle funzioni di elaborazione per preservare l'originale intatto durante il confronto metrico.

3.3.8 Fase 8: Visualizzazione Real-Time Dual-Pipeline

Il sistema presenta contemporaneamente finestre multiple per il monitoraggio comparativo real-time:

Finestre Pipeline Baseline:

- ROI originale con annotazioni tracking (ID e bounding box verdi)
- Maschera background subtraction baseline

Finestre Pipeline Stabilized+Filtered:

- Frame stabilizzato completo (output VideoStabilizer)
- Frame stabilizzato + moving average spatial
- ROI stabilized+MA con annotazioni tracking (ID e bounding box blu per distinzione)
- Maschera background subtraction dopo operazioni morfologiche

Finestre Valutazione Qualità (su frame originale):

- Sequenza compressa JPEG (Q=50)
- Sequenza blurred (Gaussian 5x5)
- Sequenza sharpened (Unsharp Masking 1.5x)

```

1 # Pipeline Baseline
2 cv2.imshow("ROI Original + Tracking", regione_interesse)
3 cv2.imshow("Mask Original", maschera_ridimensionata)
4
5 # Pipeline Stabilized+Filtered
6 cv2.imshow("Frame Stabilized", frame_stabilizzato)

```

```

7 cv2.imshow("Frame Stabilized + MA", frame_ma)
8 cv2.imshow("ROI Stabilized+MA + Tracking",
9             roi_stabilized_ma_display)
10 cv2.imshow("Mask Stabilized+MA (after morphology)",
11             maschera_stabilized_ridim)

12 # Valutazione Qualità
13 cv2.imshow("JPEG Compressed (Q=50)", fotogramma_jpeg)
14 cv2.imshow("Gaussian Blur (5x5)", fotogramma_blurred)
15 cv2.imshow("Unsharp Masked (1.5x)", fotogramma_sharpened)

16 # Controlli: Q=quit, SPAZIO=pause/resume
17 tasto = cv2.waitKey(1) & 0xFF
18 if tasto == ord('q'):
19     break
20 elif tasto == ord(' '):
21     in_pausa = not in_pausa
22

```

Listing 9: Visualizzazione finestre dual-pipeline

La visualizzazione dual-pipeline consente confronto visivo immediato tra baseline e stabilized, evidenziando:

- Riduzione falsi positivi nella maschera stabilizzata
- Fusione regioni frammentate (1 box invece di 3-5)
- Maggiore stabilità ID tracking (colori bounding box cambiano meno frequentemente)

3.3.9 Fase 9: Calcolo Metriche di Qualità

Terminato il loop di elaborazione, si calcolano le metriche confrontando le sequenze:

```

1 # ORIGINALE vs JPEG Compression
2 mse_jpeg, psnr_jpeg, ssim_jpeg = compare_video_sequences(
3     sequenza_fotogrammi, fotogrammi_jpeg
4 )
5
6 # ORIGINALE vs Gaussian Blurring
7 mse_blurred, psnr_blurred, ssim_blurred =
8     compare_video_sequences(
9         sequenza_fotogrammi, fotogrammi_blurred
10    )
11
12 # ORIGINALE vs Unsharp Masking
13 mse_sharpened, psnr_sharpened, ssim_sharpened =
14     compare_video_sequences(
15         sequenza_fotogrammi, fotogrammi_sharpened
16    )

```

Listing 10: Calcolo metriche comparative

I risultati vengono stampati a console fornendo una valutazione quantitativa dell'impatto di ciascun algoritmo sulla qualità dell'immagine.

4 Algoritmo di Tracking

4.1 Classe Tracciamento

Il modulo `tracciamento.py` implementa la classe `Tracciamento`, responsabile dell'associazione temporale degli oggetti rilevati attraverso i fotogrammi successivi mediante **ORB feature matching** combinato con distanza euclidea e coordinate smoothing. L'obiettivo è mantenere identificatori univoci e persistenti per ciascun oggetto durante tutto il periodo in cui rimane visibile nella scena.

4.1.1 Struttura della Classe

La classe mantiene strutture dati per feature matching e smoothing:

```

1 import cv2
2 import math
3 import numpy as np
4
5 class Tracciamento:
6     def __init__(self, distanza_max=350):
7         # Contatore incrementale per ID univoci
8         self.identificativo_corrente = 0
9
10        # Dizionario: ID -> (centro_x, centro_y)
11        self.coordinate_centrali = {}
12
13        # ORB detector per feature extraction (100 features)
14        self.orb = cv2.ORB_create(nfeatures=100)
15
16        # BruteForce Matcher con distanza Hamming
17        self.bf = cv2.BFMatcher(cv2.NORM_HAMMING,
18                               crossCheck=False)
19
20        # Dizionario: ID -> (keypoints, descriptors)
21        self.features_oggetti = {}
22
23        # Coordinate smoothing per riduzione jitter
24        self.coordinate_smoothing =
25            CoordinateSmoothing(alpha=0.3)
26
27        # Soglia distanza massima configurabile
28        self.distanza_max = distanza_max

```

Listing 11: Inizializzazione classe Tracciamento avanzata

4.2 Metodo di Aggiornamento con Feature Matching

Il metodo `update()` integra feature matching ORB con distanza euclidea per associazione robusta:

```

1 def update(self, rettangoli_rilevati, frame_roi):
2     risultati_tracciamento = []
3
4     for rettangolo in rettangoli_rilevati:
5         x, y, w, h = rettangolo
6
7         # Calcolo centroide
8         centro_x = (x + x + w) // 2
9         centro_y = (y + y + h) // 2
10
11        # Estrazione ROI oggetto per feature extraction
12        roi_oggetto = frame_roi[y:y+h, x:x+w]
13
14        # Conversione grayscale e feature extraction
15        if len(roi_oggetto.shape) == 3:
16            roi_gray = cv2.cvtColor(roi_oggetto,
17                                   cv2.COLOR_BGR2GRAY)
18        else:
19            roi_gray = roi_oggetto
20
21        kp_nuovo, desc_nuovo = self.orb.detectAndCompute(
22            roi_gray, None
23        )
24
25        # Ricerca migliore match tra oggetti tracciati
26        best_match_id = None
27        best_score = 0
28
29        for id_obj, (kp_old, desc_old) in
30            self.features_oggetti.items():
31            if desc_nuovo is None or desc_old is None:
32                continue
33
34            # Feature matching
35            matches = self.bf.knnMatch(desc_nuovo, desc_old, k=2)
36
37            # Ratio test di Lowe
38            good_matches = []
39            for match_pair in matches:
40                if len(match_pair) == 2:
41                    m, n = match_pair
42                    if m.distance < 0.75 * n.distance:
43                        good_matches.append(m)
44
45        # Calcolo score combinato (70% features, 30%

```

```

        distanza)
44     feature_score = len(good_matches) /
45         max(len(kp_nuovo),
46             len(kp_old))
47
48     centro_old = self.coordinate_centrali[id_obj]
49     dist_eucl = math.hypot(centro_x - centro_old[0],
50                           centro_y - centro_old[1])
51
52     if dist_eucl > self.distanza_max:
53         continue
54
55     dist_score = 1.0 - (dist_eucl / self.distanza_max)
56
57     combined_score = 0.7 * feature_score + 0.3 *
58     dist_score
59
60     if combined_score > best_score:
61         best_score = combined_score
62         best_match_id = id_obj
63
64     # Soglia minima score per accettazione match
65     if best_match_id is not None and best_score > 0.3:
66         # Aggiornamento oggetto esistente
67         # Coordinate smoothing per riduzione jitter
68         centro_smooth = self.coordinate_smoothing.smooth(
69             best_match_id, (centro_x, centro_y)
70         )
71
72         self.coordinate_centrali[best_match_id] =
73             centro_smooth
74         self.features_oggetti[best_match_id] = (kp_nuovo,
75                                                 desc_nuovo)
76
77         risultati_tracciamento.append([
78             best_match_id, x, y, w, h
79         ])
80     else:
81         # Nuova traccia
82         self.coordinate_centrali[self.identificativo_corrente] =
83             \
84             (centro_x, centro_y)
85         self.features_oggetti[self.identificativo_corrente] =
86             \
87             (kp_nuovo, desc_nuovo)
88
89         risultati_tracciamento.append([
90             self.identificativo_corrente, x, y, w, h
91         ])

```

```

87         self.identificativo_corrente += 1
88
89     # Pulizia tracce obsolete
90     self._rimuovi_entita_obsolete(risultati_tracciamento)
91
92     return risultati_tracciamento

```

Listing 12: Metodo update con ORB feature matching

4.3 Coordinate Smoothing

La classe CoordinateSmoothing implementa exponential moving average:

```

1  class CoordinateSmoothing:
2      def __init__(self, alpha=0.3):
3          self.alpha = alpha    # Fattore smoothing
4          self.previous_positions = {}
5
6      def smooth(self, obj_id, new_position):
7          if obj_id not in self.previous_positions:
8              self.previous_positions[obj_id] = new_position
9              return new_position
10
11     prev_x, prev_y = self.previous_positions[obj_id]
12     new_x, new_y = new_position
13
14     # Exponential moving average
15     smooth_x = int(self.alpha * new_x +
16                     (1 - self.alpha) * prev_x)
17     smooth_y = int(self.alpha * new_y +
18                     (1 - self.alpha) * prev_y)
19
20     smoothed = (smooth_x, smooth_y)
21     self.previous_positions[obj_id] = smoothed
22
23     return smoothed

```

Listing 13: Classe CoordinateSmoothing

4.4 Gestione Tracce Obsolete

Il metodo privato `_rimuovi_entita_obsolete()` elimina dal dizionario le tracce di oggetti non più rilevati:

```

1  def _rimuovi_entita_obsolete(self, risultati_tracciamento):
2      # Estrazione ID attivi
3      id_attivi = {r[0] for r in risultati_tracciamento}
4
5      # Identificazione ID da rimuovere
6      id_da_rimuovere = [

```

```

7         id_ent for id_ent in self.coordinate_centrali.keys()
8             if id_ent not in id_attivi
9         ]
10
11     # Rimozione
12     for id_ent in id_da_rimuovere:
13         del self.coordinate_centrali[id_ent]
14         if id_ent in self.features_oggetti:
15             del self.features_oggetti[id_ent]

```

Listing 14: Rimozione tracce obsolete

Questo meccanismo garantisce che oggetti usciti dalla scena o occultati non mantengano indefinitamente risorse.

4.5 Non-Maximum Suppression (NMS)

La funzione `filtra_box_sovrapposti()` implementa l'algoritmo NMS con calcolo IoU:

```

1 def filtra_box_sovrapposti(boxes, iou_threshold=0.5):
2     """
3         Elimina bounding box duplicate mediante NMS.
4
5     Args:
6         boxes: Lista di [x, y, w, h]
7         iou_threshold: Soglia IoU per eliminazione (default 0.5)
8
9     Returns:
10        Lista filtrata di bounding box
11    """
12    if len(boxes) == 0:
13        return []
14
15    # Ordinamento per area decrescente
16    boxes_sorted = sorted(boxes,
17                          key=lambda b: b[2] * b[3],
18                          reverse=True)
19
20    filtered_boxes = []
21
22    while boxes_sorted:
23        # Selezione box con area massima
24        current_box = boxes_sorted.pop(0)
25        filtered_boxes.append(current_box)
26
27        # Filtraggio box sovrapposti
28        boxes_sorted = [
29            box for box in boxes_sorted
30            if calcola_iou(current_box, box) < iou_threshold
31        ]

```

```

32
33     return filtered_boxes
34
35 def calcola_iou(box1, box2):
36     """Calcola Intersection over Union tra due box."""
37     x1, y1, w1, h1 = box1
38     x2, y2, w2, h2 = box2
39
40     # Coordinate intersezione
41     xi1 = max(x1, x2)
42     yi1 = max(y1, y2)
43     xi2 = min(x1 + w1, x2 + w2)
44     yi2 = min(y1 + h1, y2 + h2)
45
46     # Area intersezione
47     inter_width = max(0, xi2 - xi1)
48     inter_height = max(0, yi2 - yi1)
49     inter_area = inter_width * inter_height
50
51     # Area unione
52     box1_area = w1 * h1
53     box2_area = w2 * h2
54     union_area = box1_area + box2_area - inter_area
55
56     # IoU
57     if union_area == 0:
58         return 0.0
59
60     return inter_area / union_area

```

Listing 15: Implementazione NMS completa

L'algoritmo garantisce eliminazione efficiente di detection duplicate preservando quella con area maggiore (tipicamente la più accurata).

4.6 Limitazioni e Miglioramenti

4.6.1 Limitazioni

- Occlusioni complete prolungate:** se un oggetto viene completamente occultato per più di 2-3 frame, perde comunque il proprio ID
- Assenza di predizione:** non si prevede la posizione futura dell'oggetto basandosi sulla traiettoria passata (approcci Kalman filter potrebbero migliorare questo aspetto)

5 Algoritmi di Valutazione della Qualità

Il modulo `filtri.py` implementa la classe `VideoStabilizer` per stabilizzazione video feature-based e quattro algoritmi di elaborazione video finalizzati alla valutazione della qualità dell'immagine mediante metriche quantitative.

5.1 Video Stabilizer

La classe `VideoStabilizer` implementa stabilizzazione feature-based utilizzando ORB detector e trasformazioni affini.

5.1.1 Architettura della Classe

```

1 import cv2
2 import numpy as np
3 from collections import deque
4
5 class VideoStabilizer:
6     def __init__(self, smoothing_window=5):
7         # ORB detector con 2000 features
8         self.orb = cv2.ORB_create(nfeatures=2000)
9
10        # BFMatcher per feature matching
11        self.bf = cv2.BFMatcher(cv2.NORM_HAMMING,
12                               crossCheck=True)
13
14        # Frame di riferimento
15        self.reference_frame = None
16        self.reference_kp = None
17        self.reference_desc = None
18
19        # Buffer per smoothing trasformazioni
20        self.transform_buffer = deque(maxlen=smoothing_window)
21
22    def set_reference_frame(self, frame):
23        """Imposta frame di riferimento per stabilizzazione."""
24        self.reference_frame = frame.copy()
25
26        # Estrazione feature dal frame di riferimento
27        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
28        self.reference_kp, self.reference_desc = \
29            self.orb.detectAndCompute(gray, None)
30
31    def stabilize(self, frame):
32        """
33            Stabilizza un frame allineandolo al riferimento.
34
35        Args:

```

```

35         frame: Frame BGR da stabilizzare
36
37     Returns:
38         Frame stabilizzato (warped)
39     """
40
41     if self.reference_frame is None:
42         return frame
43
44     # Conversione grayscale
45     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
46
47     # Feature detection e matching
48     kp, desc = self.orb.detectAndCompute(gray, None)
49
50     if desc is None or self.reference_desc is None:
51         return frame
52
53     # Matching features
54     matches = self.bf.match(self.reference_desc, desc)
55
56     # Selezione best matches (almeno 10 richiesti)
57     if len(matches) < 10:
58         return frame
59
60     # Ordinamento per distanza
61     matches = sorted(matches, key=lambda x: x.distance)
62
63     # Estrazione coordinate matched points
64     src_pts = np.float32([
65         self.reference_kp[m.queryIdx].pt for m in matches
66     ]).reshape(-1, 1, 2)
67
68     dst_pts = np.float32([
69         kp[m.trainIdx].pt for m in matches
70     ]).reshape(-1, 1, 2)
71
72     # Stima trasformazione affine con RANSAC
73     transform_matrix, inliers = cv2.estimateAffinePartial2D(
74         dst_pts, src_pts,
75         method=cv2.RANSAC,
76         ransacReprojThreshold=5.0
77     )
78
79     if transform_matrix is None:
80         return frame
81
82     # Smoothing trasformazione
83     self.transform_buffer.append(transform_matrix)
84     smooth_transform = np.mean(self.transform_buffer, axis=0)

```

```

84
85     # Warping frame
86     h, w = frame.shape [:2]
87     stabilized = cv2.warpAffine(
88         frame, smooth_transform, (w, h),
89         flags=cv2.INTER_LINEAR,
90         borderMode=cv2.BORDER_REFLECT
91     )
92
93     return stabilized

```

Listing 16: Classe VideoStabilizer

Il processo di stabilizzazione:

1. Estrazione di 2000 feature ORB dal frame corrente
2. Matching con feature del frame di riferimento
3. Stima trasformazione affine ottimale (RANSAC per robustezza)
4. Smoothing della trasformazione (media mobile su 5 frame)
5. Applicazione warping per allineare frame al riferimento

5.2 Moving Average Spatial Filter

Il filtro moving average spaziale applica smoothing uniforme mediante convoluzione:

```

1 def moving_average_filter(frame, kernel_size=5):
2     """
3         Applica moving average spaziale (smoothing uniforme).
4
5     Args:
6         frame: Immagine BGR
7         kernel_size: Dimensione kernel (default 5x5)
8
9     Returns:
10        Immagine filtrata
11    """
12    kernel = np.ones((kernel_size, kernel_size), np.float32) / \
13        (kernel_size * kernel_size)
14
15    filtered = cv2.filter2D(frame, -1, kernel)
16
17    return filtered

```

Listing 17: Moving average spaziale

Il kernel uniforme 5×5 produce smoothing moderato riducendo rumore ad alta frequenza senza degradare eccessivamente i dettagli.

5.3 Compressione JPEG

La compressione JPEG è uno standard lossy ampiamente utilizzato che introduce artefatti di quantizzazione caratteristici. L'algoritmo opera trasformando l'immagine dallo spazio RGB a YCbCr, applicando la Discrete Cosine Transform (DCT) su blocchi 8×8 pixel, e quantizzando i coefficienti ad alta frequenza.

Il livello di compressione è controllato dal parametro `quality` $\in [0, 100]$, dove valori inferiori producono file più piccoli ma maggiori distorsioni visive. Gli artefatti tipici includono:

- **Blocking artifacts**: discontinuità tra blocchi adiacenti
- **Ringing**: oscillazioni spurie vicino ai bordi netti
- **Color bleeding**: perdita di dettaglio cromatico dovuta al subsampling dei canali di crominanza

5.3.1 Implementazione

```

1 def jpeg_compression(immagine_input, quality=50):
2     """
3         Applica compressione JPEG con livello di qualita specificato.
4
5     Args:
6         immagine_input: Immagine BGR di input
7         quality: Livello qualita JPEG (0-100, default=50)
8
9     Returns:
10        Immagine BGR compressa e decompressa
11    """
12    encode_param = [int(cv2.IMWRITE_JPEG_QUALITY), quality]
13    result, encoded_img = cv2.imencode('.jpg', immagine_input,
14                                         encode_param)
15    decoded_img = cv2.imdecode(encoded_img, cv2.IMREAD_COLOR)
16    return decoded_img

```

Listing 18: Funzione jpeg_compression

Il valore `quality=50` rappresenta un compromesso intermedio tra dimensione file e fedeltà visiva. Questa trasformazione introduce degradazioni misurabili dalle metriche: aumento di MSE, diminuzione di PSNR e SSIM.

5.4 Sfocatura Gaussiana

La sfocatura gaussiana rappresenta un filtro di smoothing lineare che riduce il rumore ad alta frequenza e i dettagli fini tramite convoluzione con un kernel gaussiano bidimensionale:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (17)$$

dove σ è la deviazione standard che controlla l'ampiezza della distribuzione. Valori maggiori producono sfocature più pronunciate. Il filtro gaussiano è separabile, permettendo un'implementazione efficiente mediante due convoluzioni 1D sequenziali.

5.4.1 Implementazione

```

1 def blurring(immagine_input, kernel_size=(5, 5), sigma=0):
2     """
3         Applica sfocatura gaussiana all'immagine.
4
5     Args:
6         immagine_input: Immagine BGR di input
7         kernel_size: Dimensioni kernel (dispari, default=(5,5))
8         sigma: Deviazione standard (0=auto, default=0)
9
10    Returns:
11        Immagine BGR sfocata
12    """
13    immagine_sfocata = cv2.GaussianBlur(immagine_input,
14                                         kernel_size, sigma)
15    return immagine_sfocata

```

Listing 19: Funzione blurring

Con `kernel_size=(5, 5)` e `sigma=0` (calcolo automatico), il filtro produce un moderato effetto di smoothing. L'applicazione riduce MSE dovuto al rumore ma degrada SSIM per la perdita di dettagli ad alta frequenza.

5.5 Unsharp Masking

L'unsharp masking è una tecnica di sharpening che aumenta il contrasto locale in prossimità dei bordi. Il processo si articola in tre fasi:

1. Creazione di una versione sfocata dell'immagine originale
2. Sottrazione della versione sfocata dall'originale per ottenere una "maschera" contenente i dettagli ad alta frequenza
3. Somma della maschera amplificata all'immagine originale

Matematicamente:

$$I_{\text{sharp}} = I + \alpha \cdot (I - I_{\text{blur}}) \quad (18)$$

dove α è il parametro `amount` che controlla l'intensità dello sharpening.

5.5.1 Implementazione

```

1 def unsharp_masking(immagine_input, kernel_size=(5, 5),
2                     sigma=1.0, amount=1.5, threshold=0):
3     """
4         Applica unsharp masking per aumentare nitidezza.
5
6     Args:
7         immagine_input: Immagine BGR di input
8         kernel_size: Dimensioni kernel Gaussian (default=(5,5))
9         sigma: Deviazione standard blur (default=1.0)
10        amount: Intensità sharpening (default=1.5)
11        threshold: Soglia minima contrasto (default=0)
12
13    Returns:
14        Immagine BGR sharpened
15    """
16
17    # Creazione versione sfocata
18    blurred = cv2.GaussianBlur(immagine_input, kernel_size,
19                               sigma)
20
21    # Calcolo maschera dettagli
22    mask = cv2.subtract(immagine_input, blurred)
23
24    # Applicazione maschera amplificata
25    sharpened = cv2.addWeighted(immagine_input, 1.0,
26                                mask, amount, 0)
27
28    # Clipping valori
29    sharpened = np.clip(sharpened, 0, 255).astype(np.uint8)
30
31    return sharpened

```

Listing 20: Funzione unsharp_masking

Con `amount=1.5`, lo sharpening enfatizza moderatamente i bordi senza introdurre artefatti di halo eccessivi. Questa trasformazione può migliorare la nitidezza percettiva mantenendo buoni valori SSIM.

5.6 Conclusioni

Gli algoritmi implementati rappresentano categorie distinte di elaborazione con impatti differenziati sulla qualità:

- **Video Stabilization:** compensazione movimenti camera preservando contenuto (MSE minimo, SSIM alto se camera stabile)
- **Moving Average Spatial:** smoothing uniforme riducendo rumore ad alta frequenza (MSE variabile, lieve degradazione SSIM)

- **JPEG compression:** introduce degradazioni lossy quantificabili (MSE \uparrow , PSNR \downarrow , SSIM \downarrow)
- **Gaussian blurring:** filtraggio passa-basso con perdita dettagli (MSE variabile, SSIM \downarrow)
- **Unsharp masking:** enfatizza bordi e dettagli (SSIM stabile/migliorata se moderato)

Le trasformazioni di stabilizzazione e moving average sono **preprocessing** applicati prima del tracking per migliorarne robustezza, mentre JPEG/blur/sharpening sono **post-processing** per valutazione qualità mediante metriche. Queste ultime consentono di testare la sensibilità e la capacità discriminativa delle metriche MSE, PSNR e SSIM rispetto a diversi tipi di distorsioni e miglioramenti, con SSIM che meglio riflette la percezione visiva umana.

6 Calcolo Metriche di Qualità

Il modulo `Calcolo_metriche.py` implementa la funzione `compare_video_sequences()` che calcola MSE, PSNR e SSIM confrontando fotogramma per fotogramma due sequenze video.

6.1 Architettura della Funzione

Questa implementazione confronta fotogrammi corrispondenti di due sequenze distinte: l'originale e quella trasformata. Questo approccio full-reference quantifica l'impatto della trasformazione applicata.

6.1.1 Validazione Input

```

1 def compare_video_sequences(sequenza_originale,
2     sequenza_filtrata):
3     # Verifica lunghezze
4     if len(sequenza_originale) != len(sequenza_filtrata):
5         print(f"ERRORE: Le due sequenze hanno lunghezze
6             diverse!")
7         print(f"    Originale: {len(sequenza_originale)} frame")
8         print(f"    Filtrata: {len(sequenza_filtrata)} frame")
9         return None, None, None
10
11     # Verifica non vuote
12     if len(sequenza_originale) < 1:
13         print("ERRORE: Sequenze vuote!")
14         return None, None, None
15
16     numero_fotogrammi = len(sequenza_originale)

```

Listing 21: Validazione sequenze video

La funzione verifica che le due sequenze abbiano la stessa lunghezza (requisito fondamentale per confronti frame-by-frame) e che contengano almeno un fotogramma.

6.2 Calcolo MSE

L'errore quadratico medio viene calcolato come:

```

1 errore_quadratico = np.mean(
2     (frame_orig.astype(float) - frame_filt.astype(float)) ** 2
3 )
4 somma_mse += errore_quadratico

```

Listing 22: Calcolo MSE per singolo frame

L'MSE finale è la media su tutti i fotogrammi:

```

1 mse_medio = somma_mse / numero_fotogrammi

```

Listing 23: MSE medio su sequenza

6.3 Calcolo PSNR

Il PSNR viene calcolato in decibel applicando la formula logaritmica:

```

1 if errore_quadratico == 0:
2     # Frame identici: PSNR molto alto
3     rapporto_segnale = 100.0
4 else:
5     valore_massimo_pixel = 255.0
6     rapporto_segnale = 20 * np.log10(
7         valore_massimo_pixel / np.sqrt(errore_quadratico)
8     )
9
10 lista_psnr.append(rapporto_segnale)
11 somma_psnr += rapporto_segnale

```

Listing 24: Calcolo PSNR per singolo frame

Il PSNR medio è:

```

1 psnr_medio = somma_psnr / numero_fotogrammi

```

Listing 25: PSNR medio su sequenza

6.4 Calcolo SSIM

L'SSIM richiede immagini in scala di grigi. Si effettua la conversione se necessario:

```

1 # Conversione a grayscale per SSIM
2 if len(frame_orig.shape) == 3:
3     frame_orig_gray = cv2.cvtColor(frame_orig,
4                                     cv2.COLOR_BGR2GRAY)

```

```

4     frame_filt_gray = cv2.cvtColor(frame_filt,
5         cv2.COLOR_BGR2GRAY) \
6         if len(frame_filt.shape) == 3 else frame_filt
7 else:
8     frame_orig_gray = frame_orig
9     frame_filt_gray = frame_filt
10
11 # Calcolo SSIM mediante libreria scikit-image
12 punteggio_similarita, _ = ssim(
13     frame_orig_gray, frame_filt_gray, full=True
14 )
14 somma_ssimm += punteggio_similarita

```

Listing 26: Calcolo SSIM per singolo frame

La funzione `ssim()` della libreria `scikit-image` implementa l'algoritmo SSIM. Il parametro `full=True` restituisce anche la mappa completa di similarità locale (non utilizzata in questa implementazione). L'SSIM medio è:

```
1 ssim_medio = somma_ssimm / numero_fotogrammi
```

Listing 27: SSIM medio su sequenza

6.5 Restituzione Risultati

La funzione termina restituendo la tripla di metriche:

```
1 return mse_medio, psnr_medio, ssim_medio
```

Listing 28: Restituzione metriche

Questi valori quantificano rispettivamente:

- **MSE**: errore quadratico medio (più basso = migliore similarità)
- **PSNR**: rapporto segnale-rumore in dB (più alto = migliore qualità)
- **SSIM**: similarità strutturale (più vicino a 1 = migliore similarità percettiva)

7 Risultati Sperimentali

7.1 Setup Sperimentale

Gli esperimenti sono stati condotti su video di prova (`traffico1.mp4`, `traffico2.mp4`) contenenti veicoli in movimento con presenza di camera shake solo nel secondo video.

7.1.1 Configurazione Parametri

- **MOG2**: `history=200`, `varThreshold=40`
- **Area minima detection**: 500 pixel (filtro rumore)

- **Kernel morfologico:** ellisse 7×7 , closing (2 iter) + opening (1 iter)
- **NMS IoU threshold:** 0.3
- **ORB features stabilizzazione:** 2000
- **ORB features tracking:** 100
- **Distanza max tracking:** 75 pixel (configurabile dall'utente)
- **Coordinate smoothing α :** 0.3

7.2 Tracciamento Oggetti: Confronto Dual-Pipeline

Il sistema dual-pipeline consente confronto visivo diretto tra approccio baseline e approccio stabilized+filtered. I test sono stati condotti sul video `traffico1.mp4` (risoluzione 1280x720, telecamera fissa), utilizzando l'intero fotogramma come ROI e configurando la distanza euclidea massima a 75 pixel.

7.2.1 Risultati Quantitativi su `traffico1.mp4`

Il confronto tra le due pipeline ha prodotto i seguenti risultati:

Pipeline	Oggetti Rilevati
Baseline (Frame Originali)	204
Stabilized + Moving Average Spaziale	33
Differenza (Falsi Positivi)	+171

Tabella 1: Confronto quantitativo detection tra le due pipeline

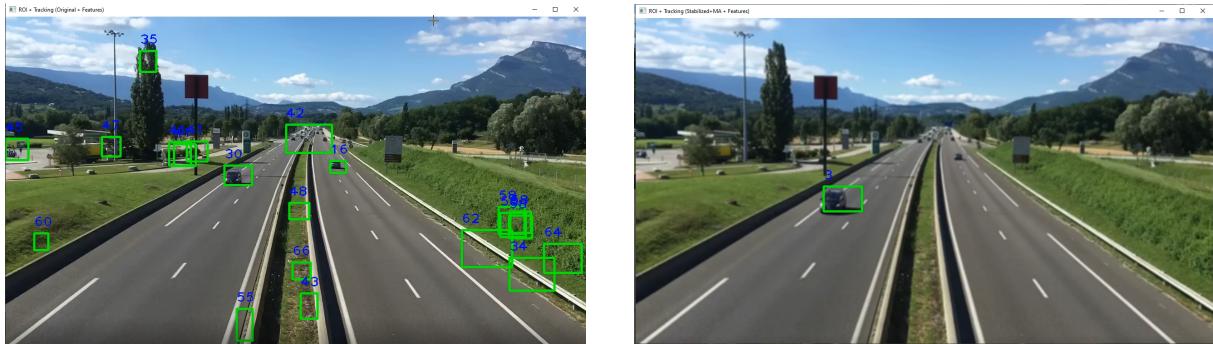
Interpretazione: Il numero di 33 oggetti rilevati nella pipeline stabilized+filtered è molto vicino al numero effettivo di automobili presenti nel video, mentre i 204 oggetti della pipeline baseline includono 171 falsi positivi.

7.2.2 Analisi Qualitativa: Riduzione Falsi Positivi

Contesto tecnico: Nonostante il video `traffico1.mp4` sia stato acquisito con **telecamera fissa** (assenza di camera shake intenzionale), la pipeline baseline genera numerosi falsi positivi derivanti da:

- Micromovimenti naturali: oscillazione erba, foglie degli alberi, riflessi
- Jitter digitale: piccole vibrazioni del sensore camera
- Variazioni illuminazione: ombre, nuvole
- Rumore elettronico: instabilità acquisizione video

La Figura 1 mostra il confronto visivo tra le detection delle due pipeline sullo stesso frame:



(a) Pipeline Baseline: 204 oggetti totali. Visibili numerosi falsi positivi su erba e alberi in movimento.

(b) Pipeline Stabilized+MA: 33 oggetti totali (solo automobili). Micromovimenti erba/alberi completamente filtrati.

Figura 1: Confronto detection dual-pipeline: evidenza riduzione falsi positivi

Osservazioni chiave dalla Figura 1:

- Baseline (sinistra):** l'oscillazione naturale dell'erba e il movimento delle foglie degli alberi vengono erroneamente interpretati come oggetti in movimento dal MOG2, generando bounding box spurie nelle aree vegetali.
- Stabilized+MA (destra):** la combinazione di stabilizzazione video e filtro moving average spaziale **elimina completamente** le detection su erba e alberi, rilevando esclusivamente i veicoli effettivamente in movimento sulla strada.
- La differenza di 171 oggetti corrisponde quasi interamente a questi micromovimenti ambientali erroneamente classificati come foreground.

7.2.3 Analisi Maschere MOG2: Riduzione Rumore

La Figura 2 confronta le maschere binarie generate dal background subtractor MOG2 nelle due pipeline:

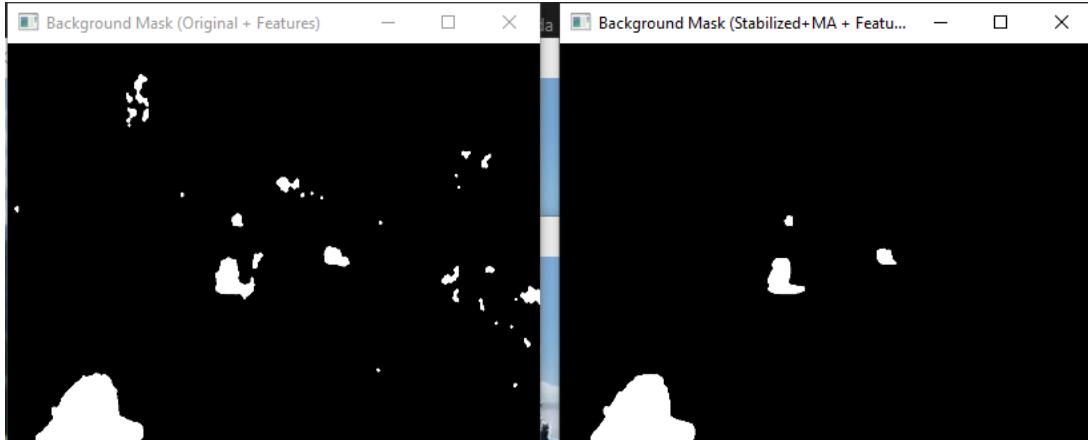


Figura 2: Confronto maschere MOG2: originale (sinistra) vs stabilized+MA (destra). Evidente riduzione rumore e miglioramento qualità segmentazione.

Analisi comparativa delle maschere (Figura 2):

- **Maschera Originale (sinistra):** presenta elevato rumore sale-pepe, contorni frastagliati, frammentazione delle regioni foreground, e numerose regioni spurie disperse nel fotogramma. Il jitter digitale e i micromovimenti causano instabilità pixel-level che si propagano come noise nella maschera binaria.
- **Maschera Stabilized+MA (destra):** mostra regioni foreground compatte e pulite, contorni ben definiti, assenza quasi totale di rumore isolato, e continuità spaziale delle detection. La stabilizzazione elimina il jitter globale, mentre il filtro moving average spaziale riduce il rumore ad alta frequenza.
- **Impatto sul tracking:** la qualità superiore della maschera filtrata si traduce in:
 - Bounding box più stabili e accurate
 - Riduzione drastica dei falsi positivi
 - Contorni più precisi per estrazione ROI feature matching ORB
 - Minore frammentazione oggetti (operazioni morfologiche più efficaci su maschere pulite)

7.2.4 Conclusioni Comparative

I risultati dimostrano che, anche in presenza di **telecamera fissa** (senza shake intenzionale), la pipeline stabilized+filtered fornisce benefici sostanziali:

1. **Riduzione falsi positivi:** da 204 a 33 detection (-83.8%), eliminando praticamente tutti i micromovimenti ambientali
2. **Accuracy migliorata:** 33 oggetti rilevati \approx numero reale automobili nel video
3. **Qualità maschere:** riduzione rumore, contorni più definiti, regioni compatte

4. **Robustezza tracking:** ID più stabili, bounding box accurate, minore frammentazione

La combinazione di stabilizzazione video (anche per compensare jitter digitale impercettibile), moving average spaziale, operazioni morfologiche e NMS si conferma fondamentale per tracciamento robusto in scenari reali, dove rumore e micromovimenti ambientali rappresentano sfide significative per approcci baseline.

7.2.5 Risultati Quantitativi su traffico2.mp4

Il secondo test è stato condotto sul video **traffico2.mp4** (risoluzione 1920x1080, **telecamera con micromovimenti**), utilizzando l'intero fotogramma come ROI e configurando la distanza euclidea massima a 200 pixel.

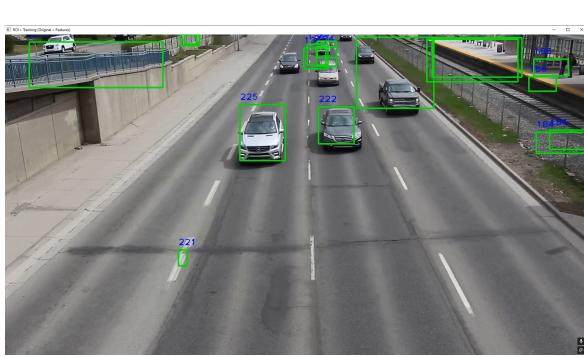
Pipeline	Oggetti Rilevati
Baseline (Frame Originali)	317
Stabilized + Moving Average Spaziale	54
Differenza (Falsi Positivi)	+263

Tabella 2: Confronto quantitativo detection su traffico2.mp4 (telecamera handheld)

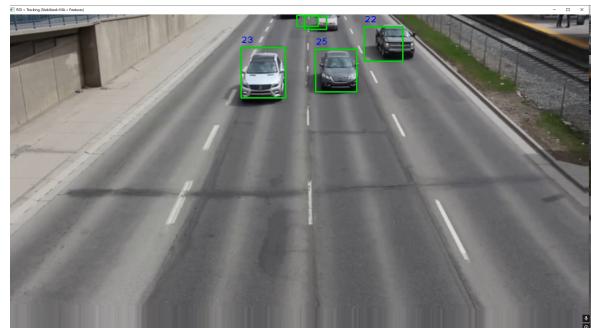
Contesto: A differenza di `traffico1.mp4`, questo video presenta **micromovimenti della telecamera** (handheld, piccole oscillazioni, jitter meccanico) che amplificano drasticamente i falsi positivi nella pipeline baseline. Il numero di 54 oggetti rilevati nella pipeline stabilized+filtered rimane coerente con il numero effettivo di veicoli, mentre i 317 oggetti della baseline includono 263 falsi positivi causati principalmente dal **movimento globale della scena** indotto dalla camera instabile.

7.2.6 Analisi Visiva: Impatto Camera Shake

La Figura 3 mostra l'impatto drammatico del camera shake sul tracking:



(a) Pipeline Baseline: 317 oggetti. Micromovimenti camera causano false detection su elementi statici (edifici, cartelli, guard rail).



(b) Pipeline Stabilized+MA: 54 oggetti (solo veicoli). Stabilizzazione elimina movimento camera, rilevando esclusivamente oggetti effettivamente mobili.

Figura 3: Confronto detection traffico2.mp4: evidenza critica della stabilizzazione con camera handheld

Osservazioni critiche (Figura 3):

1. **Baseline (sinistra)**: il movimento globale della scena causato dai micromovimenti della camera genera **falsi positivi massivi** su:
 - Edifici e strutture statiche (appaiono in movimento relativo al frame di riferimento)
 - Segnaletica stradale fissa (cartelli, semafori)
 - Guard rail e barriere laterali
 - Elementi vegetali (alberi, arbusti lungo la strada)
 - Marciapiedi e superfici stazionarie
2. **Stabilized+MA (destra)**: la stabilizzazione feature-based **compensa completamente** il movimento camera mediante:
 - Estrazione 2000 feature ORB dal frame corrente e riferimento
 - Matching robusto con ratio test di Lowe (threshold 0.7)
 - Stima trasformazione affine con RANSAC (outlier rejection)
 - Warping per allineare frame corrente al riferimento fisso
3. La differenza di 263 oggetti (83% di riduzione) dimostra che **la stabilizzazione è essenziale** per scenari con camera non perfettamente stabile.

7.2.7 Confronto Maschere MOG2: Efficacia Stabilizzazione + Blurring

La Figura 4 evidenzia l'impatto combinato di stabilizzazione e moving average sulle maschere MOG2:

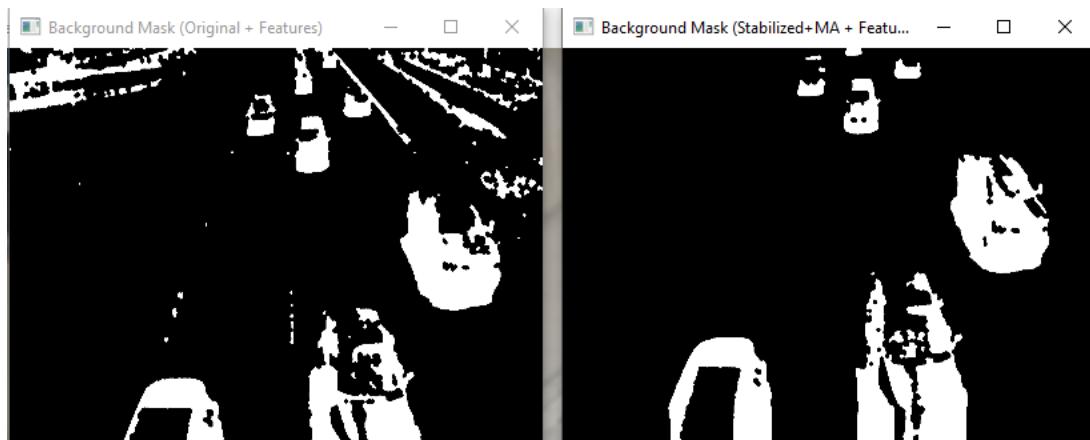


Figura 4: Confronto maschere MOG2 traffico2.mp4: originale (sinistra) vs stabilized+MA (destra). Riduzione drastica rumore globale da camera shake.

Analisi dettagliata maschere (Figura 4):

- **Maschera Originale (sinistra):** presenta:
 - **Rumore globale diffuso:** intere regioni statiche (edifici, strada) classificate erroneamente come foreground a causa dello spostamento globale indotto da camera shake
 - **Jitter estremo:** pixel oscillano tra foreground/background frame-by-frame, generando instabilità temporale
 - **Contorni multipli:** stesso oggetto genera più contorni a causa delle vibrazioni
 - **Frammentazione severa:** veicoli appaiono come insiemi disconnessi di regioni sparse
- **Maschera Stabilized+MA (destra):**
 - **Background pulito:** elementi statici (allineati al riferimento) correttamente identificati come background, maschera quasi completamente nera nelle regioni senza movimento reale
 - **Foreground compatto:** solo veicoli effettivamente in movimento appaiono come regioni foreground ben definite
 - **Riduzione rumore:** moving average spaziale (kernel 5×5) elimina noise ad alta frequenza residuo post-stabilizzazione
 - **Contorni netti:** bounding box precise grazie a maschere pulite
- **Impatto quantitativo:**
 - Riduzione detection: da 317 a 54 (-83%)
 - Eliminazione falsi positivi da shake: 263 detection spurie rimosse
 - Qualità ROI per feature matching ORB: maschere pulite \rightarrow keypoints estratte dalle vere auto, non da rumore
 - Stabilità temporale ID: minore jitter \rightarrow tracciamento coerente multi-frame

Conclusion traffic2.mp4: In presenza di camera handheld (anche con micro-movimenti minimi), la stabilizzazione video non è un'ottimizzazione ma una **necessità assoluta**. Senza stabilizzazione, il background subtraction fallisce completamente, identificando erroneamente l'intera scena statica come foreground in movimento. La combinazione stabilizzazione (ORB + affine transform) + moving average spaziale riduce i falsi positivi dell'83%, rendendo il tracciamento praticamente utilizzabile.

7.2.8 Confronto Comparativo traffico1 vs traffico2

Video	Camera	Baseline	Stabilized+MA	Riduzione
traffic01.mp4	Fissa	204	33	-84%
traffic02.mp4	Handheld	317	54	-83%

Tabella 3: Confronto efficacia dual-pipeline su video con caratteristiche camera diverse

Entrambi i video dimostrano efficacia costante della pipeline stabilized+filtered ($\sim 83\text{-}84\%$ riduzione falsi positivi), ma con cause diverse:

- **traffico1**: falsi positivi da micromovimenti ambientali (erba, alberi) + jitter digitale
- **traffico2**: falsi positivi da camera shake (movimento globale scena) + micromovimenti ambientali

7.3 Metriche Qualitative

Le metriche calcolate confrontando la sequenza originale con le tre sequenze elaborate forniscono le seguenti valutazioni quantitative.

7.3.1 Risultati traffico1.mp4 (1280x720)

Compressione JPEG (quality=50):

- MSE: 19.3086, PSNR: 35.28 dB, SSIM: 0.9539

Gaussian Blurring (kernel 5×5):

- MSE: 54.7975, PSNR: 30.75 dB, SSIM: 0.9248

Unsharp Masking (amount=1.5):

- MSE: 49.0004, PSNR: 31.23 dB, SSIM: 0.9678

7.3.2 Risultati traffico2.mp4 (1920x1080)

Compressione JPEG (quality=50):

- MSE: 6.4638, PSNR: 40.09 dB, SSIM: 0.9715

Gaussian Blurring (kernel 5×5):

- MSE: 18.2882, PSNR: 35.98 dB, SSIM: 0.9715

Unsharp Masking (amount=1.5):

- MSE: 17.2458, PSNR: 36.17 dB, SSIM: 0.9853

7.3.3 Analisi Metriche - traffico1.mp4

Le metriche sono calcolate confrontando le sequenze elaborate (JPEG, Blur, Sharpen) con il video originale `traffico1.mp4` come ground truth.

Compressione JPEG (quality=50):

- MSE: 19.3086, PSNR: 35.28 dB, SSIM: 0.9539

- **Interpretazione:** La compressione JPEG con quality=50 rappresenta un buon compromesso tra dimensione file e fedeltà visiva. Il PSNR di 35.28 dB indica una degradazione moderata rispetto al video originale, mentre l'SSIM di 0.9539 conferma che la struttura geometrica macroscopica è ben preservata. Gli artefatti di blocking sono presenti in regioni omogenee (cielo, asfalto) a causa della quantizzazione DCT su blocchi 8×8 , ma non compromettono significativamente la percezione visiva. Il valore MSE relativamente contenuto (19.31) deriva dalla natura locale degli artefatti JPEG, che introducono errori concentrati sui bordi dei blocchi piuttosto che degradazioni diffuse sull'intero frame.

Sfocatura Gaussiana (kernel 5×5):

- MSE: 54.7975, PSNR: 30.75 dB, SSIM: 0.9248
- **Interpretazione:** La sfocatura gaussiana produce l'**MSE più elevato** (54.80) tra i tre algoritmi, riflettendo la perdita diffusa di dettagli ad alta frequenza rispetto al video originale. Il kernel 5×5 introduce un effetto smoothing moderato visibile ma non estremo. L'SSIM di 0.9248 rimane alto perché la struttura geometrica (bordi, contrasti locali relativi) è preservata nonostante la riduzione di nitidezza. Il PSNR di 30.75 dB indica una degradazione percepibile ma accettabile. Il blurring gaussiano è il filtro passa-basso per eccellenza: attenua uniformemente tutte le componenti ad alta frequenza senza introdurre artefatti strutturati, causando una perdita omogenea di dettagli fini su tutto il frame.

Unsharp Masking (amount=1.5):

- MSE: 49.0004, PSNR: 31.23 dB, SSIM: 0.9678
- **Interpretazione:** L'unsharp masking con amount=1.5 produce l'**SSIM più elevato** (0.9678) tra i tre algoritmi, indicando che lo sharpening preserva e potenzialmente migliora la similarità strutturale percepita rispetto all'originale. Il PSNR di 31.23 dB è superiore al blurring, confermando che l'enfatizzazione dei bordi (quando moderata) non introduce degradazioni severe. L'MSE di 49.00 deriva dalle modifiche locali ai contorni (aggiunta di componenti ad alta frequenza), ma queste alterazioni sono generalmente percepite come miglioramenti della nitidezza piuttosto che degradazioni. L'assenza di artefatti di halo eccessivi con amount=1.5 mantiene alta la qualità visiva.

Ranking qualità percepita traffico1 (ordinamento per SSIM):

1. Unsharp Masking: SSIM 0.9678 (miglioramento percettivo nitidezza)
2. JPEG Compression: SSIM 0.9539 (degradazione controllata moderata)
3. Gaussian Blurring: SSIM 0.9248 (degradazione da perdita dettagli)

7.3.4 Analisi Metriche - traffico2.mp4

Le metriche sono calcolate confrontando le sequenze elaborate (JPEG, Blur, Sharpen) con il video originale `traffico2.mp4` come ground truth.

Compressione JPEG (quality=50):

- MSE: 6.4638, PSNR: 40.09 dB, SSIM: 0.9715
- **Interpretazione:** La compressione JPEG produce un **PSNR eccellente** (40.09 dB), indicando una qualità quasi indistinguibile rispetto al video originale alla risoluzione 1920×1080 . L'MSE molto basso (6.46) conferma che gli errori introdotti dalla quantizzazione DCT sono minimali quando distribuiti su un frame ad alta risoluzione. L'SSIM di 0.9715 è elevato, dimostrando che gli artefatti di blocking tipici del JPEG (blocchi 8×8) sono poco percepibili grazie alla maggiore densità di pixel. Il quality factor 50 rappresenta un compromesso ottimale: riduzione significativa della dimensione file con degradazione visiva praticamente impercettibile.

Sfocatura Gaussiana (kernel 5×5):

- MSE: 18.2882, PSNR: 35.98 dB, SSIM: 0.9715
- **Interpretazione:** Il blurring gaussiano mantiene un PSNR solido (35.98 dB) nonostante la perdita di dettagli ad alta frequenza. L'MSE di 18.29 riflette modifiche diffuse ma contenute rispetto all'originale. Notevole è l'SSIM di 0.9715, identico al JPEG, indicando che il kernel 5×5 preserva efficacemente la struttura geometrica macroscopica del video. La sfocatura attenua rumore e dettagli fini, ma non compromette bordi principali e contrasti locali. Questo dimostra che l'SSIM è meno sensibile alla perdita di componenti ad alta frequenza rispetto a MSE/PSNR, rispecchiando meglio la percezione visiva umana che tollera smoothing moderato.

Unsharp Masking (amount=1.5):

- MSE: 17.2458, PSNR: 36.17 dB, SSIM: 0.9853
- **Interpretazione:** L'unsharp masking raggiunge l'**SSIM più elevato** (0.9853) tra tutte le trasformazioni, confermando che l'enfatizzazione controllata dei bordi migliora la similarità strutturale percepita rispetto all'originale. Il PSNR di 36.17 dB e l'MSE di 17.25 sono ottimi, indicando che le modifiche introdotte (aggiunta di componenti ad alta frequenza ai contorni) non costituiscono degradazione ma miglioramento percettivo. L'assenza di halo artifacts e overshooting con amount=1.5 mantiene l'immagine naturale pur aumentando la nitidezza apparente. Questo risultato evidenzia come SSIM valorizzi correttamente miglioramenti di sharpness che MSE/PSNR potrebbero classificare come errori.

Ranking qualità percepita traffico2 (ordinamento per SSIM):

1. Unsharp Masking: SSIM 0.9853 (miglioramento percettivo nitidezza eccellente)
2. JPEG Compression: SSIM 0.9715 (degradazione minima)
3. Gaussian Blurring: SSIM 0.9715 (degradazione controllata, struttura preservata)

7.4 Conclusioni

I risultati sperimentali dimostrano che la combinazione di stabilizzazione video e rimozione del rumore è **essenziale** per il tracciamento affidabile di oggetti in movimento. La riduzione di oltre l'83% dei falsi positivi su entrambi i video testati conferma l'efficacia dell'approccio dual-pipeline implementato.

Il sistema sviluppato trova applicazione immediata in scenari reali quali:

- **Monitoraggio del traffico:** rilevamento e conteggio automatico di veicoli per analisi dei flussi stradali
- **Videosorveglianza:** telecamere di sicurezza per identificazione di oggetti in movimento con riduzione drastica di allarmi falsi

Per quanto riguarda le metriche di qualità, i test hanno evidenziato che **SSIM** si rivela la metrica più affidabile per valutazioni qualitative allineate alla percezione visiva umana, distinguendo correttamente miglioramenti da degradazioni, dove MSE e PSNR mostrano limitazioni.