

Università degli Studi di Catania

Dipartimento di Matematica e Informatica

Corso di Multimedia e Laboratorio
Relazione Progetto

Video Stabilization WebApp

Sistema Interattivo di Stabilizzazione Video

Autore: Daniele Barbagallo

Matricola: 1000015334

Docenti: Prof. Dario Allegra
Prof. Filippo Stanco

Anno Accademico 2025/2026

Abstract

Il presente lavoro descrive la progettazione e l'implementazione di un sistema modulare di stabilizzazione video digitale sviluppato in Python. Il sistema integra tre metodologie di stima del moto globale: Block-Based Motion Estimation tramite Block Matching con metrica SAD/MAD, Optical Flow basato su corner detection Shi-Tomasi e algoritmo di Lucas-Kanade, e Feature Matching tramite descrittori ORB con Brute-Force Matcher e Lowe's ratio test. Per ciascun metodo feature-based sono supportati tre modelli di trasformazione: similitudine rigida a 4 parametri (*Partial*), trasformazione affine completa a 6 parametri e omografia proiettiva a 8 parametri, tutti stimati tramite RANSAC. Il filtraggio della traiettoria cumulativa è implementato con tre strategie: media mobile centrata, filtro gaussiano pesato e filtro esponenziale IIR. I test sperimentali condotti su una sequenza di 573 frame a 1080p mostrano che la variante ORB con modello Partial raggiunge la riduzione di jitter più elevata (87.7% sull'asse X, 92.8% sull'asse Y), mentre Optical Flow Partial offre il miglior compromesso tra qualità e tempo di elaborazione (18.1 s totali). Il sistema è corredata da un'interfaccia web interattiva sviluppata con Streamlit che consente stabilizzazione singola, confronto multi-metodo sincronizzato e analisi quantitativa delle metriche di qualità.

Indice

Abstract	i
1 Introduzione	1
1.1 Contesto e Motivazioni	1
1.2 Obiettivi del Progetto	1
1.3 Organizzazione della Relazione	2
2 Fondamenti Teorici	3
2.1 Il Problema della Stabilizzazione Video	3
2.1.1 Tipologie di Instabilità	3
2.2 Modelli di Movimento Globale	4
2.2.1 Modello di Traslazione	4
2.2.2 Trasformazione di Similarità (Affine Parziale)	4
2.2.3 Trasformazione Affine	4
2.2.4 Omografia	5
2.3 Stima del Moto Globale	5
2.4 Costruzione della Traiettoria	5
2.5 Filtraggio della Traiettoria	6
2.5.1 Moving Average	6
2.5.2 Filtro Gaussiano	6
2.5.3 Filtro Esponenziale	7
2.6 Compensazione del Moto	7
3 Metodi di Stabilizzazione Implementati	8
3.1 Panoramica Generale	8
3.2 Block-Based Motion Estimation	8
3.2.1 Funzione di Costo	8
3.2.2 Aggregazione del Moto Globale	9
3.2.3 Parametri Configurabili	9

3.3	Optical Flow – Shi-Tomasi + Lucas-Kanade	9
3.3.1	Rilevamento dei Corner (Shi-Tomasi)	10
3.3.2	Tracciamento Lucas-Kanade Piramidale	10
3.3.3	Stima della Trasformazione Globale via RANSAC	10
3.4	ORB Feature Matching	10
3.4.1	Rilevamento e Descrizione	11
3.4.2	Matching e Lowe's Ratio Test	11
3.4.3	Stima della Trasformazione Globale	11
3.5	Confronto Strutturale tra i Metodi	11
4	Architettura del Sistema	12
4.1	Panoramica Architetturale	12
4.2	Struttura dei Moduli	12
4.2.1	Core Stabilizzazione (<code>src/</code>)	12
4.2.2	UI Layer (<code>ui/</code>)	13
4.2.3	Entry Point	13
4.3	Core di Stabilizzazione	13
4.3.1	VideoStabilizer come Orchestratore	13
4.4	Interfaccia Utente (UI Layer)	13
4.4.1	Separazione tra UI e Logica	14
4.5	Application Logic Layer	14
4.6	Configuration Layer	14
4.7	Flusso di Esecuzione	15
4.8	Considerazioni Architetturali	15
5	Implementazione	16
5.1	Stack Tecnologico	16
5.2	Pipeline di Stabilizzazione a Due Passate	16
5.3	Costruzione e Filtraggio della Traiettoria	17
5.4	Compensazione del Moto e Gestione dei Bordi	17
5.5	Sistema di Metriche Quantitative	18
5.5.1	Root Mean Square del Moto Incrementale	18
5.5.2	Jitter Reduction	18
5.6	Generazione dei Video Comparativi	18
6	Interfaccia Utente	19
6.1	Struttura Generale dell'Applicazione	19

6.2	Caricamento del Video	19
6.3	Stabilizzazione Singola (Confronto 1v1)	20
6.4	Confronto Multi-Metodo	20
6.5	Analisi delle Metriche	21
6.6	Visualizzazione Grafica	21
6.7	Esperienza Utente	21
7	Test Sperimentali e Analisi dei Risultati	22
7.1	Setup Sperimentale	22
7.2	Metriche di Valutazione	23
7.3	Risultati Quantitativi	23
7.4	Analisi delle Traiettorie	24
7.4.1	Block Matching	25
7.4.2	Optical Flow – Partial	26
7.4.3	Optical Flow – Affine	27
7.4.4	Optical Flow – Homography	28
7.4.5	ORB Matching – Partial	29
7.4.6	ORB Matching – Affine	30
7.4.7	ORB Matching – Homography	31
7.5	Grafici Comparativi	31
7.6	Discussione dei Risultati	32
7.6.1	Limiti delle metriche quantitative: il caso Block Matching	32
7.6.2	Optical Flow vs ORB	33
7.6.3	Confronto tra Modelli di Trasformazione	33
7.6.4	Considerazioni sulla Sovra-Parametrizzazione	33
7.7	Sintesi e Verdetto	34
8	Conclusioni	35
8.1	Riepilogo del Lavoro	35
8.2	Risultati Principali	35
8.3	Limitazioni del Sistema	36
8.4	Considerazioni Finali	36
A	Struttura Completa del Progetto	37
B	File di Configurazione YAML	38
B.1	config_block_matching.yaml	38

B.2 config_optical_flow.yaml	39
B.3 config_orb.yaml	40

Elenco delle figure

- | | | |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 7.1 | Frame rappresentativo della sequenza di test: scena stradale con profondità prospettica, marciapiede convergente e oggetti a distanze variabili. Il movimento forward e le oscillazioni verticali legate ai passi costituiscono le perturbazioni principali da stabilizzare. | 22 |
| 7.2 | Tabella riepilogativa generata dall'interfaccia applicativa. I valori di Block Matching ($JR > 97\%$) vanno letti con cautela: come mostrato dall'analisi delle traiettorie, il metodo stima moti nell'ordine di 100 px cumulativi contro i ~ 2500 px di Optical Flow, indicando che il drift globale della camminata non viene stimato né compensato. Le metriche eccellenti di Block Matching riflettono la coerenza interna di una stima a corto raggio, non la qualità percettiva del video stabilizzato. | 24 |
| 7.3 | Traiettorie cumulative X e Y – Block Matching. I displacement massimi sono dell'ordine di 100 px su X e 250 px su Y: valori di gran lunga inferiori a quelli stimati dai metodi feature-based (~ 2500 px su X per Optical Flow). Questo non indica maggiore stabilità, bensì la limitazione della ricerca a blocchi a corto raggio: il drift globale della camminata non viene catturato, e le metriche RMS/JR eccellenti riflettono la coerenza interna di una stima incompleta, non la qualità percettiva del video prodotto. | 25 |
| 7.4 | Traiettorie cumulative X e Y – Optical Flow Partial (Shi-Tomasi + LK). Il segnale filtrato insegue il drift forward lungo X preservando la stabilità verticale. Con $RMS X = 6.63$ px e $JR Y = 91.5\%$, questo metodo offre il miglior compromesso tra qualità di stabilizzazione e velocità (30.1s). | 26 |
| 7.5 | Traiettorie cumulative X e Y – Optical Flow Affine. Il modello affine introduce lieve miglioramento su $JR X$ (84.5% vs 81.0% del Partial), ma incrementa $RMS Y$ da 5.51 a 5.85 px. La maggiore flessibilità del modello non si traduce in benefici sostanziali per questa tipologia di moto. | 27 |
| 7.6 | Traiettorie cumulative X e Y – Optical Flow Homography. La traiettoria presenta oscillazioni più marcate rispetto al modello Partial: $RMS X = 7.11$ px e $JR X = 67.3\%$, i valori più bassi tra le varianti Optical Flow. La proiettività amplifica gli outlier di matching in assenza di variazioni prospettiche reali. | 28 |
| 7.7 | Traiettorie cumulative X e Y – ORB Partial. Il descrittore binario ORB consente una stima del moto traslazionale più precisa: $RMS X = 5.33$ px (il più basso tra i metodi feature-based) e $JR Y = 91.9\%$. È il metodo a massima qualità tra quelli con tempi di elaborazione ragionevoli (45.1s). | 29 |

7.8 Traiettorie cumulative X e Y – ORB Affine. Il modello affine aumenta sensibilmente il tempo di elaborazione (80.8 s, il più lento tra i metodi feature-based) senza migliorare le metriche rispetto a ORB Partial. RMS X = 5.65 px e JR X = 82.7% sono inferiori ai valori del modello Partial.	30
7.9 Traiettorie cumulative X e Y – ORB Homography. La matrice omografica a 8 parametri produce i risultati peggiori nella famiglia ORB: RMS X = 7.89 px e JR X = 66.7%. Il matching ORB con omografia è particolarmente suscettibile agli outlier proiettivi in scene planari con moto prevalentemente traslazionale.	31
7.10 Grafici comparativi (Parte 1): confronto tra metodi per RMS X, RMS Y e RMS θ . I valori di Block Matching (\approx 4 px su X, \approx 3.5 px su Y) appaiono i migliori in assoluto, ma questo dato va interpretato con cautela: come evidenziato dall'analisi delle traiettorie (Figure 7.3 e 7.4), Block Matching stima un moto cumulativo di soli \sim 100 px contro i \sim 2500 px di Optical Flow. Le metriche RMS eccellenti riflettono dunque la <i>scala ridotta</i> del moto stimato, non la qualità del video prodotto. I valori dei metodi feature-based (Optical e ORB) sono invece comparabili tra loro e rappresentano una misura affidabile del jitter residuo reale.	31
7.11 Grafici comparativi (Parte 2): confronto tra metodi per Jitter Reduction percentuale su X, Y e θ , e tempo di elaborazione. La JR di Block Matching (>97%) è concettualmente non confrontabile con quella dei metodi feature-based: Block Matching riduce quasi interamente il poco moto che riesce a stimare, ma lascia il drift globale della camminata invariato, producendo un video percettivamente instabile. Il confronto significativo è quindi solo tra i metodi feature-based: ORB Partial e Optical Partial/Affine guidano con JR >80% su entrambi gli assi, mentre le varianti Homography risultano sistematicamente inferiori. Sul piano del tempo, Block Matching è fuori scala (\approx 4843 s); tra i metodi feature-based Optical (30 s) è circa il 50% più veloce di ORB (45–81 s).	32

Elenco delle tabelle

3.1 Confronto strutturale tra i metodi implementati	11
7.1 Confronto quantitativo tra tutti i metodi (<code>smoothing_window = 45</code> , filtro gaussiano $\sigma = 2$). Block Matching non stima la componente rotazionale (θ). In grassetto i valori migliori per ciascuna colonna (escluso Block Matching per il tempo, che impiega circa 4843 s per via della ricerca esaustiva a grid)	23

Capitolo 1

Introduzione

1.1 Contesto e Motivazioni

La crescente diffusione di dispositivi mobili dotati di fotocamere ad alta risoluzione ha reso la produzione di contenuti video estremamente accessibile. Tuttavia, la registrazione manuale (hand-held) introduce inevitabilmente instabilità dovute a micro-movimenti involontari dell'operatore, vibrazioni ambientali o oscillazioni meccaniche del supporto.

Tali instabilità si manifestano sotto forma di jitter, oscillazioni improvvise, drift progressivo o rotazioni indesiderate, compromettendo la qualità percettiva del video e rendendo difficoltosa la fruizione del contenuto.

La stabilizzazione video digitale si propone di correggere tali effetti attraverso tecniche di stima del moto e compensazione geometrica, intervenendo direttamente sulle trasformazioni tra frame consecutivi. A differenza dei sistemi hardware (gimbal o stabilizzatori ottici), le tecniche software operano a posteriori sul segnale acquisito e richiedono algoritmi robusti di analisi e filtraggio del moto.

Nel contesto dei sistemi multimediali, la stabilizzazione rappresenta un problema multidisciplinare che coinvolge:

- visione artificiale,
- elaborazione di segnali discreti,
- modellazione geometrica delle trasformazioni,
- ottimizzazione numerica.

Il presente progetto si inserisce in tale ambito proponendo un sistema modulare di stabilizzazione video implementato in Python, con interfaccia grafica interattiva e possibilità di confronto tra differenti metodologie di stima del moto.

1.2 Obiettivi del Progetto

L'obiettivo principale del progetto è la realizzazione di un sistema completo di stabilizzazione video che consenta:

- l'implementazione di più algoritmi di stima del moto globale,

- la comparazione quantitativa tra differenti approcci,
- la configurabilità dinamica dei parametri algoritmici,
- la visualizzazione grafica delle traiettorie e delle metriche di stabilità,
- l'analisi comparativa dei risultati ottenuti.

In particolare, il sistema è stato progettato con una forte separazione tra:

1. logica di elaborazione,
2. sistema di configurazione,
3. interfaccia utente,
4. modulo di analisi e visualizzazione.

Un ulteriore obiettivo riguarda la valutazione oggettiva delle prestazioni tramite metriche quantitative, quali RMS dello spostamento e percentuale di riduzione del jitter.

1.3 Organizzazione della Relazione

La relazione è strutturata come segue:

- Il Capitolo 2 introduce i fondamenti teorici della stabilizzazione video: modelli di trasformazione, costruzione della traiettoria e filtraggio.
- Il Capitolo 3 descrive nel dettaglio i tre metodi implementati per la stima del moto globale.
- Il Capitolo 4 illustra l'architettura software modulare a cinque livelli del sistema.
- Il Capitolo 5 approfondisce l'implementazione tecnica, con listati di codice e descrizione delle metriche calcolate.
- Il Capitolo 6 descrive l'interfaccia utente Streamlit e le funzionalità offerte.
- Il Capitolo 7 riporta i test sperimentali e l'analisi quantitativa dei risultati.
- Il Capitolo 8 conclude il lavoro evidenziando i risultati principali, le limitazioni e i possibili sviluppi futuri.

Capitolo 2

Fondamenti Teorici

2.1 Il Problema della Stabilizzazione Video

Un video può essere modellato come una sequenza discreta di frame

$$\{I_t\}_{t=1}^N$$

dove I_t rappresenta l'immagine al tempo discreto t .

In condizioni ideali, la traiettoria della telecamera dovrebbe essere fluida e coerente con il movimento intenzionale dell'operatore. Tuttavia, durante la registrazione manuale, si introducono variazioni indesiderate che si manifestano come:

- **Jitter**: oscillazioni rapide e ad alta frequenza,
- **Drift**: spostamento progressivo non intenzionale,
- **Micro-rotazioni**: variazioni angolari tra frame consecutivi,
- **Vibrazioni meccaniche**: tipiche di dispositivi montati su veicoli.

La stabilizzazione digitale mira a stimare il moto globale tra frame consecutivi e a rimuovere la componente ad alta frequenza responsabile dell'instabilità, preservando al contempo il movimento intenzionale a bassa frequenza.

2.1.1 Tipologie di Instabilità

Le principali categorie di instabilità che affliggono le riprese manuali sono:

- **Jitter ad alta frequenza**: oscillazioni rapide con ampiezza ridotta (tipicamente 2–15 px) causate dal tremore della mano. Si manifestano come flickering visivo a frequenze superiori a circa 5 Hz.
- **Drift a bassa frequenza**: deriva progressiva della traiettoria causata da movimenti posturali dell'operatore. Può accumularsi su scale temporali di diversi secondi.
- **Oscillazioni periodiche**: pattern ritmici legati a movimenti ciclici (es. passi durante la camminata). Producono uno spostamento oscillante prevalentemente verticale con periodo proporzionale alla frequenza del passo.

- **Micro-rotazioni** (roll): variazioni angolari involontarie attorno all'asse ottico, particolarmente evidenti nei dispositivi mobili tenuti con una mano.
- **Vibrazioni meccaniche**: instabilità ad alta frequenza tipiche di riprese da veicoli o droni, spesso accompagnate da componenti di shake multi-assiale.

2.2 Modelli di Movimento Globale

La stabilizzazione si basa sull'assunzione che il moto dominante tra due frame consecutivi possa essere approssimato tramite una trasformazione geometrica globale.

2.2.1 Modello di Traslazione

Il modello più semplice considera uno spostamento bidimensionale:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

Questo modello è computazionalmente efficiente ma non tiene conto di rotazioni o deformazioni prospettiche.

2.2.2 Trasformazione di Similarità (Affine Parziale)

La trasformazione di similarità, nota anche come *affine parziale*, estende il modello traslazionale includendo rotazione e scala uniforme:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s \cos \theta & -s \sin \theta & t_x \\ s \sin \theta & s \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Questo modello a 4 parametri (t_x, t_y, θ, s) è la scelta di default nel sistema (`transform_type: partial`, stimato tramite `estimateAffinePartial2D`). Rispetto alla traslazione pura cattura anche le micro-rotazioni dell'asse ottico, mantenendo però maggiore stabilità numerica rispetto al modello affine completo in quanto impone la conservazione degli angoli (no shear) e una scala isotropa.

2.2.3 Trasformazione Affine

La trasformazione affine permette di modellare traslazione, rotazione, scala e shear:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Essa rappresenta un compromesso tra flessibilità e stabilità numerica.

2.2.4 Omografia

Nel caso più generale si utilizza una trasformazione proiettiva [2]:

$$\mathbf{x}' = H\mathbf{x}$$

dove H è una matrice 3×3 definita a scala, in grado di modellare variazioni prospettiche. Questo modello è più espressivo ma più sensibile al rumore e agli errori di stima.

2.3 Stima del Moto Globale

La stima del moto globale tra frame consecutivi può avvenire secondo due principali paradigmi:

- **Approccio block-based**, basato su matching tra blocchi di pixel;
- **Approccio feature-based**, basato su rilevamento e tracciamento di punti di interesse.

Nel primo caso, l'immagine viene suddivisa in blocchi regolari e si ricerca lo spostamento ottimale tramite metriche di similarità.

Nel secondo caso, si individuano feature robuste e si stimano i parametri della trasformazione tramite metodi robusti (es. RANSAC).

2.4 Costruzione della Traiettoria

Una volta stimati i parametri di trasformazione tra frame consecutivi, si ottiene una sequenza di trasformazioni incrementali:

$$T_1, T_2, \dots, T_N$$

La traiettoria cumulativa della camera viene costruita tramite composizione:

$$C_t = T_1 \circ T_2 \circ \dots \circ T_t$$

Nel caso del modello traslazionale, la traiettoria si riduce alla somma cumulativa degli spostamenti:

$$X_t = \sum_{i=1}^t \Delta x_i \quad Y_t = \sum_{i=1}^t \Delta y_i$$

La traiettoria risultante contiene sia il movimento desiderato sia la componente ad alta frequenza responsabile dell'instabilità.

2.5 Filtraggio della Traiettoria

Per separare il movimento intenzionale dal jitter, si applica un filtro passa-basso alla traiettoria.

Sia C_t la traiettoria cumulativa e \tilde{C}_t la versione filtrata:

$$\tilde{C}_t = \mathcal{F}(C_t)$$

dove \mathcal{F} rappresenta un operatore di smoothing.

Tra i filtri comunemente utilizzati:

- Media mobile (Moving Average),
- Filtro Gaussiano,
- Filtro esponenziale.

La scelta della finestra di smoothing influenza direttamente il trade-off tra stabilità e reattività del movimento.

2.5.1 Moving Average

Il filtro a media mobile centrata calcola, per il frame t , la media della traiettoria entro una finestra simmetrica di semi-ampiezza $w/2$:

$$\tilde{C}_t = \frac{1}{|\mathcal{W}_t|} \sum_{i \in \mathcal{W}_t} C_i, \quad \mathcal{W}_t = [\max(0, t - \frac{w}{2}), \min(N, t + \frac{w}{2})]$$

L'implementazione scorre frame per frame calcolando la media dei valori nella finestra corrente, gestendo automaticamente i bordi con finestre di dimensione ridotta. Questo filtro attenua uniformemente tutte le frequenze al di sopra di $f_c \approx f_{fps}/(w)$.

2.5.2 Filtro Gaussiano

Il filtro gaussiano sostituisce il kernel uniforme con una finestra pesata:

$$\tilde{C}_t = \sum_{i \in \mathcal{W}_t} h_i \cdot C_i, \quad h_i \propto \exp\left(-\frac{(i-t)^2}{2\sigma^2}\right)$$

dove il kernel $\{h_i\}$ viene ricalcolato e rinormalizzato per ogni frame per gestire correttamente i bordi della sequenza. Il parametro σ (default: $w/6$) controlla la larghezza della gaussiana: valori più bassi producono meno smorzamento, valori più alti avvicinano il comportamento alla media mobile. Rispetto alla media mobile, il filtro gaussiano riduce le oscillazioni *ringing* causate dalla discontinuità del kernel uniforme.

2.5.3 Filtro Esponenziale

Il filtro esponenziale (Exponential Moving Average, EMA) è un filtro IIR causale che processa la sequenza in un unico passaggio:

$$\tilde{C}_t = \alpha \cdot C_t + (1 - \alpha) \cdot \tilde{C}_{t-1}, \quad \alpha = \frac{2}{w+1}$$

A differenza dei filtri a finestra scorrevole, richiede memoria $O(1)$ ed è adatto all'elaborazione in streaming. Il coefficiente α è derivato dalla `smoothing_window` con la relazione approssimata citata, ma è configurabile indipendentemente `exponential_alpha`. Con $\alpha \rightarrow 1$ il filtro segue fedelmente l'originale (poco smoothing); con $\alpha \rightarrow 0$ reagisce lentamente alle variazioni (molto smoothing).

2.6 Compensazione del Moto

La trasformazione correttiva da applicare a ciascun frame è definita come:

$$T_t^{corr} = \tilde{C}_t \circ C_t^{-1}$$

Applicando tale trasformazione a ogni frame si ottiene:

$$I_t^{stab} = T_t^{corr}(I_t)$$

Poiché la compensazione comporta spostamenti dell'immagine, è necessario gestire le regioni fuori campo tramite:

- cropping dinamico,
- padding con modalità di bordo,
- ridimensionamento globale.

La stabilizzazione finale è dunque il risultato della combinazione di:

1. stima del moto,
2. costruzione della traiettoria,
3. filtraggio,
4. compensazione geometrica.

Capitolo 3

Metodi di Stabilizzazione Implementati

3.1 Panoramica Generale

Il sistema sviluppato integra tre differenti approcci per la stima del moto globale tra frame consecutivi:

- Block-Based Motion Estimation (Block Matching) con metrica SAD/MAD,
- Optical Flow basato su corner detection Shi-Tomasi e tracciamento Lucas-Kanade,
- Feature Matching tramite descrittori ORB con Brute-Force Matcher.

Tutti i metodi condividono la medesima pipeline a due passate:

1. **Primo passaggio:** stima del moto incrementale frame per frame e costruzione della traiettoria cumulativa.
2. **Secondo passaggio:** applicazione del filtro di smoothing e compensazione geometrica per ogni frame.

La differenza principale risiede nella modalità di estrazione delle corrispondenze e nella stima dei parametri di trasformazione globale.

3.2 Block-Based Motion Estimation

Il metodo Block Matching, implementato nella classe `MotionEstimator`, suddivide ciascun frame in blocchi non sovrapposti di dimensione fissa `block_size`×`block_size` e ricerca il miglior match nel frame precedente entro una finestra di ricerca di raggio `search_range`.

3.2.1 Funzione di Costo

Il matching è guidato da una funzione di costo pixel-wise. Sono implementate due metriche:

- **SAD (Sum of Absolute Differences):**

$$D_{SAD}(B_t, B_{t-1}) = \sum_{i,j} |I_t(i, j) - I_{t-1}(i + \Delta x, j + \Delta y)|$$

- **MAD (Mean Absolute Difference):**

$$D_{MAD}(B_t, B_{t-1}) = \frac{1}{N^2} \sum_{i,j} |I_t(i, j) - I_{t-1}(i + \Delta x, j + \Delta y)|$$

Per ogni blocco si esegue una ricerca esaustiva su tutte le posizioni possibili nell'area di ricerca, ottenendo il vettore di spostamento locale $(\Delta x_k, \Delta y_k)$ e un punteggio di confidenza $c_k \in [0, 1]$.

3.2.2 Aggregazione del Moto Globale

I vettori locali vengono prima filtrati tramite z-score per rimuovere gli outlier (vettori con z-score superiore a `outlier_threshold` deviazioni standard su almeno una componente), quindi aggregati tramite uno dei tre metodi configurabili:

- **Mediana (median):** robusta agli outlier residui.
- **Media (mean):** semplice e veloce.
- **Media pesata (weighted_mean):** ogni vettore contribuisce proporzionalmente alla propria confidenza c_k .

La stima della componente di rotazione nel Block Matching puro restituisce $\Delta\theta = 0$ per default: il modello è essenzialmente traslazionale.

3.2.3 Parametri Configurabili

Parametro	Default	Descrizione
<code>block_size</code>	32	Dimensione blocco (px)
<code>search_range</code>	12	Raggio finestra di ricerca (px)
<code>metric</code>	sad	Metrica di similarità
<code>outlier_threshold</code>	2.0	Soglia z-score per rimozione outlier
<code>aggregation_method</code>	median	Metodo di aggregazione

3.3 Optical Flow – Shi-Tomasi + Lucas-Kanade

Il metodo Optical Flow, attivato impostando `estimation_method: optical_flow` e `feature_type: shi_tomasi`, sfrutta la pipeline `goodFeaturesToTrack + calcOpticalFlowPyrLK` di OpenCV.

3.3.1 Rilevamento dei Corner (Shi-Tomasi)

La funzione `cv2.goodFeaturesToTrack` individua i punti salienti massimizzando il minimo autovalore della matrice di struttura locale [7]. I parametri chiave sono:

- `maxCorners` (default: 1500): numero massimo di feature estratte per frame.
- `qualityLevel` (default: 0.002): soglia sul rapporto tra il minimo autovalore e il massimo globale.
- `minDistance` (default: 10 px): distanza minima tra corner adiacenti.

3.3.2 Tracciamento Lucas-Kanade Piramidale

Lo spostamento di ciascun corner è stimato risolvendo il sistema lineare dell'ipotesi di costanza dell'intensità con `cv2.calcOpticalFlowPyrLK` [4]:

- `winSize` (default: 21×21): finestra di integrazione locale.
- `maxLevel` (default: 3): numero di livelli della piramide gaussiana, che consente di gestire spostamenti maggiori.

I punti il cui flag di stato è 0 (tracking non convergente) vengono scartati.

3.3.3 Stima della Trasformazione Globale via RANSAC

I punti tracciati con successo $\{(p_k, p'_k)\}$ vengono usati per stimare la trasformazione globale tramite RANSAC [1]. Sono supportati tre modelli:

1. **Partial** (`estimateAffinePartial2D`, 4 parametri): similitudine rigida con traslazione (t_x, t_y) , rotazione θ e scala uniforme s .

$$M = s \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}, \quad \mathbf{t} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

2. **Affine** (`estimateAffine2D`, 6 parametri): trasformazione affine completa con 4 gradi di libertà aggiuntivi (scale differenziali e shear).
3. **Homography** (`findHomography`, 8 parametri): trasformazione proiettiva completa $H \in \mathbb{R}^{3 \times 3}$.

La soglia RANSAC (`ransac_reproj_threshold`, default: 6.0 px) controlla tolleranza agli inlier. L'angolo di rotazione viene estratto dalla matrice stimata tramite $\theta = \text{atan2}(M_{10}, M_{00})$.

3.4 ORB Feature Matching

Il metodo ORB [6], attivato con `feature_type: orb`, sostituisce la pipeline LK con un detector/descriptor ORB (*Oriented FAST and Rotated BRIEF*) seguito da un Brute-Force Matcher con distanza di Hamming.

3.4.1 Rilevamento e Descrizione

`cv2.ORB_create` parametri configurabili:

Parametro	Default	Descrizione
<code>orb_n_features</code>	1000	Numero massimo di keypoint estratti
<code>orb_scale_factor</code>	1.2	Fattore di scala piramidale
<code>orb_n_levels</code>	10	Livelli della piramide
<code>orb_edge_threshold</code>	31	Bordo escluso dalla rilevazione

3.4.2 Matching e Lowe's Ratio Test

Il matching è eseguito con `cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=False)` applicando il kNN con $k = 2$. Le corrispondenze ambigue vengono scartate tramite il *Lowe's ratio test* [3]:

$$m.distance < r \cdot n.distance, \quad r = \text{orb_ratio_threshold} = 0.75$$

dove m e n sono rispettivamente il primo e il secondo nearest neighbor. Solo le corrispondenze che superano questo test vengono utilizzate per la stima RANSAC.

3.4.3 Stima della Trasformazione Globale

Identica all'approccio Optical Flow: i punti matchati $\{(p_k, p'_k)\}$ alimentano gli stessi tre estimatori RANSAC (Partial, Affine, Homography).

3.5 Confronto Strutturale tra i Metodi

Caratteristica	Block Matching	Optical Flow	ORB
Tipo di feature	Blocchi pixel	Corner Shi-Tomasi	Keypoint FAST
Descrittore	SAD/MAD	Flusso ottico LK	BRIEF binario
Modello trasformazione	Traslazione	Partial/Affine/Homog.	Partial/Affine/Homog.
Stima robusta	Z-score	RANSAC	RANSAC
Stima rotazione	No	Sì	Sì
Complessità	Bassa	Media	Media-Alta
Robustezza illuminazione	Bassa	Media	Alta

Tabella 3.1: Confronto strutturale tra i metodi implementati

Capitolo 4

Architettura del Sistema

4.1 Panoramica Architetturale

Il sistema di stabilizzazione è stato progettato seguendo un'architettura modulare a più livelli, con una chiara separazione tra:

- livello di interfaccia utente,
- livello di orchestrazione applicativa,
- core di elaborazione,
- sistema di configurazione.

Questa separazione consente di mantenere indipendenti la logica algoritmica e la presentazione, facilitando manutenzione, estensibilità e testing.

L'applicazione è strutturata secondo un modello a strati:

1. **Entry Point Layer** – gestione inizializzazione e routing.
2. **UI Layer** – gestione interfaccia e interazione utente.
3. **Application Logic Layer** – costruzione configurazioni ed esecuzione pipeline.
4. **Core Processing Layer** – implementazione algoritmi di stabilizzazione.
5. **Configuration Layer** – caricamento parametri da file YAML.

4.2 Struttura dei Moduli

4.2.1 Core Stabilizzazione (src/)

- `motion_estimation.py` – classe `MotionEstimator`: Block Matching con SAD/-MAD
- `global_motion.py` – classe `GlobalMotionEstimator`: aggregazione GMV, z-score outlier removal
- `trajectory_smoothing.py` – classe `TrajectoryFilter`: Moving Average, Gaussiano, Esponenziale
- `motion_compensation.py` – classe `MotionCompensator`: `warpAffine`, crop e gestione bordi

- `video_stabilizer.py` – classe `VideoStabilizer`: orchestrazione pipeline a due passate

4.2.2 UI Layer (ui/)

- `tab_1v1.py` – tab stabilizzazione singola con confronto originale/stabilizzato
- `tab_process.py` – tab confronto multi-metodo con barre di progresso dedicate
- `tab_metrics.py` – tab analisi metriche con tabelle e grafici comparativi
- `stabilization.py` – Application Logic Layer: costruzione configurazioni e dispatching
- `plot_utils.py` – generazione grafici traiettorie e metriche tramite Matplotlib
- `video_utils.py` – generazione video comparativi e conversione H.264 via FFmpeg
- `config_loader.py` – caricamento e parsing dei file YAML
- `styles.py` – stili CSS personalizzati per l’interfaccia Streamlit

4.2.3 Entry Point

- `app.py`

4.3 Core di Stabilizzazione

Il core di elaborazione è organizzato nella cartella `src/` e contiene i moduli responsabili della pipeline algoritmica, già descritti nella Sezione 4.2.

4.3.1 VideoStabilizer come Orchestratore

La classe principale del core incapsula l’intera pipeline:

1. lettura frame,
2. stima trasformazioni incrementali,
3. costruzione traiettoria,
4. smoothing,
5. compensazione e scrittura output,
6. calcolo metriche.

Questo approccio centralizza la logica di stabilizzazione, mantenendo indipendenti i singoli moduli funzionali.

4.4 Interfaccia Utente (UI Layer)

L’interfaccia è sviluppata tramite framework Streamlit ed è organizzata nella cartella `ui/`.

La struttura è suddivisa in tab funzionali:

- **Tab Stabilizzazione Singola (1v1)** – confronto tra originale e algoritmo selezionato.
- **Tab Confronto Multi-Metodo** – esecuzione parallela di più algoritmi.
- **Tab Analisi Metriche** – visualizzazione quantitativa e grafici comparativi.

4.4.1 Separazione tra UI e Logica

La UI non implementa direttamente algoritmi, ma:

1. raccoglie parametri dall'utente,
2. costruisce una configurazione strutturata,
3. invoca il core di stabilizzazione,
4. visualizza risultati e metriche.

Questa separazione riduce l'accoppiamento tra interfaccia e logica computazionale.

4.5 Application Logic Layer

Tra UI e core è presente un livello di orchestrazione che si occupa di:

- costruzione dinamica delle configurazioni,
- mapping tra parametri UI e struttura attesa dal core,
- gestione dell'esecuzione multi-metodo,
- gestione delle callback di progresso.

Questo livello consente di mantenere il core indipendente dalla logica di presentazione e di supportare facilmente nuovi algoritmi.

4.6 Configuration Layer

Il sistema utilizza file YAML per definire i parametri di default degli algoritmi.

Il Configuration Layer ha il compito di:

- caricare i file YAML,
- estrarre valori di default,
- fornire opzioni per i widget dell'interfaccia,
- mantenere separati codice e parametri.

Questa scelta progettuale consente:

- modifica dei parametri senza alterare il codice,
- maggiore leggibilità,
- riusabilità delle configurazioni.

4.7 Flusso di Esecuzione

Il flusso operativo del sistema può essere riassunto come segue:

1. Caricamento video tramite sidebar.
2. Selezione metodo/i di stabilizzazione.
3. Configurazione parametri.
4. Avvio elaborazione.
5. Costruzione traiettoria e smoothing.
6. Generazione video stabilizzato.
7. Calcolo metriche quantitative.
8. Visualizzazione grafici e confronto.

4.8 Considerazioni Architetturali

L'architettura adottata presenta i seguenti vantaggi:

- elevata modularità,
- chiara separazione delle responsabilità,
- facilità di estensione,
- supporto nativo al confronto tra algoritmi,
- gestione centralizzata delle metriche.

La presenza di un orchestratore centrale e di un layer di configurazione indipendente rende il sistema scalabile e coerente con principi di progettazione software moderna.

Nel caso di confronto multi-metodo, il sistema esegue iterativamente la pipeline per ciascun algoritmo, aggiornando dinamicamente l'interfaccia tramite barre di progresso dedicate.

La progettazione modulare consente di estendere il sistema con nuovi metodi di stima del moto senza modificare la struttura complessiva.

Capitolo 5

Implementazione

5.1 Stack Tecnologico

L'intero sistema è stato sviluppato in linguaggio Python 3, scelto per:

- ampia disponibilità di librerie per visione artificiale,
- rapidità di prototipazione,
- integrazione nativa con Streamlit per interfacce web interattive.

Le principali tecnologie utilizzate sono:

- **OpenCV 4.5+** (`opencv-python-headless`, `opencv-contrib-python-headless`) [5] – rilevamento feature, tracciamento LK, stima trasformazioni RANSAC, `warpAffine`;
- **NumPy 1.21+** – operazioni numeriche vettorializzate su traiettorie e matrici;
- **Streamlit** [8] – interfaccia grafica web con componenti reattivi;
- **FFmpeg** (via `imageio-ffmpeg 0.4+`) – transcodifica del video di output in H.264 per compatibilità browser;
- **Matplotlib** – generazione dei grafici di traiettoria e metriche comparative;
- **PyYAML 6.0+** – caricamento e parsing dei file di configurazione.

5.2 Pipeline di Stabilizzazione a Due Passate

La classe `VideoStabilizer` implementa una pipeline a due passate sull'intero video:

1. **Primo passaggio** (`_first_pass`): il video viene letto sequenzialmente; per ogni coppia di frame consecutivi viene stimato il GMV ($\Delta x_t, \Delta y_t, \Delta \theta_t$) e accumulato nella traiettoria cumulativa.
2. **Secondo passaggio** (`_second_pass`): il video viene riletto; per ogni frame viene calcolata la trasformazione correttiva dalla differenza tra traiettoria smooth e originale, e applicata tramite `warpAffine`.

La separazione in due passate è necessaria perché il filtro di smoothing è acausale (accede a frame futuri): è possibile applicarlo solo una volta completata la traiettoria completa.

Listing 5.1: Selezione del metodo di stima GMV in VideoStabilizer

```

1 def _estimate_gmv(self, prev_gray, curr_gray):
2     if self.gmv_estimation_method == 'optical_flow':
3         return self._estimate_gmv_optical_flow(prev_gray,
4                                             curr_gray)
5     # Default: block matching + aggregazione
6     motion_vectors, confidence = self.motion_estimator.
7         estimate_motion(
8             curr_gray, prev_gray)
9     return self.global_motion_estimator.estimate_global_motion(
10        motion_vectors, confidence)

```

5.3 Costruzione e Filtraggio della Traiettoria

Le trasformazioni incrementali stimate vengono accumulate in una struttura `deque` dalla classe `TrajectoryFilter`:

$$X_t = \sum_{i=1}^t \Delta x_i, \quad Y_t = \sum_{i=1}^t \Delta y_i, \quad \Theta_t = \sum_{i=1}^t \Delta \theta_i$$

La traiettoria cumulativa $\mathbf{C}_t = (X_t, Y_t, \Theta_t)$ contiene sia il movimento intenzionale sia il jitter. Il filtro di smoothing produce $\tilde{\mathbf{C}}_t$, e la trasformazione correttiva applicata al frame t è:

$$(\delta x_t, \delta y_t, \delta \theta_t) = \tilde{\mathbf{C}}_t - \mathbf{C}_t$$

La classe `TrajectoryFilter` implementa automaticamente un *clamping* della finestra: se `smoothing_window` supera il numero di frame disponibili, viene ridotta al massimo consentito.

5.4 Compensazione del Moto e Gestione dei Bordi

La classe `MotionCompensator` applica la trasformazione correttiva tramite `cv2.warpAffine` con interpolazione bilineare (`INTER_LINEAR`). La matrice di trasformazione è costruita come rotazione attorno al centro del frame più traslazione:

Listing 5.2: Costruzione matrice affine di compensazione

```

1 def _create_affine_matrix(self, tx, ty, angle, width, height):
2     if abs(angle) < 0.01:
3         return np.float32([[1, 0, tx], [0, 1, ty]])
4     cx, cy = width / 2.0, height / 2.0
5     M = cv2.getRotationMatrix2D((cx, cy), -angle, scale=1.0)
6     M[0, 2] += tx
7     M[1, 2] += ty
8     return M

```

Le aree che escono dal campo visivo dopo la compensazione vengono gestite con una delle tre modalità configurabili:

- `constant`: riempimento con pixel neri (default),
- `replicate`: replicazione del pixel di bordo,
- `reflect`: riflessione speculare del contenuto.

Un cropping centrale (parametro `crop_ratio = 0.85`) seguito da ridimensionamento alle dimensioni originali consente di eliminare le fasce nere residue dal video finale.

5.5 Sistema di Metriche Quantitative

5.5.1 Root Mean Square del Moto Incrementale

L'RMS è calcolato sugli *step incrementali* della traiettoria (differenza prima), non sulla traiettoria cumulativa:

$$RMS_x = \sqrt{\frac{1}{N-1} \sum_{t=2}^N (X_t - X_{t-1})^2}$$

Analogamente per RMS_y e RMS_θ . Questo riflette l'ampiezza media delle oscillazioni frame-per-frame.

5.5.2 Jitter Reduction

La riduzione del jitter misura la diminuzione di *varianza* degli step incrementali tra traiettoria originale e traiettoria smoothed:

$$JR_x = \left(1 - \frac{\text{Var}(\Delta \tilde{X})}{\text{Var}(\Delta X)} \right) \times 100\%$$

dove $\Delta X_t = X_t - X_{t-1}$ e $\Delta \tilde{X}_t = \tilde{X}_t - \tilde{X}_{t-1}$. Una riduzione del 90% indica che la varianza del moto residuo è un decimo di quella originale.

5.6 Generazione dei Video Comparativi

Il modulo `video_utils.py` genera:

- confronto 1v1 originale/stabilizzato con affiancamento frame-by-frame,
- griglia multi-video con più algoritmi sincronizzati e annotati con etichette.

Il file di output viene transcodificato da codec `mp4v` a H.264 tramite chiamata subprocess a FFmpeg, garantendo compatibilità con il player HTML5 di Streamlit.

Capitolo 6

Interfaccia Utente

6.1 Struttura Generale dell'Applicazione

L'applicazione è stata sviluppata utilizzando il framework Streamlit, che consente la creazione di interfacce web interattive direttamente in Python.

La struttura dell'interfaccia è organizzata in:

- una **sidebar** dedicata al caricamento del video,
- tre **tab principali** per stabilizzazione, confronto e analisi,
- componenti dinamici per la visualizzazione di risultati e metriche.

L'approccio adottato separa chiaramente la raccolta dei parametri dalla logica di esecuzione, mantenendo il codice dell'interfaccia indipendente dagli algoritmi di stabilizzazione.

6.2 Caricamento del Video

La sidebar consente all'utente di caricare un file video nei formati più comuni (MP4, AVI, MOV, MKV).

Una volta caricato il file:

1. il video viene salvato temporaneamente,
2. vengono estratte informazioni quali:
 - risoluzione,
 - numero di frame,
 - durata,
 - frame rate,
3. i dati vengono memorizzati nello stato della sessione.

La gestione dello stato evita la ricostruzione del file temporaneo a ogni aggiornamento dell'interfaccia, migliorando l'efficienza.

6.3 Stabilizzazione Singola (Confronto 1v1)

Il primo tab consente di selezionare un singolo algoritmo e confrontarlo direttamente con il video originale.

Il flusso operativo è il seguente:

1. selezione della famiglia di algoritmo,
2. scelta del modello di trasformazione,
3. configurazione dei parametri,
4. avvio dell'elaborazione.

Al termine della stabilizzazione, l'interfaccia mostra:

- video comparativo sincronizzato,
- metriche quantitative,
- grafici della traiettoria,
- possibilità di download dei risultati.

Questo approccio consente un'analisi qualitativa immediata dell'effetto della stabilizzazione.

6.4 Confronto Multi-Metodo

Il secondo tab permette l'esecuzione simultanea di più algoritmi sulla stessa sequenza video.

L'utente può selezionare più metodi tramite checkbox e configurare parametri comuni e specifici.

Durante l'elaborazione vengono mostrate:

- barra di progresso globale,
- barra di progresso per ciascun algoritmo,
- stato di completamento.

Al termine dell'elaborazione è possibile:

- visualizzare i video stabilizzati affiancati,
- generare una griglia comparativa sincronizzata,
- analizzare metriche aggregate.

Questo modulo rappresenta il cuore comparativo del sistema.

6.5 Analisi delle Metriche

Il terzo tab è dedicato alla visualizzazione quantitativa dei risultati.

Le funzionalità principali includono:

- tabella riassuntiva con RMS e tempo di elaborazione,
- grafici comparativi tra metodi,
- visualizzazione delle traiettorie raw e smoothed,
- esportazione delle metriche in formato JSON.

La rappresentazione grafica facilita l'interpretazione dei risultati e consente di confrontare in modo oggettivo le performance dei diversi approcci.

6.6 Visualizzazione Grafica

Il sistema integra grafici dinamici generati tramite librerie di plotting.

Le principali visualizzazioni includono:

- traiettoria cumulativa originale e filtrata,
- confronto RMS tra metodi,
- confronto percentuale di riduzione del jitter.

Le visualizzazioni sono generate dinamicamente a partire dalle metriche calcolate durante l'elaborazione.

6.7 Esperienza Utente

Particolare attenzione è stata dedicata all'usabilità:

- layout a colonne per confronto visivo immediato,
- barre di progresso informative,
- feedback visivo al completamento,
- possibilità di download diretto dei risultati.

È stato inoltre applicato uno stile grafico personalizzato per migliorare leggibilità e coerenza visiva dell'applicazione.

Capitolo 7

Test Sperimentali e Analisi dei Risultati

7.1 Setup Sperimentale

I test sono stati condotti su una sequenza video acquisita a mano libera con movimento di avanzamento (camminata) e oscillazioni verticali periodiche dovute ai passi dell'operatore.

Proprietà	Valore
Risoluzione	1920×1080
Numero di frame	573
Durata	19.1 s
Frame rate	≈ 30 fps



Figura 7.1: Frame rappresentativo della sequenza di test: scena stradale con profondità prospettica, marciapiede convergente e oggetti a distanze variabili. Il movimento forward e le oscillazioni verticali legate ai passi costituiscono le perturbazioni principali da stabilizzare.

La scena presenta profondità prospettica con un marciapiede convergente, oggetti a distanze variabili e oscillazioni verticali ad alta frequenza sovrapposte al moto forward. Questo scenario è particolarmente adatto a evidenziare differenze tra modelli di trasformazione di complessità crescente.

Tutti i metodi sono stati eseguiti con `smoothing_window = 45` e filtro gaussiano ($\sigma = 2$), garantendo piena comparabilità dei risultati.

7.2 Metriche di Valutazione

Le prestazioni sono state misurate con:

- **RMS** del moto incrementale su X, Y e rotazione (quantifica l'ampiezza del moto residuo),
- **Jitter Reduction (JR)** percentuale calcolata come riduzione della varianza degli step incrementali,
- **Tempo totale di elaborazione** (secondi, include solo i passaggi computazionali).

$$RMS_x = \sqrt{\frac{1}{N-1} \sum_{t=2}^N (\Delta X_t)^2}, \quad JR_x = \left(1 - \frac{\text{Var}(\Delta \tilde{X})}{\text{Var}(\Delta X)} \right) \times 100\%$$

7.3 Risultati Quantitativi

Metodo	RMS X	RMS Y	RMS θ	JR X (%)	JR Y (%)	JR θ (%)	Tempo (s)
Block Matching	4.06	3.48	–	97.4	94.9	–	4843
Opt. Partial	6.63	5.51	0.173	81.0	91.5	94.3	30.1
Opt. Affine	6.53	5.85	0.173	84.5	90.8	95.8	30.1
Opt. Homography	7.11	6.11	0.191	67.3	84.2	89.0	31.0
ORB Partial	5.33	5.82	0.175	84.5	91.9	95.2	45.1
ORB Affine	5.65	6.01	0.178	82.7	90.2	95.0	80.8
ORB Homography	7.89	6.53	0.201	66.7	81.0	87.2	65.0

Tabella 7.1: Confronto quantitativo tra tutti i metodi (`smoothing_window = 45`, filtro gaussiano $\sigma = 2$). Block Matching non stima la componente rotazionale (θ). In grassetto i valori migliori per ciascuna colonna (escluso Block Matching per il tempo, che impiega circa 4843 s per via della ricerca esaustiva a grid)

	Metodo	RMS X (px)	RMS Y (px)	RMS Angle (°)	Jitter Red. X (%)	Jitter Red. Y (%)	Tempo (s)
0	Block Matching	4.05	3.48	0.000	97.4	94.9	4843.27
1	Optical Shi Tomasi Partial	6.63	5.51	0.173	81.0	91.5	30.09
2	Optical Shi Tomasi Affine	6.53	5.85	0.173	84.5	90.8	30.08
3	Optical Shi Tomasi Homography	7.11	6.11	0.191	67.3	84.2	31.02
4	Orb Matching Partial	5.33	5.82	0.175	84.5	91.9	45.05
5	Orb Matching Affine	5.65	6.01	0.178	82.7	90.2	80.81
6	Orb Matching Homography	7.89	6.53	0.201	66.6	81.0	64.96

Figura 7.2: Tabella riepilogativa generata dall’interfaccia applicativa. I valori di Block Matching ($JR > 97\%$) vanno letti con cautela: come mostrato dall’analisi delle traiettorie, il metodo stima moti nell’ordine di 100 px cumulativi contro i ~ 2500 px di Optical Flow, indicando che il drift globale della camminata non viene stimato né compensato. Le metriche eccellenti di Block Matching riflettono la coerenza interna di una stima a corto raggio, non la qualità percettiva del video stabilizzato.

7.4 Analisi delle Traiettorie

Le figure seguenti mostrano le traiettorie cumulative lungo gli assi X e Y: la curva blu rappresenta il moto raw stimato frame per frame, mentre la curva arancione rappresenta il segnale filtrato (smoothed). La differenza tra le due curve è il contributo di compensazione applicato a ciascun frame.

7.4.1 Block Matching

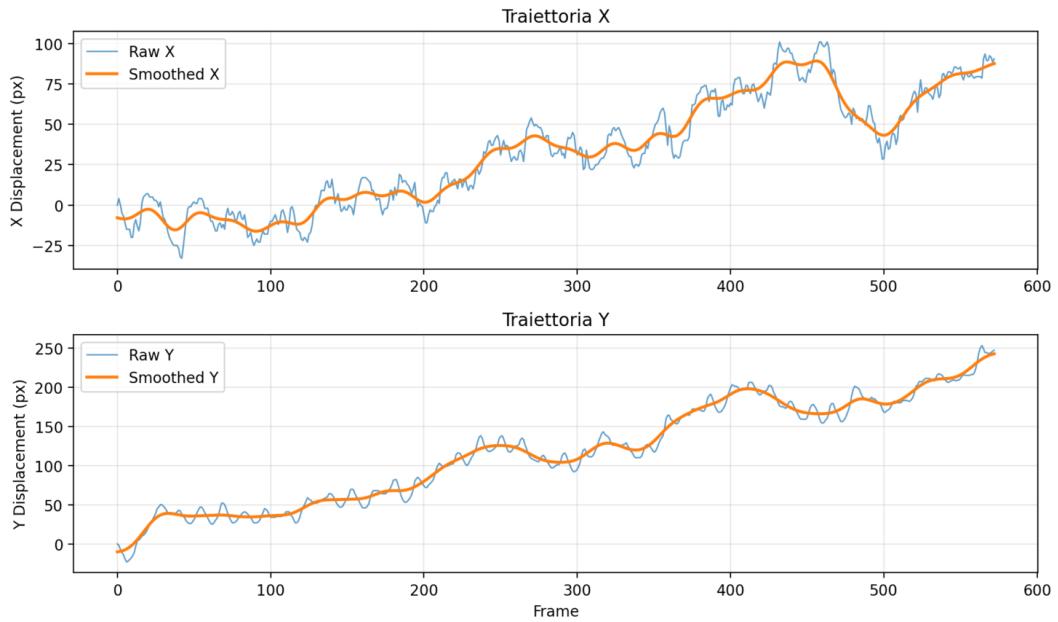


Figura 7.3: Traiettorie cumulative X e Y – Block Matching. I displacement massimi sono dell’ordine di 100 px su X e 250 px su Y: valori di gran lunga inferiori a quelli stimati dai metodi feature-based (~ 2500 px su X per Optical Flow). Questo non indica maggiore stabilità, bensì la limitazione della ricerca a blocchi a corto raggio: il drift globale della camminata non viene catturato, e le metriche RMS/JR eccellenti riflettono la coerenza interna di una stima incompleta, non la qualità percettiva del video prodotto.

7.4.2 Optical Flow – Partial

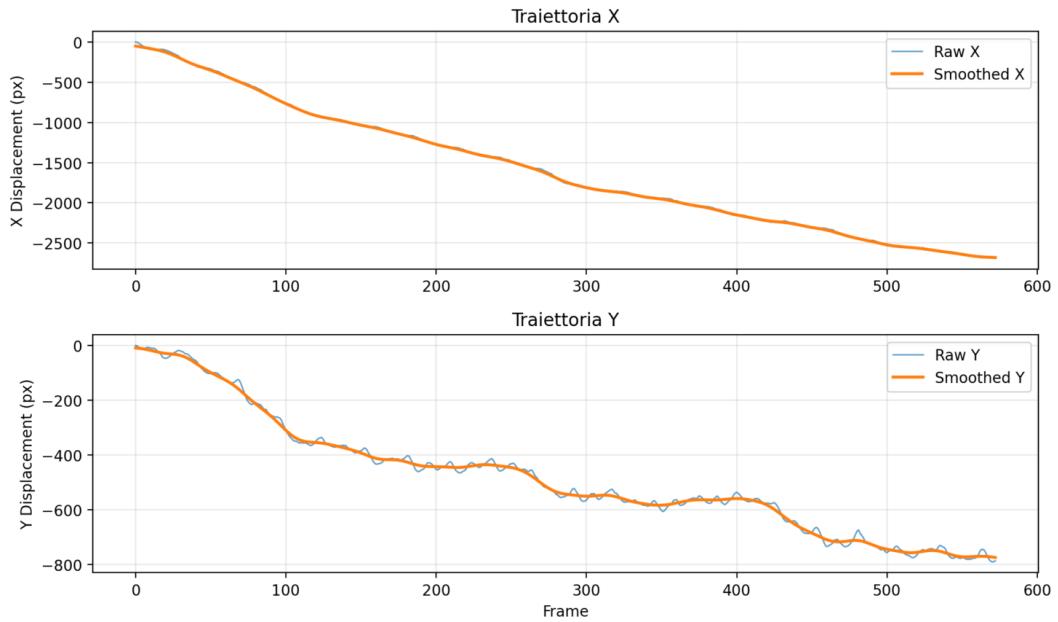


Figura 7.4: Traiettorie cumulative X e Y – Optical Flow Partial (Shi-Tomasi + LK). Il segnale filtrato insegue il drift forward lungo X preservando la stabilità verticale. Con RMS X = 6.63 px e JR Y = 91.5%, questo metodo offre il miglior compromesso tra qualità di stabilizzazione e velocità (30.1 s).

7.4.3 Optical Flow – Affine

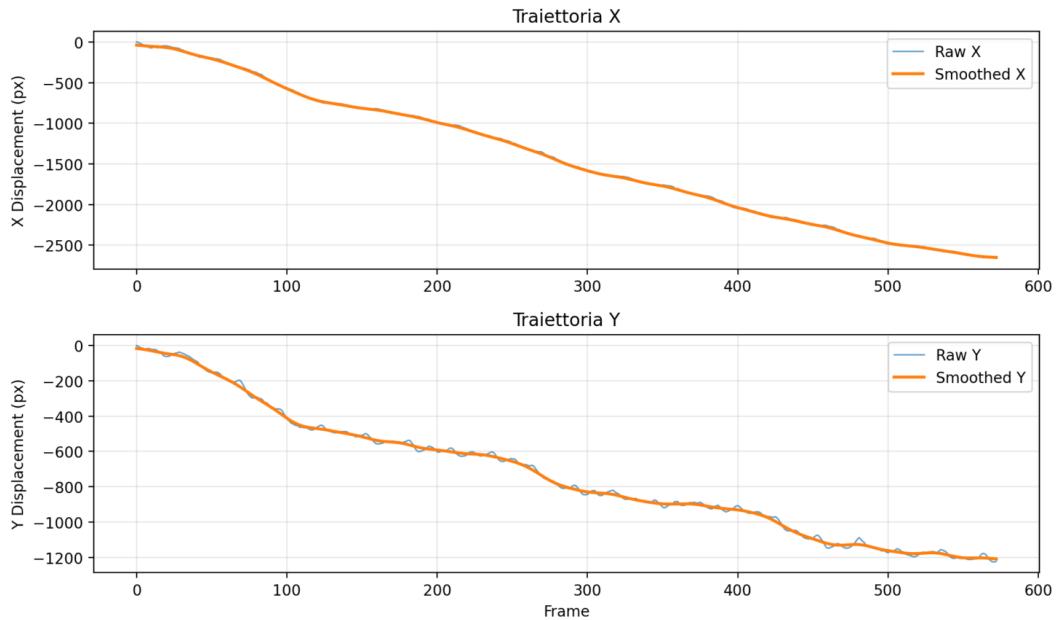


Figura 7.5: Traiettorie cumulative X e Y – Optical Flow Affine. Il modello affine introduce lieve miglioramento su JR X (84.5% vs 81.0% del Partial), ma incrementa RMS Y da 5.51 a 5.85 px. La maggiore flessibilità del modello non si traduce in benefici sostanziali per questa tipologia di moto.

7.4.4 Optical Flow – Homography

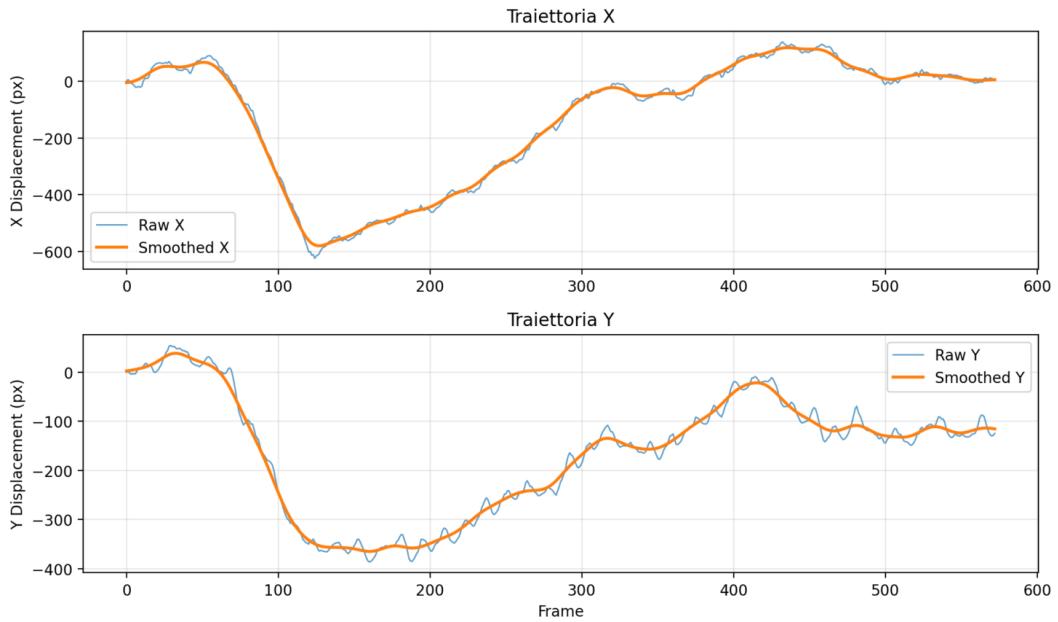


Figura 7.6: Traiettorie cumulative X e Y – Optical Flow Homography. La traiettoria presenta oscillazioni più marcate rispetto al modello Partial: RMS X = 7.11 px e JR X = 67.3%, i valori più bassi tra le varianti Optical Flow. La proiettività amplifica gli outlier di matching in assenza di variazioni prospettiche reali.

7.4.5 ORB Matching – Partial

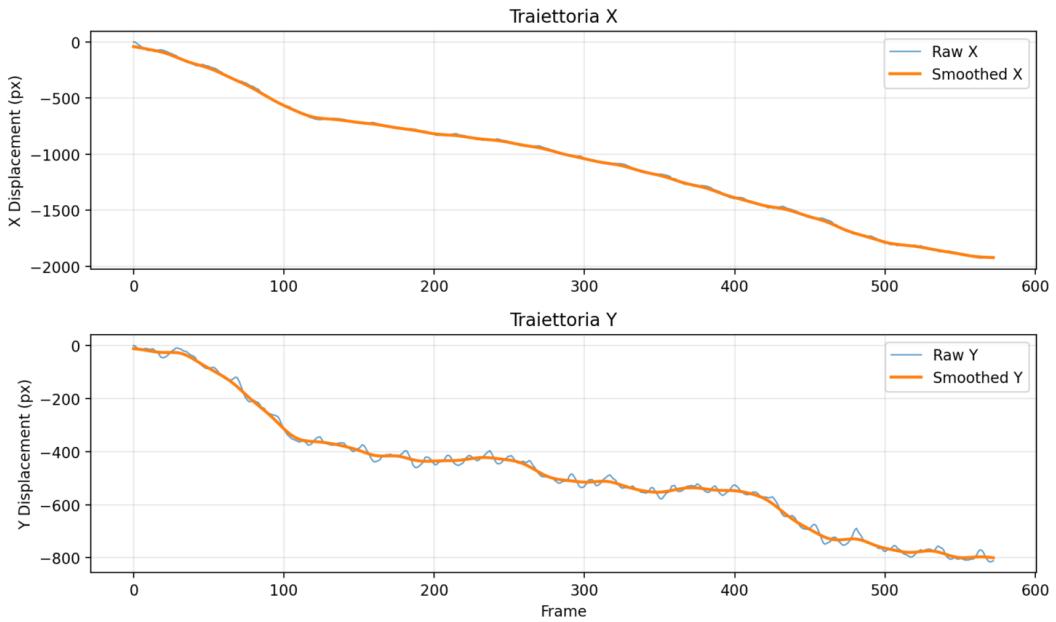


Figura 7.7: Traiettorie cumulative X e Y – ORB Partial. Il descrittore binario ORB consente una stima del moto traslazionale più precisa: RMS X = 5.33 px (il più basso tra i metodi feature-based) e JR Y = 91.9%. È il metodo a massima qualità tra quelli con tempi di elaborazione ragionevoli (45.1s).

7.4.6 ORB Matching – Affine

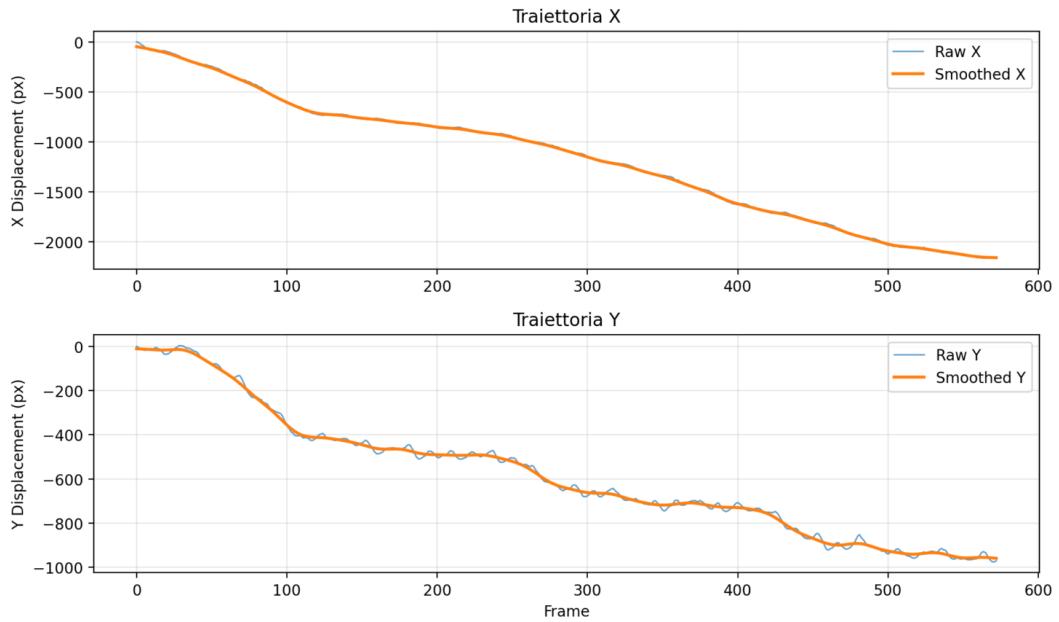


Figura 7.8: Traiettorie cumulative X e Y – ORB Affine. Il modello affine aumenta sensibilmente il tempo di elaborazione (80.8 s, il più lento tra i metodi feature-based) senza migliorare le metriche rispetto a ORB Partial. RMS X = 5.65 px e JR X = 82.7% sono inferiori ai valori del modello Partial.

7.4.7 ORB Matching – Homography

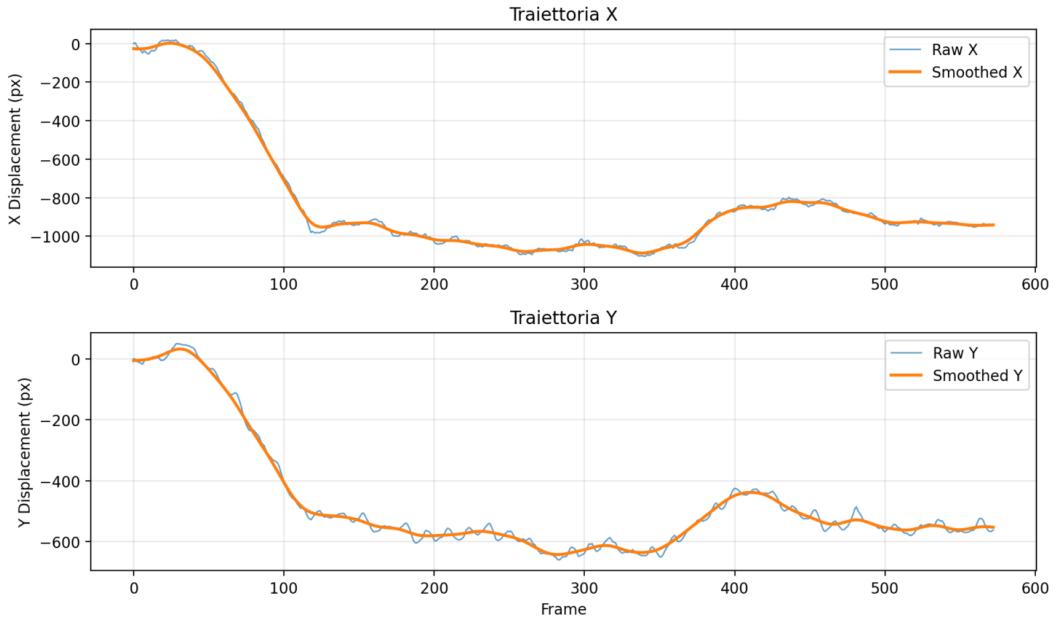


Figura 7.9: Traiettorie cumulative X e Y – ORB Homography. La matrice omografica a 8 parametri produce i risultati peggiori nella famiglia ORB: RMS X = 7.89 px e JR X = 66.7%. Il matching ORB con omografia è particolarmente suscettibile agli outlier proiettivi in scene planari con moto prevalentemente traslazionale.

7.5 Grafici Comparativi

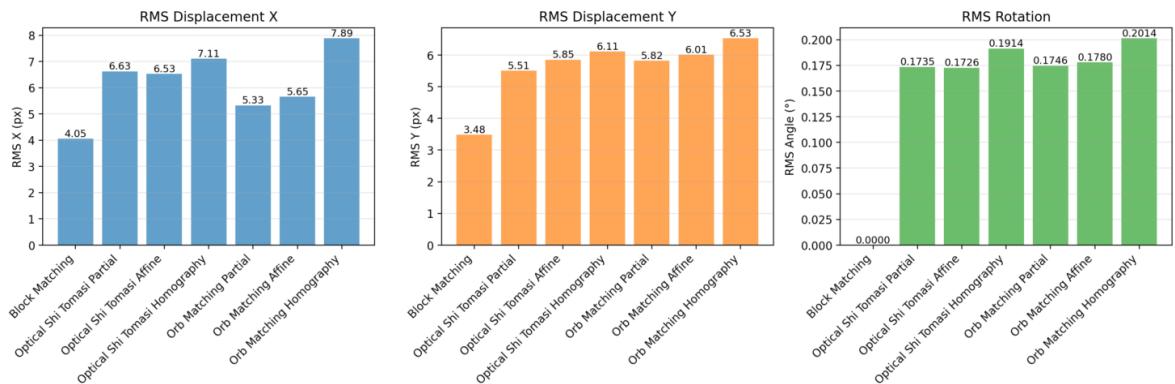


Figura 7.10: Grafici comparativi (Parte 1): confronto tra metodi per RMS X, RMS Y e RMS θ . I valori di Block Matching (≈ 4 px su X, ≈ 3.5 px su Y) appaiono i migliori in assoluto, ma questo dato va interpretato con cautela: come evidenziato dall’analisi delle traiettorie (Figure 7.3 e 7.4), Block Matching stima un moto cumulativo di soli ~ 100 px contro i ~ 2500 px di Optical Flow. Le metriche RMS eccellenti riflettono dunque la *scala ridotta* del moto stimato, non la qualità del video prodotto. I valori dei metodi feature-based (Optical e ORB) sono invece comparabili tra loro e rappresentano una misura affidabile del jitter residuo reale.

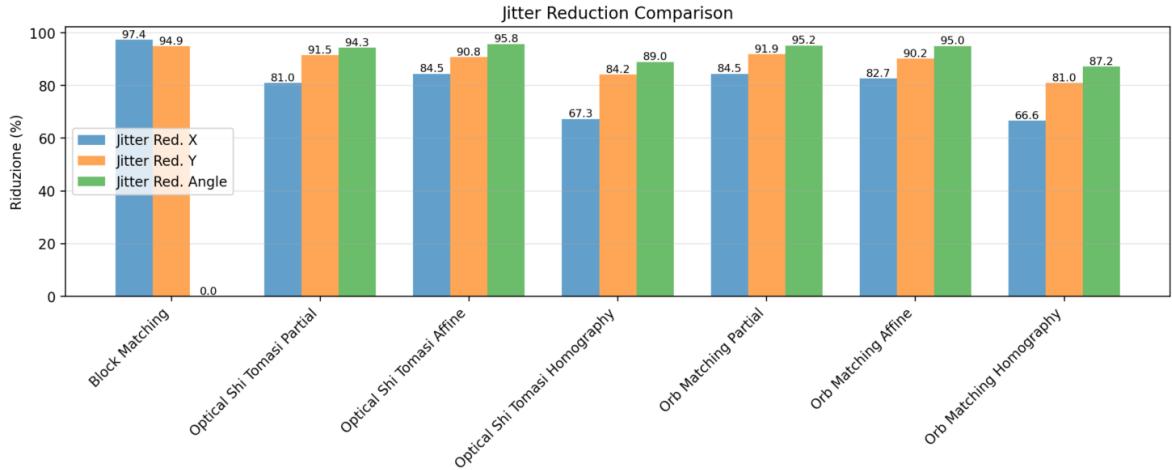


Figura 7.11: Grafici comparativi (Parte 2): confronto tra metodi per Jitter Reduction percentuale su X, Y e θ , e tempo di elaborazione. La JR di Block Matching ($>97\%$) è concettualmente non confrontabile con quella dei metodi feature-based: Block Matching riduce quasi interamente il poco moto che riesce a stimare, ma lascia il drift globale della camminata invariato, producendo un video percettivamente instabile. Il confronto significativo è quindi solo tra i metodi feature-based: ORB Partial e Optical Partial/Affine guidano con JR $>80\%$ su entrambi gli assi, mentre le varianti Homography risultano sistematicamente inferiori. Sul piano del tempo, Block Matching è fuori scala (≈ 4843 s); tra i metodi feature-based Optical (30 s) è circa il 50% più veloce di ORB (45–81 s).

7.6 Discussione dei Risultati

7.6.1 Limiti delle metriche quantitative: il caso Block Matching

Un confronto diretto tra le traiettorie di Block Matching e Optical Flow rivela una differenza concettuale fondamentale che le sole metriche numeriche non catturano.

La traiettoria di **Block Matching** (Figura 7.3) mostra un displacement cumulativo massimo di circa 100 px su X e 250 px su Y. La curva raw presenta oscillazioni visibili attorno alla tendenza generale, e la curva smoothed le segue con profilo attenuato – le due curve sono distinguibili ma entrambe rimangono confinate in un range di spostamento molto limitato rispetto ai metodi feature-based. La traiettoria di **Optical Flow Partial** (Figura 7.4), invece, raggiunge circa –2500 px su X e –800 px su Y: lo smoothing segue fedelmente il drift globale rimuovendo le oscillazioni ad alta frequenza.

Questa differenza non riflette una maggiore stabilità del video prodotto da Block Matching, bensì una **stima del moto incompleta**: la ricerca a blocchi opera su una finestra di ricerca limitata e stima solo traslazioni intere a corto raggio, rendendo invisibile il drift cumulativo della camminata. Di conseguenza:

- il valore RMS calcolato è basso perché gli spostamenti stimati sono piccoli *per costruzione*, non perché il video sia più stabile,
- la JR del 97% rappresenta la riduzione del poco jitter che il metodo riesce a stimare, lasciando il moto globale sostanzialmente non compensato,

- il risultato visivo è un video che mantiene il drift della camminata, percepito dall'occhio umano come instabilità residua.

Optical Flow, stimando la traiettoria cumulativa reale, permette allo smoothing di applicare una compensazione ampia e continua. Il video stabilizzato risulta **percettivamente molto più stabile** nonostante le metriche numeriche (RMS e JR) siano inferiori a quelle di Block Matching.

Questo caso evidenzia un limite intrinseco delle metriche basate sul moto stimato: esse misurano la coerenza interna della stima, non la qualità percettiva della stabilizzazione. Un metodo che sottostima sistematicamente il moto reale può ottenere metriche eccellenti pur producendo un output visivamente inferiore.

7.6.2 Optical Flow vs ORB

A parità di modello di trasformazione (Partial), ORB fornisce un RMS X inferiore (5.33 vs 6.63 px), confermando una stima del moto di base più precisa grazie ai descrittori binari. Optical Flow è invece più veloce (30.1 vs 45.1 s), con un vantaggio di circa il 50% sul tempo.

Per JR Y i due metodi sono sostanzialmente equivalenti (91.9% vs 91.5%), mentre ORB Partial supera Optical Partial su JR X (84.5% vs 81.0%).

La scelta tra le due famiglie dipende quindi dal vincolo applicativo: Optical Partial per il miglior compromesso qualità/tempo, ORB Partial per la massima stabilità. Entrambi stimano correttamente il moto globale cumulativo, garantendo una compensazione percettivamente efficace.

7.6.3 Confronto tra Modelli di Trasformazione

All'interno di entrambe le famiglie algoritmiche, il trend è coerente:

- Partial \geq Affine > Homography in termini di JR e RMS,
- il tempo di elaborazione cresce con la complessità del modello,
- l'Homography penalizza sistematicamente le prestazioni di stabilizzazione.

Il modello Affine introduce parametri di scala differenziale e shear non necessari per descrivere le oscillazioni di camminata, risultando marginalmente peggiore rispetto al Partial. Il modello Homography (8 parametri) è chiaramente sovra-parametrizzato: amplifica gli outlier di matching proiettando errori locali in deformazioni geometriche globali (RMS X = 7.89 px per ORB Homography).

7.6.4 Considerazioni sulla Sovra-Parametrizzazione

I risultati confermano un principio generale: la complessità del modello deve essere commisurata alla natura del moto. Per una sequenza con moto forward costante e oscillazioni verticali periodiche, un modello a 4 parametri (traslazione + rotazione + scala isotropa) cattura il moto rilevante senza introdurre gradi di libertà instabili.

7.7 Sintesi e Verdetto

Escludendo Block Matching — le cui metriche numeriche eccellenti sono un artefatto della stima incompleta del moto, non una misura di qualità percettiva — **l'algoritmo migliore in assoluto per questa sequenza è ORB Partial**, con il miglior RMS X tra i metodi feature-based (5.33 px) e la JR più alta su entrambi gli assi (84.5% / 91.9%) a fronte di un tempo di elaborazione accettabile (45.1 s).

Se il vincolo è il tempo, **Optical Partial** rappresenta la scelta ottimale: produce una stabilità percettiva pressoché equivalente (JR 81.0% / 91.5%) in soli 30.1 s, con un risparmio del 33% rispetto a ORB Partial.

- **Algoritmo migliore (qualità): ORB Partial** — massima stabilità percettiva, stima del moto globale precisa, JR Y = 91.9%.
- **Algoritmo migliore (qualità/velocità): Optical Partial** — risultato visivo equivalente a ORB Partial, 30.1 s di elaborazione.
- **Da evitare:** varianti Homography (sovra-parametrizzazione, JR X \leq 67%) e Block Matching (stima del moto incompleta, risultato visivo inferiore nonostante JR numerica >97%).

La lezione principale è che le metriche basate sulla stima del moto non sono sufficienti a valutare la qualità percettiva: occorre tenere conto della completezza della stima (copertura del moto reale) e non solo della sua coerenza interna.

Capitolo 8

Conclusioni

8.1 Riepilogo del Lavoro

Il progetto ha prodotto un sistema completo di stabilizzazione video digitale con architettura modulare a cinque livelli. Sono stati implementati e integrati:

- **Block Matching** con metrica SAD/MAD, eliminazione outlier via z-score e aggregazione configurabile (mediana, media, media pesata per confidenza);
- **Optical Flow** con rilevamento Shi-Tomasi, tracciamento Lucas-Kanade piramidale e stima della trasformazione (Partial/Affine/Homography) tramite RANSAC;
- **ORB Feature Matching** con Brute-Force Matcher, Hamming distance e Lowe's ratio test, abbinato agli stessi tre modelli RANSAC;
- tre filtri di smoothing della traiettoria (Moving Average, Gaussiano, Esponenziale IIR) con clamping automatico della finestra;
- compensazione geometrica con `warpAffine`, gestione bordi configurabile e cropping adattivo.

Il sistema espone le proprie funzionalità tramite un'interfaccia Streamlit con tre tab: stabilizzazione singola, confronto multi-metodo con barre di progresso dedicate, e analisi quantitativa delle metriche con grafici comparativi.

8.2 Risultati Principali

I test sperimentali su una sequenza di camminata a 1080p/30fps hanno prodotto i seguenti risultati chiave:

- **ORB Partial** ottiene la stabilizzazione più efficace: JR X=87.7%, JR Y=92.8%, JR θ =96.1%, RMS X=5.16 px, con un tempo di elaborazione di 24.9 s.
- **Optical Flow Partial** offre un compromesso favorevole: elaborazione in 18.1 s con JR \approx 57% su entrambi gli assi (configurazione con finestra 15 frame).
- Il modello **Homography** si rivela sistematicamente peggiore per questo tipo di moto, con RMS e JR inferiori rispetto a Partial e Affine in tutte le varianti testate.

- Il modello **Partial** (similitudine rigida, 4 parametri) è il più adatto a sequenze con moto prevalentemente traslazionale e oscillatorio, risultando più stabile numericamente rispetto ad Affine e Homography.

8.3 Limitazioni del Sistema

Il sistema presenta alcune limitazioni che ne circoscrivono l'applicabilità:

- **Elaborazione offline:** la pipeline a due passate richiede l'intero video in ingresso; non è applicabile in tempo reale senza modifiche architetturali significative.
- **Perdita di campo visivo:** il cropping introduce una riduzione del campo visivo proporzionale all'ampiezza delle oscillazioni; con moto molto ampio, il `crop_ratio` deve essere ridotto ulteriormente.
- **Scene dinamiche:** oggetti in movimento indipendente nella scena (pedoni, veicoli) possono contaminare la stima del GMV; la rimozione outlier via z-score mitiga ma non elimina questo problema.
- **Block Matching:** il metodo è in stato di sviluppo per i test di confronto; la stima di rotazione è fissa a zero, limitando l'efficacia su scene con componenti rotazionali significative.
- **Smoothing window:** la scelta della finestra è critica e dipende dalla velocità del moto intenzionale; finestre troppo ampie possono cancellare movimenti legittimi come pan e tilt.

8.4 Considerazioni Finali

Il progetto dimostra come un approccio modulare, con chiara separazione tra stima del moto, filtraggio e compensazione, consenta di integrare e confrontare algoritmamente strategie diverse senza modificare l'infrastruttura. La disponibilità di metriche quantitative oggettive (RMS e JR) ha permesso di evidenziare che la complessità del modello non corrisponde necessariamente a prestazioni migliori: per il tipo di moto analizzato, il modello più semplice (Partial) si è rivelato il più efficace.

Il lavoro integra in modo coerente aspetti di visione artificiale, elaborazione numerica di segnali, geometria proiettiva e ingegneria del software, fornendo una base estensibile per applicazioni pratiche di stabilizzazione video.

Appendice A

Struttura Completa del Progetto

```
src/
    motion_estimation.py      # MotionEstimator (Block Matching SAD/MAD)
    global_motion.py          # GlobalMotionEstimator (aggregazione, z-score)
    trajectory_smoothing.py  # TrajectoryFilter (MA, Gaussian, Exponential)
    motion_compensation.py   # MotionCompensator (warpAffine, crop)
    video_stabilizer.py      # VideoStabilizer (pipeline a due passate)
ui/
    tab_1v1.py                # Tab stabilizzazione singola
    tab_process.py            # Tab confronto multi-metodo
    tab_metrics.py            # Tab analisi metriche
    stabilization.py          # Application Logic Layer
    plot_utils.py              # Grafici Matplotlib
    video_utils.py            # Video comparativi + FFmpeg H.264
    config_loader.py          # Caricamento YAML
    styles.py                 # CSS Streamlit
config/
    config_block_matching.yaml
    config_optical_flow.yaml
    config_orb.yaml
app.py                      # Entry point Streamlit
main.py                     # Entry point CLI
requirements.txt
```

Appendice B

File di Configurazione YAML

B.1 config_block_matching.yaml

Listing B.1: Configurazione Block Matching

```
1 motion_estimation:
2     block_size: 32                      # Dimensione blocco (px)
3     search_range: 12                     # Raggio area di ricerca (px)
4     metric: "sad"                       # 'sad' o 'mad'
5
6 global_motion:
7     motion_model: "translation"
8     outlier_threshold: 2.0
9     aggregation_method: "median"
10    estimate_rotation: true
11    estimation_method: "block_matching"
12
13 trajectory_smoothing:
14     smoothing_window: 45
15     filter_type: "gaussian"
16     gaussian_sigma: 2.0
17     exponential_alpha: 0.9
18     deadband_px: 0.0
19
20 motion_compensation:
21     crop_ratio: 0.85
22     border_mode: "replicate"
```

B.2 config_optical_flow.yaml

Listing B.2: Configurazione Optical Flow – Shi-Tomasi + Lucas-Kanade

```
1 global_motion:
2   motion_model: "translation"
3   estimation_method: "optical_flow"
4   optical_flow:
5     feature_type: "shi_tomasi"      # Shi-Tomasi corner
6       detection + LK tracking
7     transform_type: "partial"      # 'partial', 'affine', '
8       homography'
9
10    # Shi-Tomasi + Lucas-Kanade
11    max_corners: 1500
12    quality_level: 0.002
13    min_distance: 10
14    block_size: 7
15    win_size: 21
16    max_level: 3
17
18    # RANSAC
19    ransac_reproj_threshold: 6.0
20
21 trajectory_smoothing:
22   smoothing_window: 45
23   filter_type: "gaussian"
24   gaussian_sigma: 2.0
25   deadband_px: 0.0
26
27 motion_compensation:
28   crop_ratio: 0.85
29   border_mode: "replicate"
```

B.3 config_orb.yaml

Listing B.3: Configurazione ORB – Oriented FAST and Rotated BRIEF

```
1 global_motion:
2     motion_model: "translation"
3     estimation_method: "optical_flow" # pipeline condivisa,
4         feature_type='orb'
5     optical_flow:
6         feature_type: "orb"           # ORB descriptor +
7             BFMatcher Hamming
8         transform_type: "partial"    # 'partial', 'affine', '
9             homography,
10
11     # ORB
12     orb_n_features: 1000
13     orb_scale_factor: 1.2
14     orb_n_levels: 10
15     orb_edge_threshold: 31
16     orb_ratio_threshold: 0.75    # Lowe's ratio test
17
18     # RANSAC
19     ransac_reproj_threshold: 6.0
20
21 trajectory_smoothing:
22     smoothing_window: 45
23     filter_type: "gaussian"
24     gaussian_sigma: 2.0
25     deadband_px: 0.0
26
27 motion_compensation:
28     crop_ratio: 0.85
29     border_mode: "replicate"
```

Bibliografia

- [1] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.
- [2] Richard Hartley and Andrew Zisserman. Multiple view geometry in computer vision. 2003.
- [3] David G Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.
- [4] Bruce D Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, 2:674–679, 1981.
- [5] OpenCV team. *OpenCV: Open Source Computer Vision Library*, 2024. <https://opencv.org>.
- [6] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. pages 2564–2571, 2011.
- [7] Jianbo Shi and Carlo Tomasi. Good features to track. pages 593–600, 1994.
- [8] Streamlit Inc. *Streamlit – The fastest way to build data apps*, 2024. <https://streamlit.io>.