

POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria



**TAMPERING DETECTION PER
CAMERE INTELLIGENTI A BASSO
CONSUMO ENERGETICO**

Relatore: Prof. Giacomo BORACCHI
Correlatori: Ing. Claudio MARCHISIO
Ing. Alessandro SENTINELLI

Tesi di Laurea di:
Adriano GAIBOTTI, matricola 780200

Anno Accademico 2013-2014

Sommario

Nel campo dei sistemi di monitoraggio video, uno dei principali problemi è quello di identificare eventi che possano compromettere la corretta ripresa della scena. Può capitare, ad esempio, che dell'acqua piovana si depositi sulla lente della camera, rendendo l'immagine acquisita sfocata, oppure che la camera si sposti, a causa di un intenzionale intervento umano o per eventi naturali quali una raffica di vento, e non riprenda più la scena da sorvegliare.

Il problema di individuare, in maniera automatica, questo tipo di eventi prende il nome di *tampering detection*. Nella letteratura scientifica questo problema è stato affrontato solamente per applicazioni di *videosorveglianza*, dove la camera opera con acquisizione continua, dispone di una certa potenza computazionale e viene alimentata a corrente. Lo scopo della tesi è lo sviluppo di un algoritmo di tampering detection per sistemi *embedded* e a basso consumo da utilizzarsi in scenari di monitoraggio. In particolare, l'algoritmo è caratterizzato da un basso carico computazionale ed è pensato per scenari, tipo il monitoraggio ambientale, dove il sistema, per ridurre il consumo energetico, acquisisce e analizza poche immagini al minuto o all'ora (*framerate bassi*). In questi casi scene ad *alta dinamicità*, come una strada in cui passano macchine e pedoni, non permettono di identificare eventi di tampering tramite un confronto tra frame consecutivi. Inoltre, operando a bassi framerate, si verificano sostanziali variazioni di luminosità tra immagini consecutive.

L'algoritmo proposto si basa su indicatori, estratti dalle singole immagini, calcolati a bassa complessità computazionale; tali indicatori vengono monitorati nel tempo attraverso tecniche *sequenziali* e di *outlier detection* per identificare l'istante in cui avviene l'evento di tampering. Data l'alta variabilità degli indicatori utilizzati, abbiamo introdotto una fase di *segmentazione* della scena, in modo da limitare l'analisi ad alcune regioni specifiche: questo permette di migliorare le prestazioni dell'algoritmo e diminuire il numero di falsi allarmi.

La tesi è stata svolta durante un lavoro di stage presso il gruppo *Advanced System Technology* di *STMicroelectronics*, dove la ricerca è volta allo sviluppo di algoritmi intelligenti di elaborazione immagini da integrare nei propri dispositivi embedded. Sono stati messi a punto diversi sistemi di acquisizio-

ne operanti a diversi framerate, che hanno permesso di generare i dataset per testare l'efficacia della soluzione proposta e, in particolare, dei vantaggi nell'utilizzo della segmentazione a supporto del tampering detection.

Indice

Sommario	i
1 Introduzione	3
2 Stato dell'arte	5
2.1 Modello della camera	5
2.1.1 La matrice prospettica della camera	6
2.1.2 Parametri intrinseci	8
2.1.3 Parametri estrinseci	10
2.2 Rappresentazione digitale delle immagini	11
2.3 Identificazione del cambiamento nella stazionarietà di un processo	12
2.3.1 Metodi di Change-Point	13
2.3.2 Change-Detection Test	15
2.4 Tampering Detection	22
2.4.1 Concetti e terminologia	24
2.4.2 Tecniche basate su confronto di background	25
2.4.3 Tecniche basate su monitoraggio sequenziale	29
3 Impostazione del problema di ricerca	33
3.1 Modello delle osservazioni	33
3.1.1 Sfocatura	33
3.1.2 Spostamento della camera	35
3.1.3 Occlusione e guasti della camera	37
3.2 Tampering detection	37
4 Soluzione proposta	39
4.1 Indicatori utilizzati per identificare gli eventi di tampering . .	39
4.1.1 Misura della sfocatura nell'immagine	39
4.1.2 Misura dello spostamento della camera	41
4.1.3 Comportamento degli indicatori nel tempo	41
4.1.4 Detrending degli indicatori	45
4.2 Algoritmo di identificazione di sfocature e di spostamenti della camera	52

4.2.1	Identificazione delle sfocature	53
4.2.2	Identificazione degli spostamenti della camera	54
4.3	Algoritmo di segmentazione	56
4.3.1	Calcolo dei descrittori utilizzati per la segmentazione .	57
4.3.2	Partizionamento dei feature vector per l'estrazione delle regioni	59
4.3.3	Separazione delle regioni	62
5	Prove sperimentali e valutazione	65
5.1	Metodi considerati e obiettivi	65
5.2	Acquisizione dei dataset	65
5.2.1	Eventi di tampering	67
5.2.2	Definizione dei ground truth	67
5.3	Figure di merito	68
5.4	Esperimento 1: tampering sintetico	68
5.4.1	Creazione dei tampering	68
5.4.2	Risultati	68
5.5	Esperimento 2: tampering reale	68
6	Direzioni future di ricerca e conclusioni	69
	Bibliografia	71

Elenco delle figure

2.1	Athanasius Kircher, Principio della prospettiva nella camera obscura, 1671	6
2.2	Schematizzazione del sistema ottico della camera	7
2.3	Similitudini tra triangoli usate nel calcolo dei parametri intrinseci	8
2.4	Coordinate di un frame	9
2.5	Trasformazione dal sistema di coordinate dell'ambiente a quello della camera	10
2.6	Esempio di estrazione dei canali RGB	12
2.7	Esempio di estrazione dei canali YUV	13
2.8	Esempio di CPM	15
2.9	Procedura del CUSUM adattativo	18
2.10	Estrazione dei descrittori nel metodo CI-CUSUM	18
2.11	Esempio di come opera la regola dell'intersezione degli intervalli di confidenza (ICI-rule)	21
2.12	Schema del modo di operare del CDT di tipo gerarchico	22
2.13	Tecniche di tampering detection	23
2.14	Sistema di monitoraggio video	24
2.15	Comportamento della trasformata di Fourier nel caso di sfocatura	27
2.16	Sequenza di frame acquisiti a 30 fps	28
2.17	Sequenza di frame acquisiti ogni 30 secondi	29
3.1	Esempi di sfocature	34
3.2	Sequenza di otto frame consecutivi acquisiti ogni minuto	35
3.3	Esempio di spostamento della camera	36
3.4	Esempi di occlusione	37
3.5	Esempi di guasti	38
4.1	Esempio di cambi di luminosità tra il giorno e la notte	41
4.2	Esempio di presenza di sfocature dovute all'aumento del tempo di esposizione della camera	42
4.3	Energia media del gradiente lungo un'acquisizione di 24 ore (a) e suo detrending (b)	43

4.4	Energia media della luma lungo un'acquisizione di 24 ore (a) e suo detrending (b)	44
4.5	Energia media del gradiente (a) e suo detrending (b) nel caso di una sfocatura	46
4.6	Energia media della luma (a) e suo detrending (b) nel caso di una sfocatura	47
4.7	Energia media del gradiente (a) e suo detrending (b) nel caso di uno spostamento della camera	48
4.8	Energia media della luma (a) e suo detrending (b) nel caso di uno spostamento della camera	49
4.9	Esempio di sequenza dell'energia media della luma (a) e del suo detrending (b) con un displacement difficile da identificare	51
4.10	Esempio di spostamento della camera	52
4.11	Esempio di segmentazione della scena	52
4.12	Passaggio dallo spazio dei frame allo spazio dei feature vector	57
4.13	Descrittori per la segmentazione della scena	59
4.14	Segmentazione della scena	62
5.1	Il Raspberry Pi (a) e il sistema di acquisizione basato su di esso (b)	67

Capitolo 1

Introduzione

Una branca dell’industria tecnologica che sta prendendo sempre più piede è quella dei *sistemi di monitoraggio video*. L’abbattimento dei costi delle camere e una sempre maggiore integrazione di tecnologie hardware con sensori e infrastrutture di rete hanno permesso una rapida crescita di questo settore.

Inoltre, il grande successo di Internet degli ultimi anni ha permesso l’utilizzo dei sistemi di monitoraggio video per applicazioni che prima non erano neanche pensabili. Oggigiorno è possibile avere un sistema di videosorveglianza domestico spendendo poche centinaia di euro, controllabile con il proprio *smartphone* attraverso internet, oppure è possibile vedere le condizioni meteorologiche o del traffico nella propria città attraverso le immagini riprese da una *webcam*. Un servizio di questo genere è fornito da molte piattaforme web di previsioni del tempo. Ad esempio, il sito *ilMeteo.it* [1] permette di vedere in tempo reale il contenuto video di webcam presenti nelle principali città italiane o in località di interesse turistico.

Uno dei maggiori problemi quando si ha a che fare con sistemi di monitoraggio video consiste nell’identificazione di particolari eventi in grado di compromettere la corretta ripresa della scena da parte del sensore.

Questo tipo di eventi viene classificato generalmente sotto il nome di *tampering*, dall’inglese *manomissione*. Il termine deriva dal fatto che, di solito, si ha a che fare con azioni intenzionali atte a impedire la ripresa della scena, da parte della camera, in particolari momenti. Può succedere, ad esempio, che un ladro copra l’obiettivo della camera con qualche oggetto, in modo da poter agire indisturbato, oppure che la sposti in modo che riprenda qualcos’altro. In questa tesi definiamo con il termine tampering un qualsiasi evento, intenzionale o meno, che possa compromettere la corretta acquisizione della scena inquadrata dalla camera. Ad esempio una sfocatura può essere causata da dell’acqua piovana che si deposita a ridosso della lente, oppure può succedere che una folata di forte vento sposti la camera, cambiando quindi l’inquadratura della scena.

Questa definizione permette di analizzare il problema anche per quelle applicazioni di monitoraggio video che non sono direttamente legate alla videosorveglianza, ad esempio per lo scenario delle webcam introdotto sopra, in cui l'evento non è necessariamente dovuto a un intervento umano intenzionale.

Le tecniche algoritmiche che vengono utilizzate per identificare gli eventi di tampering prendono il nome di algoritmi di *tampering detection*.

La letteratura scientifica ha prodotto molte soluzioni a riguardo, ma la maggior parte sono legate ad applicazioni di videosorveglianza. Questo particolare tipo di applicazioni ha bisogno di camere che acquisiscano a framerate *continuo*, solitamente tra i 30 frame per secondo (fps) e i 2 fps. È infatti impensabile che una camera di videosorveglianza possa acquisire, ad esempio, un frame ogni minuto.

Questo, però, può andare bene nel caso si considerino applicazioni come quella delle webcam dove, al contrario, un framerate elevato è inutile e provoca una dissipazione di energia non giustificata.

Lo scopo della tesi è lo sviluppo di un algoritmo di tampering detection per camere *embedded* e a basso consumo. In particolare abbiamo considerato lo scenario in cui la camera può essere adoperata a basso framerate

La tesi è strutturata nel seguente modo.

Nel Capitolo 2 si mostra lo stato dell'arte.

Nel Capitolo 3 si illustra come è stato formalizzato il problema.

Nel Capitolo 4 si illustra la soluzione proposta per risolvere il problema.

Nel Capitolo 5 si mostrano le prove realizzate per validare la soluzione proposta, descrivendo anche la realizzazione dei dataset e i risultati ottenuti.

Nel Capitolo 6 infine si mostrano le prospettive future di ricerca e si tirano le conclusioni.

Capitolo 2

Stato dell'arte

In questo capitolo presentiamo i concetti fondamentali della teoria alla base della *visione artificiale* e dell'elaborazione di segnali e lo studio fatto finora sul problema del *tampering detection*.

Nei Paragrafi 2.1 e 2.2 diamo una visione generale del modello della camera e di come vengono rappresentate le immagini digitali, in modo da comprendere meglio gli argomenti trattati nei capitoli successivi.

Nel Paragrafo 2.3 illustriamo le tecniche maggiormente utilizzate per identificare cambiamenti nella stazionarietà di un segnale o una sequenza di dati. Nel Paragrafo 2.4 descriviamo le soluzioni al problema del tampering detection presenti nella letteratura scientifica, definendo inoltre i concetti e la terminologia che verrà utilizzata nel resto della trattazione.

2.1 Modello della camera

Affinché un sistema di visione artificiale possa risolvere il problema per cui è stato progettato, è necessario che esso sia in grado di acquisire una porzione della realtà che lo circonda. Questo compito viene svolto da un particolare sensore chiamato *camera*, il cui scopo principale è quello di creare una *proiezione* dell'ambiente *tridimensionale* in un sistema *bidimensionale*. Il principio alla base della camera è un concetto inventato da *Filippo Brunelleschi* nel XV secolo, chiamato *prospettiva a punto unico di fuga* (*pinhole perspective*) [2]. Il modello matematico utilizzato considera uno spazio tridimensionale, detto *ambiente*, e un piano bidimensionale chiamato *immagine*. I punti su tale piano sono la *proiezione* dell'ambiente tridimensionale. La proiezione è dovuta a un punto, chiamato *centro ottico*, in cui viene convogliata la luce emanata dall'ambiente tridimensionale. In questa astrazione, ciascun punto dello spazio tridimensionale viene associato univocamente a un punto nel piano immagine attraverso il raggio di luce che passa dal centro ottico. All'epoca di Brunelleschi, il modello del centro ottico veniva realizz-

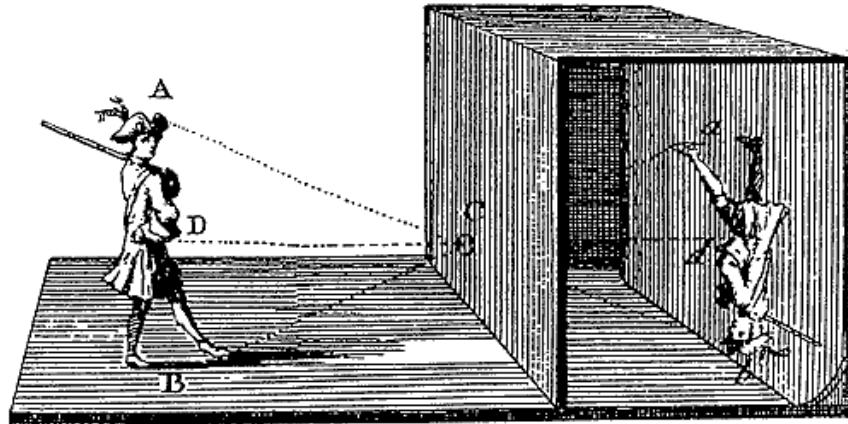


Figura 2.1: Athanasius Kircher, *Principio della prospettiva nella camera obscura*, 1671

zato da una *camera oscura*, come si può osservare nell’illustrazione in Figura 2.1, mentre nei moderni sistemi di acquisizione viene realizzato tramite *lenti ottiche*.

2.1.1 La matrice prospettica della camera

Nella Figura 2.2 sono illustrati i concetti che sono utilizzati per descrivere il sistema ottico della camera. Indichiamo con la lettera maiuscola $\mathbf{M} = [X, Y, Z]^T$ un qualsiasi punto nello spazio tridimensionale e con la lettera minuscola $\mathbf{m} = [u, v]^T$ la sua proiezione sul piano immagine dovuta al centro ottico \mathbf{C} . La linea che congiunge il centro ottico \mathbf{C} perpendicolarmente al piano immagine è detta *asse principale*, indicata con Z , e il suo punto di intersezione con il piano stesso viene definita *punto principale* \mathbf{c} . La distanza tra il punto principale e il centro ottico viene definita *distanza focale*, e viene indicata con f .

La rappresentazione dei punti viene fatto attraverso le loro *coordinate omogenee*. In questo modo è possibile rappresentare le trasformazioni tra sistemi di coordinate di ordine differente tramite una singola trasformazione matriciale. Chiamiamo coordinate omogenee di un punto $\mathbf{m} = [u, v]$ del piano una qualsiasi terna ordinata $[U, V, W]$ di numeri reali tali che

$$W \neq 0, \frac{U}{W} = u, \frac{V}{W} = v.$$

Analogamente, le coordinate omogenee di un punto $\mathbf{M} = [x, y, z]$ nello spazio tridimensionale saranno costituite da una quaterna di numeri $[X, Y, Z, W]$ tali che

$$W \neq 0, \frac{X}{W} = x, \frac{Y}{W} = y, \frac{Z}{W} = z.$$

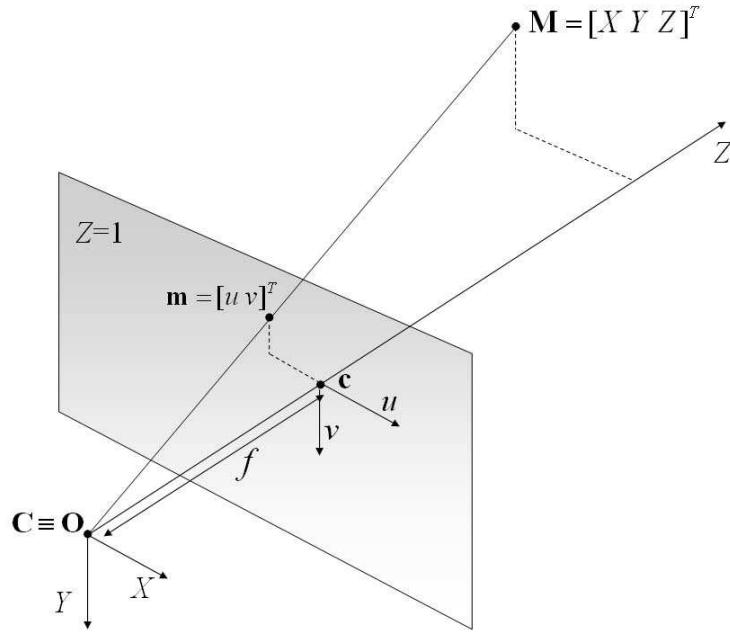


Figura 2.2: Schematizzazione del sistema ottico della camera

La coordinata W viene definita *valore di scala*; nel caso $W = 1$ le rimanenti coordinate omogenee rappresentano le *coordinate cartesiane* del punto.

Consideriamo un punto \mathbf{M} nello spazio tridimensionale, rappresentato tramite le coordinate omogenee $[X, Y, Z, 1]^T$, e il suo punto immagine \mathbf{m} rappresentato dalle coordinate omogenee $[u, v, 1]^T$. La proiezione della camera può essere espressa come

$$\mathbf{m} = P\mathbf{M}, \quad (2.1)$$

dove P è una matrice di dimensioni 3×4 chiamata *matrice prospettica* [3]. Tale matrice contiene al suo interno informazioni sui parametri della camera a cui è associata, e descrive sia la proiezione sul piano immagine che le trasformazioni di camera rispetto al sistema di riferimento tridimensionale. Formalmente la matrice prospettica viene definita attraverso la concatenazione di due matrici: una che rappresenta i *parametri intrinseci* e un'altra che rappresenta i *parametri estrinseci* [4].

- I *parametri intrinseci* rappresentano le caratteristiche interne della camera come la distanza focale, il centro focale e le caratteristiche di *distorsione* della lente.
- I *parametri estrinseci* rappresentano la posizione della camera rispetto al sistema di riferimento dello spazio tridimensionale.

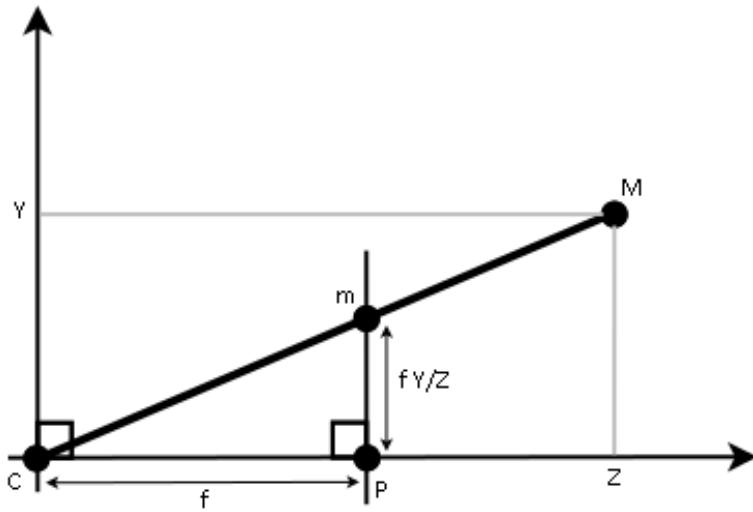


Figura 2.3: Similitudini tra triangoli usate nel calcolo dei parametri intrinseci

2.1.2 Parametri intrinseci

La relazione fra le coordinate tridimensionali di un punto $\mathbf{M} = [X, Y, Z]$ e le coordinate della sua proiezione $\mathbf{m} = [u, v]$ è:

$$u = f \cdot \frac{X}{Z},$$

$$v = f \cdot \frac{Y}{Z}.$$

Tale relazione deriva dalle proprietà dei triangoli simili, come mostra la Figura 2.3. La *mappatura* viene definita, dunque, dalla seguente relazione:

$$[X, Y, Z]^T \mapsto \left[f \cdot \frac{X}{Z}, f \cdot \frac{Y}{Z} \right]^T. \quad (2.2)$$

Se il punto nello spazio e la sua proiezione sono rappresentati attraverso le loro coordinate omogenee, la (2.2) può essere riscritta come

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} f \cdot X \\ f \cdot Y \\ Z \end{bmatrix} = K \begin{bmatrix} I & | & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \quad (2.3)$$

dove la matrice

$$K = \begin{bmatrix} f & & \\ & f & \\ & & 1 \end{bmatrix} \quad (2.4)$$

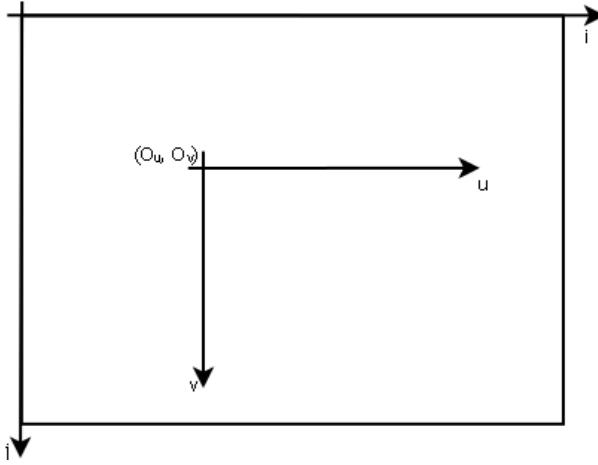


Figura 2.4: Coordinate di un frame

viene detta *matrice di calibrazione* della camera.

Nella formula (2.2) abbiamo assunto che l'origine delle coordinate del piano immagine si trovi nel punto principale, come è illustrato nella Figura 2.2. Se consideriamo il punto principale di coordinate $[p_u, p_v]$, la (2.2) diventa

$$[X, Y, Z]^T \mapsto \left[f \cdot \frac{X}{Z} + p_u, f \cdot \frac{Y}{Z} + p_v \right]^T, \quad (2.5)$$

mentre la matrice di calibrazione(2.4) diventa

$$K = \begin{bmatrix} f & p_u \\ f & p_v \\ 1 & \end{bmatrix}. \quad (2.6)$$

Nel caso in cui il sensore della camera sia *digitale*, l'immagine deve essere quantizzata come una matrice di W colonne e H righe, in cui ciascun elemento prende il nome di *pixel* (dalla contrazione di *picture element*). Ogni pixel possiede, quindi, delle coordinate che definiscono la sua posizione sulla matrice del frame. Se definiamo con (i, j) le coordinate del generico pixel sulla matrice del frame, dove l'origine è posta nell'angolo in alto a sinistra, e con (O_u, O_v) le coordinate del punto principale secondo il sistema di riferimento del frame (si veda a riguardo la Figura 2.4), possiamo mettere in relazione le coordinate dell'immagine con le coordinate frame nel seguente modo:

$$u = (i - O_u) \cdot S_u$$

e

$$v = (j - O_v) \cdot S_v,$$

dove S_u e S_v sono le dimensioni orizzontali e verticali del singolo pixel. È necessario introdurre, nella matrice di calibrazione, l'informazione del

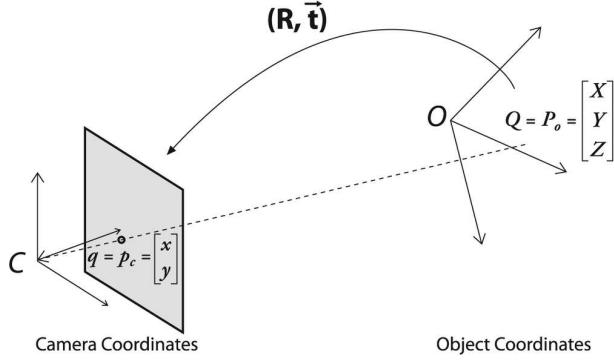


Figura 2.5: Trasformazione dal sistema di coordinate dell'ambiente a quello della camera

numero di pixel per unità di distanza in coordinate immagine m_u e m_v , rispettivamente nelle direzioni u e v , modificando la matrice di calibrazione definita in (2.6) come

$$K = \begin{bmatrix} \alpha_u & u_0 \\ \alpha_v & v_0 \\ 1 & \end{bmatrix}, \quad (2.7)$$

dove $\alpha_u = fm_u$ e $\alpha_v = fm_v$, mentre il punto principale $[p_u, p_v]$ viene riscritto come $[u_0, v_0] = [m_u p_u, m_v p_v]$.

Nel caso in cui la matrice di calibrazione sia nota, la camera viene detta *completamente calibrata*.

La matrice di calibrazione K rappresenta, quindi, il cambio di sistema di riferimento nel piano immagine, ridefinendo il centro della camera (u_0, v_0) in modo che coincida con il punto principale.

2.1.3 Parametri estrinseci

La matrice K descritta dall'equazione (2.7) è valida per camere poste all'origine del sistema di riferimento. Se consideriamo delle camere poste in un punto arbitrario dell'ambiente, quindi, è necessario che le coordinate dei punti, definite nel sistema di riferimento tridimensionale, siano ridefinite secondo il sistema di coordinate della camera. Questa trasformazione viene definita attraverso una *rototraslazione*, nello spazio, del sistema di riferimento dell'ambiente, in modo che coincida con quello della camera. Se $\tilde{\mathbf{M}}$ rappresenta le coordinate cartesiane di un punto nel sistema di riferimento dell'ambiente e $\tilde{\mathbf{M}}_{cam}$ rappresenta lo stesso punto nel sistema di riferimento della camera, si può scrivere

$$\tilde{\mathbf{M}}_{cam} = R(\tilde{\mathbf{M}} - C), \quad (2.8)$$

dove C rappresenta le coordinate del centro della camera nel sistema di riferimento dell'ambiente, R è una *matrice di rotazione* 3×3 che rappresenta

l'orientamento della camera rispetto al sistema di riferimento dell'ambiente e K è la matrice di calibrazione della camera. L'equazione (2.8) può essere riscritta in forma matriciale: se definiamo \mathbf{M} e $\tilde{\mathbf{M}}_{cam}$ come le rappresentazioni in coordinate omogenee di \mathbf{M} e $\tilde{\mathbf{M}}_{cam}$, abbiamo

$$\mathbf{M}_{cam} = \begin{bmatrix} R & -RC \\ 0 & 1 \end{bmatrix} \mathbf{M},$$

da cui, riprendendo l'equazione 2.3,

$$\begin{aligned} \mathbf{m} &= K [I \mid 0] \mathbf{M}_{cam} \\ &= K [I \mid 0] \begin{bmatrix} R & -RC \\ 0 & 1 \end{bmatrix} \mathbf{M} \\ &= K [R \mid -RC] \mathbf{M}. \end{aligned}$$

Possiamo, quindi, scrivere l'equazione della matrice di proiezione della camera come

$$P = K [R \mid t],$$

dove $t = -RC$ prende il nome di *vettore di traslazione*.

2.2 Rappresentazione digitale delle immagini

Nel Paragrafo 2.1.2 abbiamo introdotto il concetto di *immagine digitale* come rappresentazione numerica dell'immagine bidimensionale. Esistono vari modi di rappresentare un'immagine in formato digitale. Le rappresentazioni che a noi interessa considerare sono [4]:

- *scala di grigi*, in cui l'immagine è rappresentata come una matrice \mathbf{I} di dimensioni $H \times W$, dove l'elemento $\mathbf{I}(i, j)$ rappresenta il valore di *intensità luminosa* (*luma*) del pixel di coordinate (i, j) ;
- *RGB*, dove l'immagine è rappresentata come una matrice *tridimensionale* \mathbf{I}_{RGB} di dimensioni $H \times W \times 3$, in cui vengono memorizzati i livelli di intensità luminosa dei tre colori fondamentali *rosso* (R), *verde* (G) e *blu* (B) per ciascun pixel (Figura 2.6);
- *YUV*, in cui lo spazio colore viene definito tramite un componente di luma (Y) e due componenti di *crominanza* (UV), che rappresentano rispettivamente i valori di differenza dal colore blu ($U = B - Y$) e dal colore rosso ($V = R - Y$) (Figura 2.7).

Le immagini digitali possono essere memorizzate in diversi formati. Spesso vengono utilizzati algoritmi di *compressione*, che possono essere

- a perdita di informazione (*lossy*), come nel caso delle immagini *JPEG*;
- senza perdita di informazione (*lossless*), come nel caso dei file d'immagine *PNG* o *GIF*.

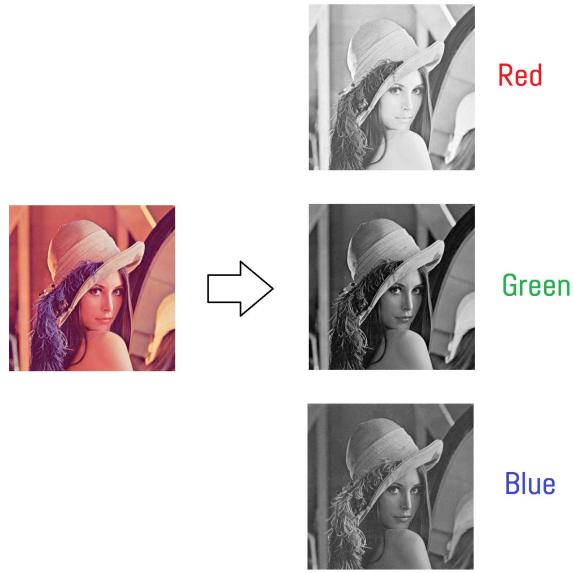


Figura 2.6: Esempio di estrazione dei canali RGB

2.3 Identificazione del cambiamento nella stazionarietà di un processo

Prima di illustrare come è stato affrontato il problema del tampering detection nella letteratura scientifica, affrontiamo il problema, più generale, di come identificare un cambiamento all'interno di un processo. Una caratteristica, richiesta in molte applicazioni di analisi e elaborazione intelligente di dati, riguarda l'assunzione che il processo generante questi dati non cambi nel tempo [5]. In questo caso si parla di processi *stazionari*, ovvero processi dove i dati possono essere considerati generati da un'unica distribuzione probabilistica.

In molte situazioni, soprattutto se vengono utilizzati dei sensori per misurare delle grandezze fisiche, si può incorrere in situazioni in cui questa ipotesi di stazionarietà venga meno. Ad esempio, può succedere che il sensore si guasti, oppure che fenomeni esterni non permettano un'acquisizione ottimale. In questi casi è opportuno avere a disposizione degli strumenti in grado di identificare quando la condizione di stazionarietà di un processo viene meno, in modo da poter lanciare un allarme a riguardo. Le tecniche principalmente utilizzate per questo scopo prendono il nome di *Change-Point Methods* (CPM) e *Change-Detection Tests* (CDT), e verranno illustrate nel resto del paragrafo.

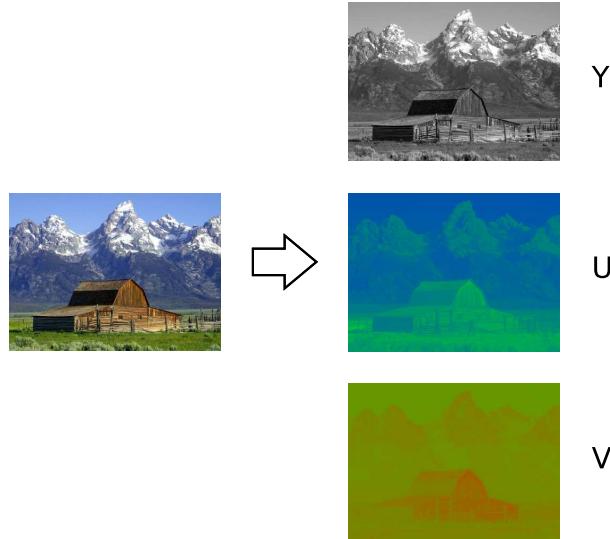


Figura 2.7: Esempio di estrazione dei canali YUV

2.3.1 Metodi di Change-Point

I metodi di Change-Point (CPM) vengono utilizzati per ispezionare una certa sequenza di dati e verificare la sua stazionarietà. Ciò viene fatto verificando che, all'interno della sequenza, esista un istante temporale in cui la distribuzione dei dati cambia (*change point*).

In maniera formale possiamo dire che, data una sequenza di dati

$$\mathcal{X} = \{x(t), t = 1, \dots, n\},$$

essa contiene un change point all'istante $\tau < n$ se è possibile suddividerla in due sotto-sequenze

$$\begin{aligned}\mathcal{A}_\tau &= \{x(t), t = 1, \dots, \tau\} \\ \mathcal{B}_\tau &= \{x(t), t = \tau + 1, \dots, n\}\end{aligned},$$

le quali sono due realizzazioni *indipendenti e identicamente distribuite* (i.i.d.) di due differenti variabili aleatorie con distribuzione \mathcal{F}_0 e \mathcal{F}_1 . È possibile, quindi, convertire il problema del change point in uno equivalente dove si valuta se \mathcal{A}_τ e \mathcal{B}_τ sono generati da due distribuzioni diverse.

Il problema, a questo punto, può essere riformulato come un *test d'ipotesi a due campioni* [6], dove l'*ipotesi nulla* (H_0) e l'*ipotesi alternativa* (H_1) sono le seguenti:

$$\begin{aligned}H_0 : \quad x(t) &\sim \mathcal{F}_0 \quad \forall t \\ H_1 : \quad x(t) &\sim \begin{cases} \mathcal{F}_0 & \text{se } t < \tau \\ \mathcal{F}_1 & \text{se } t \geq \tau \end{cases}.\end{aligned}$$

Per la valutazione delle ipotesi fatte sopra, occorrono delle opportune *statistiche di test a due campioni*

$$\mathcal{T}_\tau = \mathcal{T}(\mathcal{A}_\tau, \mathcal{B}_\tau),$$

in modo da poter comparare \mathcal{A}_τ e \mathcal{B}_τ . In questo modo è possibile rifiutare l'ipotesi nulla quando il valore della statistica \mathcal{T}_τ supera una certa soglia $h_{n,\alpha}$, calcolata in funzione di un certo *intervallo di confidenza* α e del numero di campioni n .

Se, ad esempio, consideriamo l'ipotesi che \mathcal{A}_τ e \mathcal{B}_τ siano generate da due distribuzioni *gaussiane*, è possibile valutare la dissimilarità tra le due distribuzioni controllando i due *valori attesi*. Possiamo usare, come statistica di test, la *differenza standardizzata tra le medie di due campioni*, definita come

$$D_\tau = \sqrt{\frac{\tau(n-\tau)}{n}} \frac{\bar{\mathcal{A}}_\tau - \bar{\mathcal{B}}_\tau}{S_\tau},$$

dove $\bar{\mathcal{A}}_\tau$ e $\bar{\mathcal{B}}_\tau$ denotano le *medie campionarie* valutate rispettivamente su \mathcal{A}_τ e \mathcal{B}_τ , mentre S_τ è la *varianza campionaria* valutata su tutto l'*insieme dei dati*.

Quando la statistica di test utilizzata, corrispondente a una specifica partizione dell'insieme dei dati, non fornisce l'evidenza necessaria a rifiutare H_0 , l'unica cosa che è possibile stabilire è che nell'istante τ scelto non si è avuto un cambiamento nella stazionarietà dell'insieme. *Ciò non implica che l'istante di cambiamento non sia un altro ancora da valutare*. In maniera rigorosa è necessario valutare, quindi, tutte le possibili partizioni dell'insieme dei dati considerati. Le ipotesi nulla e alternativa vengono, perciò, modificate nel seguente modo [7]:

$$\begin{aligned} H_0 : \quad & \forall t, \quad x(t) \sim \mathcal{F}_0 \\ H_1 : \quad & \exists \tau \quad x(t) \sim \begin{cases} \mathcal{F}_0 & \text{se } t < \tau \\ \mathcal{F}_1 & \text{se } t \geq \tau \end{cases}. \end{aligned}$$

Entrando più nel dettaglio, per ciascun punto $s \in \{2, \dots, n-1\}$, candidato a essere un change point, valutiamo la statistica di test \mathcal{T}_s . Viene scelto come change point quello che massimizza la statistica

$$M = \underset{s=2, \dots, n-1}{\operatorname{argmax}} (\mathcal{T}_s),$$

corrispondente al valore \mathcal{T}_M di \mathcal{T}

$$\mathcal{T}_M = \max_{s=2, \dots, n-1} (\mathcal{T}_s).$$

Una volta trovato \mathcal{T}_M il test va finalizzato comparando questa statistica con la soglia $h_{n,\alpha}$, in modo da verificare se l'ipotesi nulla sia da scartare

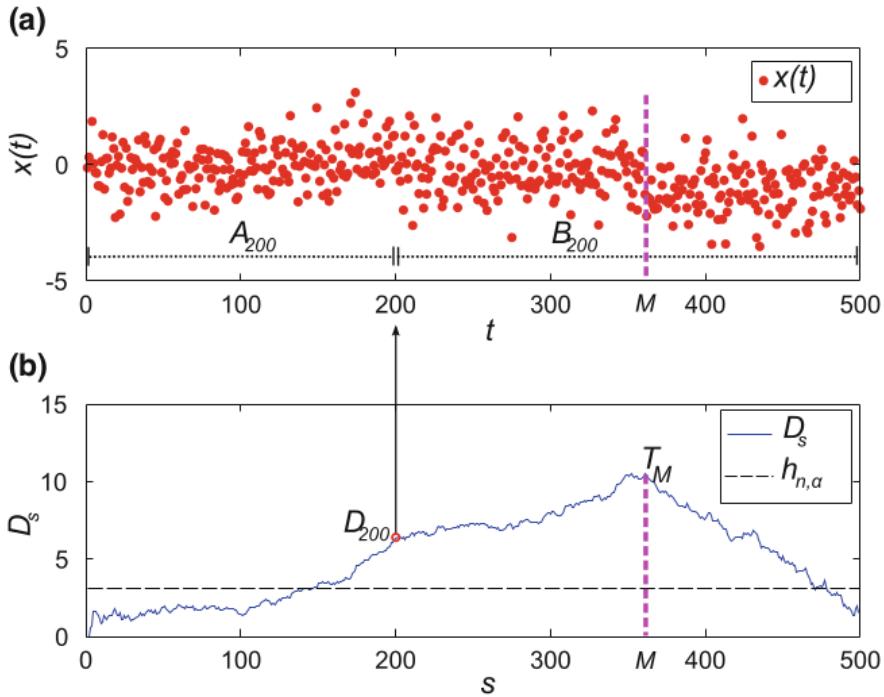


Figura 2.8: Esempio di CPM

o meno. Nella Figura 2.8 possiamo vedere un esempio di CPM basato su una statistica *t di Student* [5]. La sequenza è formata da 500 dati (a) ed è stato individuato un cambiamento all’istante $\tau = 350$. I valori delle varie statistiche in funzione di s sono visualizzati in (b).

Spesso è preferibile utilizzare dei test statistici *non parametrici*, soprattutto quando le distribuzioni dei dati sono incognite. Le statistiche maggiormente utilizzate sono quelle basate su *calcolo del rango* [7]. Quando si ha a che fare con cambiamenti nella *località* dei dati una statistica solitamente utilizzata è quella di *Mann-Whitney*, mentre quando si vogliono verificare cambiamenti *di scala* è possibile utilizzare la statistica di *Mood*. La statistica di *Lepage* invece può risultare utile per analizzare entrambi i tipi di cambiamento. Un altro approccio consiste nel comparare le *distribuzioni empiriche* dei due insiemi di dati, usando ad esempio la statistica di *Kolmogorov-Smirnov*.

2.3.2 Change-Detection Test

I metodi di change point sono stati principalmente pensati per essere eseguiti *a valle* della completa acquisizione dei dati. Inoltre il suo elevato costo computazionale fa sì che questa tecnica difficilmente possa essere utilizzabile all’interno di un sistema embedded. Le tecniche che vengono descritte in questo paragrafo, che prendono il nome di *change detection test* (CDT),

hanno come scopo principale quello di fare un processo dei dati *in linea*, ovvero non appena questi sono disponibili. Tecniche di questo tipo vengono anche chiamate *tecniche sequenziali*. La loro relativa semplicità dal punto di vista computazionale permette di utilizzarle all'interno di sistemi embedded ma, rispetto alle tecniche basate su CPM, abbiamo una latenza (ovvero l'intervallo di tempo tra l'istante di identificazione del cambiamento e quello in cui effettivamente questo è avvenuto) e un numero di falsi positivi (viene segnalato un cambiamento nella distribuzione quando in, realtà, questo non è avvenuto) maggiori.

CDT basati su CUSUM

Le *carte di controllo per le somme cumulate* (CUMulative SUM control chart, abbreviate in *CUSUM*) [5, 6] permettono di identificare un cambiamento all'interno di una sequenza di dati in maniera molto accurata utilizzando alcune informazioni *a priori* sul processo generante i dati. Queste informazioni vengono generalmente acquisite durante un *fase di configurazione* del test, in modo da fissare i parametri di test necessari all'analisi. Nel seguito presentiamo due metodi di CDT che estendono il metodo CUSUM tradizionale rilassando alcune delle sue assunzioni troppo restrittive. La prima variante permette di identificare in maniera automatica la configurazione dei parametri di test (*CUSUM adattativo*), mentre la seconda, chiamata *computational Intelligence CUSUM* (CI-CUSUM) è in grado di considerare un insieme più ricco di descrittori in modo da aumentare l'efficienza nell'identificare i cambiamenti [8].

CUSUM tradizionale e versione adattativa Sia

$$\mathcal{X} = \{x(t), t = 1, \dots, N\}, x(t) \in \mathbb{R}$$

una sequenza di N dati generata da un processo con densità probabilistica $f_\theta(x)$, che assumiamo sconosciuta e parametrizzata secondo un vettore $\theta \in \mathbb{R}^n$. Assumiamo, inoltre, che il processo cambi la sua stazionarietà a un istante T^0 sconosciuto. Questo avvenimento può essere modellato come un passaggio dal vettore dei parametri θ_0 a θ_1 , associati rispettivamente alle densità $f_{\theta_0}(x)$ e $f_{\theta_1}(x)$. La discrepanza tra le due densità viene valutata comparando il rapporto tra le log-verosimiglianze (*log-likelihood ratio*)

$$s(t) = \ln \frac{f_{\theta_1}(x(t))}{f_{\theta_0}(x(t))} \text{ per ogni } t = 1, \dots, N$$

e la *somma cumulata*

$$S(t) = \sum_{\tau=1}^t s(\tau)$$

CUSUM identifica un cambiamento in \mathcal{X} all'istante \hat{T} quando

$$g(t) = S(t) - m(t),$$

ovvero la differenza tra il valore della somma cumulata all'istante t e il suo valore minimo nel tempo

$$m(t) = \min_{\tau=1,\dots,t} S(\tau),$$

superà una certa soglia h .

Il metodo CUSUM tradizionale assume che i parametri θ_0 , θ_1 e la soglia h siano disponibili a priori. La versione adattativa viene incontro a questa assunzione che, generalmente, è difficile da soddisfare.

La variante prima di tutto genera la sequenza cumulata $Y = \{y(1), y(2), \dots\}$, dove $y(s)$ rappresenta il valore della media campionaria stimata su una finestra mobile non sovrapponibile di ampiezza n presa da \mathcal{X}

$$y(s) = \frac{1}{n} \sum_{t=(s-1)n+1}^{sn} x(t)$$

Scegliendo n abbastanza grande, per il *teorema del limite centrale* [6], la distribuzione di Y può essere approssimata con una distribuzione *gaussiana*. In questo modo, quindi, si può utilizzare il metodo CUSUM tradizionale direttamente sulla sequenza Y . I primi K dati di \mathcal{X} vengono usati come *insieme di configurazione* per i parametri necessari a identificare il cambiamento. Il vettore dei parametri θ_0 è caratterizzato da media e varianza di Y , stimate attraverso i primi K/n campioni di Y . Il vettore dei parametri θ_1 viene ottenuto attraverso l'identificazione di un intorno di confidenza per θ_0 , mentre la soglia h viene calcolata come il valore massimo di $g(t)$ nella sequenza Y

$$h = \max_{1 \leq t \leq N/n} g(t).$$

L'intera procedura è illustrata nella Figura 2.9.

CI-CUSUM Il metodo CI-CUSUM rappresenta un'estensione del CUSUM adattativo molto potente, in quanto ciascun descrittore, estratto dal flusso di dati, può essere utilizzato per avere differenti gradi di sensitività durante l'identificazione di un cambiamento. Facciamo riferimento alla Figura 2.10. I descrittori φ considerati contengono i principali momenti della distribuzione dei dati, come media e varianza, ma anche momenti superiori al secondo come skewness (*skew*) e kurtosis (*kurt*), oltre che altre informazioni derivate dalla densità probabilistica (*pdf*) e dalla funzione di ripartizione dei dati (*cdf*).

A ogni istante queste grandezze vengono confrontate con quelle appartenenti all'intervallo di configurazione (*training set*, indicato in Figura 2.10

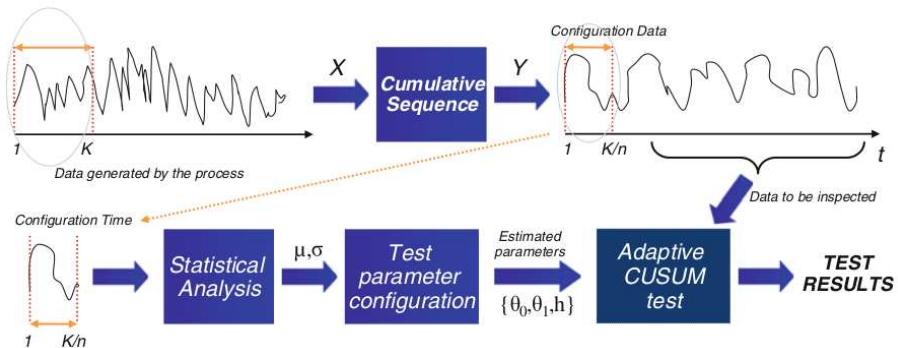


Figura 2.9: Procedura del CUSUM adattativo

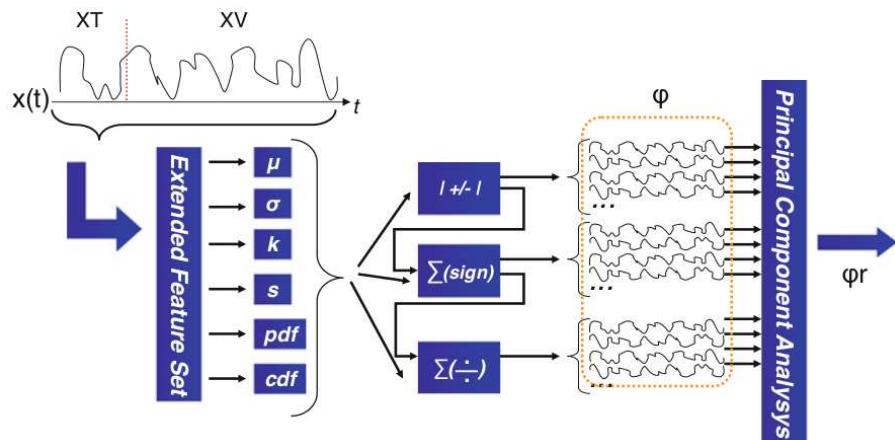


Figura 2.10: Estrazione dei descrittori nel metodo CI-CUSUM

come TS), in modo da valutare la discrepanza tra questi valori. I descrittori utilizzati sono

$$\begin{aligned}\varphi_1(t) &= |\mu_0 - \mu_V|, \varphi_2(t) = |\sigma_0 - \sigma_V|, \\ \varphi_3(t) &= |kurt_0 - kurt_V|, \varphi_4(t) = |skew_0 - skew_V|, \\ \varphi_5(t) &= \int_x |pdf_0(x) - pdf_V(x)| dx, \\ \varphi_6(t) &= \int_x |cdf_0(x) - cdf_V(x)| dx, \\ \varphi_{7 \leq j \leq 12}(t) &= \sum_{v=1}^{t-1} sgn(\varphi_{j-6}(v+1) - \varphi_{j-6}(v)), \\ \varphi_{13 \leq j \leq 24}(t) &= \sum_{v=1}^{t-1} \frac{\varphi_{j-12}(v+1)}{\varphi_{j-12}(v)}\end{aligned}$$

In particolare, i descrittori $\varphi_5(t)$ e $\varphi_6(t)$ valutano la discrepanza tra la densità e funzione di ripartizione correnti e quelle rispettive dell'intervallo di configurazione. I descrittori da $\varphi_7(t)$ a $\varphi_{12}(t)$ vanno ad analizzare i cambiamenti di segno in elementi consecutivi e quelli da $\varphi_{13}(t)$ a $\varphi_{24}(t)$ la somma cumulata del rapporto tra elementi consecutivi. Per ridurre la complessità dello spazio dei descrittori viene applicata un'*analisi delle componenti principali* su φ , in modo da avere la trasformata φ_r . A questo punto, dato che la distribuzione di φ_r non è nota a priori, viene applicato un CUSUM adattativo su di essa.

CDT basati su Intersezione di Intervalli di Confidenza

La famiglia di tecniche che andiamo ad analizzare adesso è in grado di identificare cambi di stazionarietà, in un flusso di dati, monitorando l'evoluzione nel tempo di un certo numero di descrittori estratti dai dati stessi. L'ipotesi principale che viene fatta è che le osservazioni della sequenza monitorata siano *indipendenti e identicamente distribuiti (i.i.d.)* e generati da una distribuzione *gaussiana*.

ICI-CDT In *ICI-CDT* (Intersection of Confidence Intervals Change Detection Test) [9], i descrittori sono estratti attraverso una *fenestratura* dei dati disponibili in sotto-sequenze composte da n istanze. Per ciascuna sotto-sequenza calcoliamo media e varianza campionarie che, per il *teorema del limite centrale* [6], possono essere considerate distribuite in maniera gaussiana. Entrando nel dettaglio, se consideriamo s come l' s -esima sotto-sequenza,

i descrittori estratti sono:

$$M(s) = \frac{\sum_{t=(s-1)n+1}^{ns} x(t)}{n}, \text{ e } V(s) = \left(\frac{\left(\sum_{t=(s-1)n+1}^{ns} (x(t) - M(s))^2 \right)}{n-1} \right)^{h_0},$$

dove il parametro h_0 è un esponente utilizzato per generare un'approssimazione gaussiana della varianza campionaria.

ICI-CDT utilizza una prima sequenza dei dati in cui si assume l'assenza di cambiamenti nella distribuzione. Tale sequenza prende il nome di *insieme di training* e viene indicata con O_{T_0} . Da questa sequenza viene stimato il parametro h_0 e si configura il CDT. La configurazione avviene attraverso le due sequenze di descrittori $\{M(s), s = 1, \dots, S_0\}$ e $\{V(s), s = 1, \dots, S_0\}$, con $S_0 = T_0/n$.

Calcoliamo quindi le medie $\hat{\mu}_{S_0}^M$, $\hat{\mu}_{S_0}^V$ e le deviazioni standard $\hat{\sigma}_{S_0}^M$, $\hat{\sigma}_{S_0}^V$ dei descrittori nell'insieme di training, ovvero:

$$\begin{aligned} \hat{\mu}_{S_0}^M &= \frac{\sum_{s=1}^{S_0} M(s)}{S_0}, \text{ e } \hat{\sigma}_{S_0}^M = \sqrt{\frac{\sum_{s=1}^{S_0} (M(s) - \hat{\mu}_{S_0}^M)^2}{S_0 - 1}}, \\ \text{e} \\ \hat{\mu}_{S_0}^V &= \frac{\sum_{s=1}^{S_0} V(s)}{S_0}, \text{ e } \hat{\sigma}_{S_0}^V = \sqrt{\frac{\sum_{s=1}^{S_0} (V(s) - \hat{\mu}_{S_0}^V)^2}{S_0 - 1}}. \end{aligned}$$

Queste stime definiscono gli *intervalli di confidenza* per i descrittori media e deviazione standard che, sotto la condizione di stazionarietà, sono definiti come:

$$\begin{aligned} I_{S_0}^M &= [\hat{\mu}_{S_0}^M - \Gamma \hat{\sigma}_{S_0}^M, \hat{\mu}_{S_0}^M + \Gamma \hat{\sigma}_{S_0}^M], \\ I_{S_0}^V &= [\hat{\mu}_{S_0}^V - \Gamma \hat{\sigma}_{S_0}^V, \hat{\mu}_{S_0}^V + \Gamma \hat{\sigma}_{S_0}^V], \end{aligned}$$

dove $\Gamma > 0$ è un parametro che controlla l'ampiezza dell'intervalllo di confidenza e, quindi, la probabilità che un descrittore appartenga all'intervalllo sotto l'ipotesi di stazionarietà.

Una volta che la fase di training viene completata, ICI-CDT diventa operativo e può essere utilizzato per identificare cambiamenti nella stazionarietà del flusso di dati. Non appena sono disponibili n dati, viene creata una nuova sequenza s e si calcolano i descrittori che andranno a popolare gli intervallli I_s^M e I_s^V .

A questo punto è possibile applicare la *regola dell'intersezione degli intervalli di confidenza* (ICI-rule) [10]. Questa regola verifica se una nuova istanza di un descrittore possa essere intesa come una realizzazione della già esistente

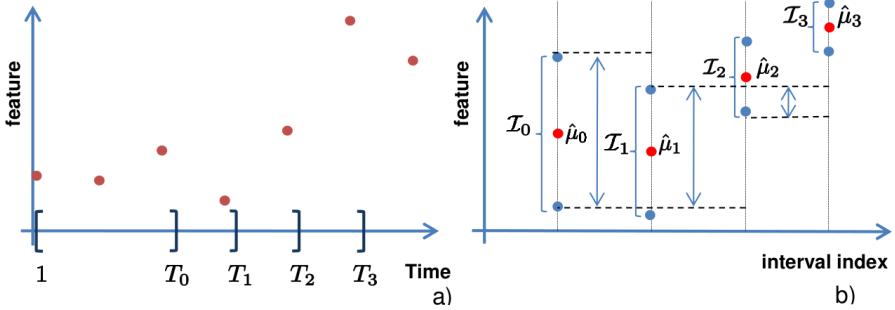


Figura 2.11: Esempio di come opera la regola dell'intersezione degli intervalli di confidenza (ICI-rule)

distribuzione gaussiana. Se ciò non viene verificato, allora è avvenuto un cambiamento nella distribuzione.

Da un punto di vista operazionale, per ciascuna sequenza s vengono calcolati gli intervalli di confidenza dei descrittori come è stato fatto per quelli dell'insieme di training. ICI-CDT individua un cambiamento all'interno della sotto-sequenza \hat{s} se

$$\bigcap_{s < \hat{s}} I_s^M \neq \emptyset \text{ e } \bigcap_{s \leq \hat{s}} I_s^M = \emptyset$$

oppure

$$\bigcap_{s < \hat{s}} I_s^V \neq \emptyset \text{ e } \bigcap_{s \leq \hat{s}} I_s^V = \emptyset$$

e il tempo di identificazione $\hat{T} = n\hat{s}$ corrisponde all'ultimo termine della sotto-sequenza \hat{s} .

La Figura 2.11 illustra un esempio di come opera ICI-rule.

In **a** vengono rappresentati i valori dei descrittori e l'insieme di intervalli $\{[1, T_0], [1, T_1], [1, T_2], [1, T_3]\}$, mentre in **b** sono illustrati gli stimatori delle medie e i loro intervalli di confidenza. In questo caso ICI-rule individua un cambiamento nell'intervallo $[1, T_2]$, dato che $I_0 \cap \dots \cap I_2 \neq \emptyset$ e $I_0 \cap \dots \cap I_3 = \emptyset$. Il principale problema di questa tecnica consiste nella sua *intrinseca* latenza, dovuta al fatto che opera a finestre. Per risolvere questo limite è possibile applicare delle trasformazioni sui dati in modo da renderli approssimativamente gaussiani. Le trasformazioni più utilizzate sono quelle di *Box-Cox* [11], per dati positivi, e di *Manly*, [12] che può essere utilizzato anche per dati negativi. Usando questo tipo di trasformazioni possiamo evitare di lavorare a finestre e utilizzare un approccio *elemento per elemento* [13].

CDT di tipo gerarchico Un altro problema di ICI-CDT consiste nel fatto che esso genera un numero molto elevato di falsi positivi. Per fare in modo che questo numero si abbassi, impattando il meno possibile sul tempo

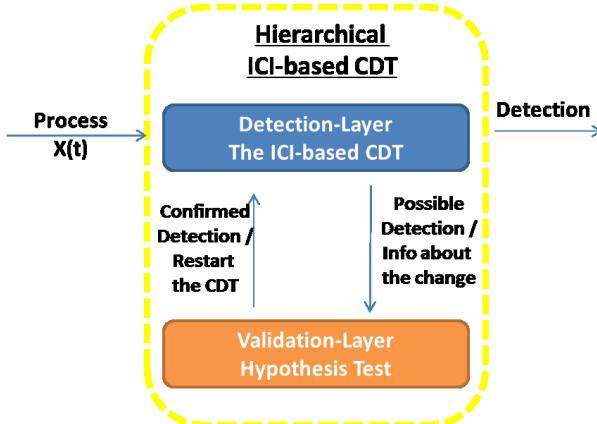


Figura 2.12: Schema del modo di operare del CDT di tipo gerarchico

di esecuzione, è possibile fare un CDT di tipo *gerarchico* [14], basato su due livelli:

- Il primo livello è composto da ICI-CDT, e opera sequenzialmente su tutti i dati.
- Il secondo livello consiste in un test d’ipotesi statistico in grado di validare o rifiutare l’ipotesi di avvenuto cambiamento. Questo livello viene attivato ogni volta che viene identificato un cambiamento da ICI-CDT.

Uno schema di come opera questo tipo di CDT è visualizzato nella Figura 2.12. Il CDT di tipo gerarchico è progettato in modo da avere un primo livello molto efficiente dal punto di vista del tempo di esecuzione, anche se ciò si traduce in un numero elevato di falsi positivi. Questo limite viene mitigato usando nel secondo livello della gerarchia un test più efficiente dal lato del numero di falsi positivi prodotti. Il fatto che il secondo livello ha un tempo di esecuzione maggiore non è un problema, in quanto esso viene attivato solamente nei casi in cui il primo livello della gerarchia segnala un possibile cambiamento nella distribuzione dei dati.

2.4 Tampering Detection

Nei moderni sistemi di videosorveglianza troviamo spesso algoritmi in grado di identificare particolari eventi all’interno della scena ripresa dalla camera. Ad esempio è possibile avere un software in grado di identificare le targhe delle automobili che superano il limite di velocità, oppure la presenza di oggetti incustoditi in una stazione [15]. Affinché questi algoritmi funzionino correttamente, è importante che l’affidabilità del sistema di acquisizione sia preservata. Questa proprietà viene soddisfatta quando:

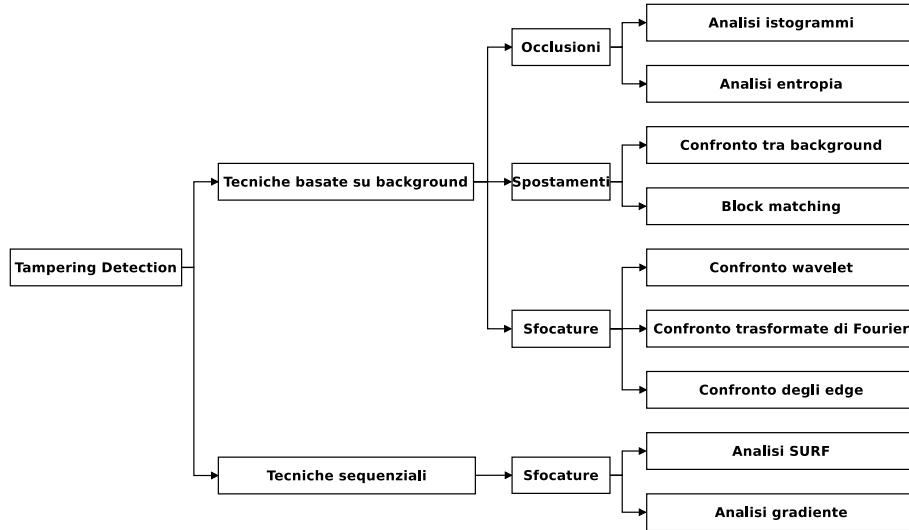


Figura 2.13: Tecniche di tampering detection

- la camera mantiene la stessa inquadratura nell’area di interesse;
- tutti gli elementi della scena sono distinguibili all’interno dell’immagine.

Definiamo tampering un qualsiasi evento che determini un cambio di inquadratura o che non permetta la corretta acquisizione di una parte o della totalità della scena. Possiamo classificare gli eventi di tampering in quattro categorie:

- sfocature,
- spostamenti della camera,
- occlusioni dell’obiettivo,
- guasti della camera.

La letteratura scientifica presenta molte tecniche in grado di affrontare il problema dell’identificazione di questo tipo di eventi. Lo schema in Figura 2.13 mostra le principali tecniche di tampering detection presenti in letteratura. Queste possono essere divise in due categorie:

- tecniche basate su confronto di background,
- tecniche basate su monitoraggio sequenziale.

Nel seguito del paragrafo verranno descritti i principali metodi utilizzati in queste due categorie.

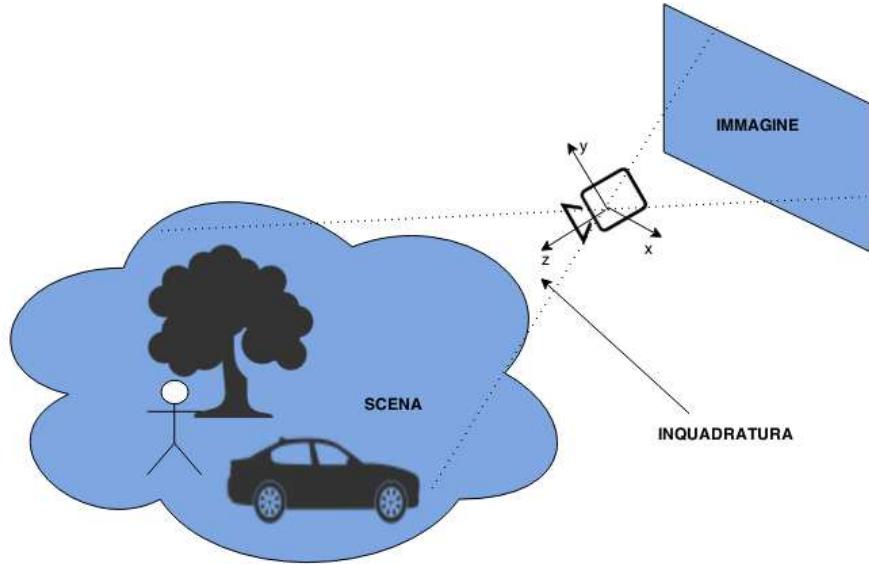


Figura 2.14: Sistema di monitoraggio video

2.4.1 Concetti e terminologia

Prima di concentrarci su come è stato affrontato il problema del tampering detection, definiamo i concetti e i termini che verranno utilizzati nel seguito della trattazione.

Lo scenario che consideriamo è quello di una camera che deve riprendere una particolare *scena*. La posizione e l'orientamento della camera determinano l'*inquadratura* della scena. L'acquisizione, da parte della camera, della scena nell'istante di tempo t viene definita *immagine* o *frame* t -esimo. La Figura 2.14 illustra questi concetti.

Per semplificare la trattazione, considereremo immagini estratte in *scala di grigi*. Ciascuna immagine, quindi, verrà rappresentata come una matrice di pixel, in cui ciascun elemento rappresenta l'intensità luminosa (*luma*) del pixel corrispondente. L'estensione al caso di immagini *RGB* è immediata: possiamo, infatti, applicare i vari algoritmi in maniera indipendente per ciascun canale di colore, per poi mediare i risultati ottenuti.

Nel seguito della trattazione useremo una specifica terminologia. Indicheremo con \mathcal{X} l'insieme dei *pixel* costituenti l'immagine acquisita dalla camera,

$$\mathcal{X} \subset \mathbb{N}^2,$$

e con $x \in \mathcal{X}$ il singolo pixel. Quando vorremo considerare il frame acquisito all'istante di tempo t , useremo z_t , con $t = 1, \dots, \infty$. In particolare, per indicare il valore della *luminosità* del pixel x per il frame t -esimo, useremo

il termine $z_t(x)$, con

$$z_t(x) \in [0, 255] \forall x \in \mathcal{X}.$$

2.4.2 Tecniche basate su confronto di background

Nella maggior parte dei lavori dedicati al problema del tampering detection, il metodo principalmente utilizzato consiste nel confrontare ciascun frame con un modello che viene calcolato a partire dalle osservazioni precedenti. Tale metodo è ampiamente utilizzato in vari ambiti della visione artificiale e prende il nome di *background subtraction*. Una tecnica generale di calcolo del background è presentata in [16]. Il valore del modello di background per il pixel x è calcolato, in maniera ricorsiva, secondo la seguente formula:

$$B_{t+1}(x) = \begin{cases} aB_t(x) + (1-a)z_t(x) & , \text{ se } |z_t(x) - z_{t-1}(x)| \leq T_t(x) \\ B_t(x) & , \text{ se } |z_t(x) - z_{t-1}(x)| > T_t(x) \end{cases}, \forall x \in \mathcal{X},$$

dove $0 < a < 1$ è chiamato *parametro di aggiornamento (update parameter)* e $T_t(x)$ è una soglia che permette di identificare un cambiamento sostanziale di luminosità nel pixel (x) . Questa soglia viene aggiornata in maniera ricorsiva secondo la seguente formula:

$$\Gamma_{t+1}(x) = \begin{cases} a\Gamma_t(x) + (1-a)(c|z_t(x) - B_t(x)|) & , \text{ se } |z_t(x) - z_{t-1}(x)| \leq \Gamma_t(x) \\ \Gamma_t(x) & , \text{ se } |z_t(x) - z_{t-1}(x)| > \Gamma_t(x) \end{cases}, \forall x \in \mathcal{X},$$

dove $c > 1$ e $0 < a < 1$. Lo stesso modello viene utilizzato in altri lavori, come ad esempio in [17] e [18], mentre una variante molto usata consiste nel calcolare il modello di background a partire dall'*estrazione dei contorni* di ciascun frame ([19], [20]). Un approccio diverso, ma comunque riconducibile allo stesso filone, lo troviamo in [21], dove il background è sostituito da un *buffer* contenente gli ultimi n frame acquisiti dalla camera, che vengono confrontati con l'ultima osservazione per identificare eventi di tampering.

Identificazione di occlusioni

L'evento di occlusione avviene quando un oggetto opaco viene posto vicino alla camera, in modo da coprire la scena ripresa. In [16] e [17] questo evento è associato a un cambiamento nella struttura dell'*istogramma* del frame occluso rispetto a quello del background. In particolare, se z_t è un frame in cui è avvenuta un'occlusione, ci si aspetta che il suo istogramma contenga i valori più alti in un intervallo più ristretto rispetto a quello del background B_t , in quanto la maggior parte dell'immagine prenderà il colore dell'oggetto occludente o diventerà più scura.

Indichiamo con $H_b(\cdot)$ il valore del b -esimo bin dell'istogramma di un frame,

con $1 \leq b \leq 32$. Per identificare un evento di occlusione vengono considerate due disequazioni:

$$\begin{aligned} & (H_{\max(H(z_t))-1}(z_t) + H_{\max(H(z_t))}(z_t) + H_{\max(H(z_t))+1}(z_t)) \\ & > \Gamma_1 (H_{\max(H(z_t))-1}(B_t) + H_{\max(H(z_t))}(B_t) + H_{\max(H(z_t))+1}(B_t)), \\ & \sum_{b=1}^{32} H_b(|z_t - B_t|) > \Gamma_2 \sum_{b=1}^k H_b(|z_t - B_t|), \end{aligned}$$

dove $\Gamma_1 > 1$, $\Gamma_2 > 1$ e $0 \leq k < 32$ sono soglie che possono essere modificate in base alla sensibilità che si richiede da parte dell'algoritmo.

Un approccio simile è presente in [19], [20] e [22], in cui l'evento di occlusione è associato a un abbassamento dell'*entropia*:

$$E = - \sum_k p_k \ln(p_k),$$

dove p_k rappresenta la probabilità che il livello di grigio k sia presente all'interno dell'immagine.

Per riuscire a identificare delle occlusioni *parziali* le tecniche descritte sopra non sono efficaci, in quanto tali eventi non sono in grado di modificare in maniera sostanziale la struttura dell'istogramma dei frame. Una soluzione ([20]) consiste nel dividere ciascuna immagine in un certo numero di *blocchi* della stessa dimensione, in modo da applicare le tecniche descritte sopra per ciascuna partizione.

Identificazione di spostamenti della camera

Quando viene spostata la camera, in modo cambi l'inquadratura della scena, l'immagine di background B_i viene lentamente aggiornato in modo da riflettere i cambiamenti avvenuti nei frame. In [17] il metodo proposto per identificare uno spostamento della camera consiste nel confrontare l'immagine di background B_t con B_{t-k} , ovvero con un secondo modello *ritardato* di k frame, dove $k \in \mathbb{Z}^+$. L'approccio consiste nel calcolare un *valore di proporzione* P , ottenuto dal confronto tra i due modelli:

$$P = \begin{cases} P + 1 & , \text{ se } B_t(x) \neq B_{t-k}(x) \\ P & , \text{ se } B_t(x) = B_{t-k}(x) \end{cases}.$$

Lo spostamento della camera viene identificato quando $P > \Gamma \cdot K$, dove $0 < \Gamma < 1$ è un valore di soglia scelto in base alla sensibilità che si vuole dare all'algoritmo e K è il numero totale di pixel dell'immagine.

Un altro approccio è quello di utilizzare tecniche di *block matching*. L'algoritmo di block matching fornisce due parametri: il primo parametro, m , indica il *vettore della traslazione* tra il background e il frame corrente, mentre il secondo parametro è lo ZNCC relativo a quel vettore. In [20] e [19]

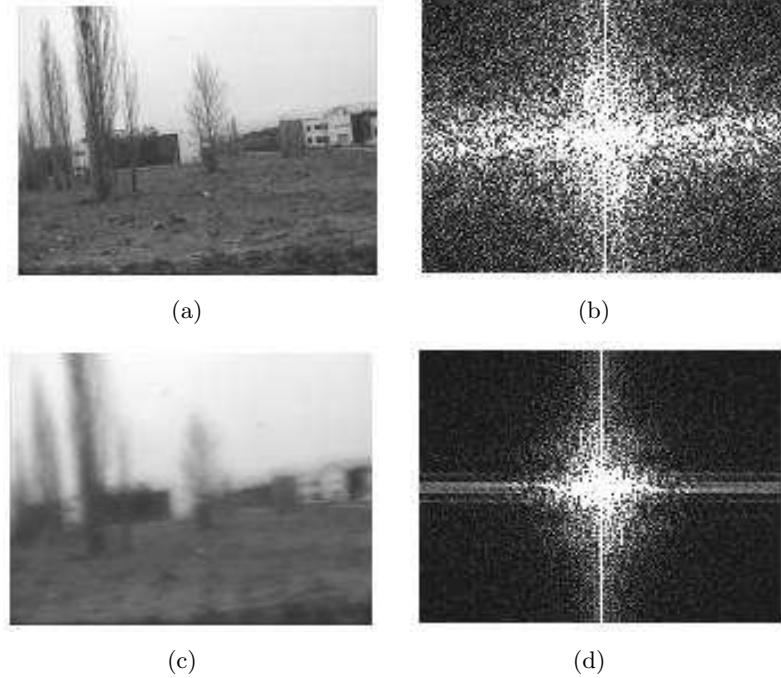


Figura 2.15: Comportamento della trasformata di Fourier nel caso di sfocatura

l'algoritmo calcola il vettore di spostamento m del frame corrente rispetto al background, e calcola la *zero-mean normalized cross correlation* (ZNCC) [23]:

$$ZNCC_t(m) = \frac{\sum_{x \in \mathcal{X}} (B_{t-1}(x) - \mu_{B_{t-1}})(z_t(x+m) - \mu_{z_t})}{\sigma_{B_{t-1}} \sigma_{z_t}},$$

dove μ_{z_t} e σ_{z_t} rappresentano la media e la deviazione standard della luminosità nell'immagine z_t . Lo spostamento viene individuato quando il vettore m ha una lunghezza minima e lo ZNCC corrispondente supera una certa soglia. Un metodo simile viene utilizzato anche in [24], con la differenza che, invece di analizzare la correlazione dei pixel, viene analizzata quella degli istogrammi.

Identificazione di sfocature

La conseguenza di un evento di sfocatura è la perdita di dettagli nell'immagine. In [16] e [17] questo fenomeno è associato a una diminuzione dell'energia ad alta frequenza. Per analizzare questo cambiamento, [16] confronta ciascun frame con il background nel dominio delle *wavelet* [25], mentre [17] utilizza il dominio della *trasformata di Fourier* [26]. Nella Figura 2.15 vediamo un esempio di come si comporta la trasformata di Fourier nel caso di sfocature: nel passaggio da un frame nitido (Fig. 2.15(a)) a uno sfocato

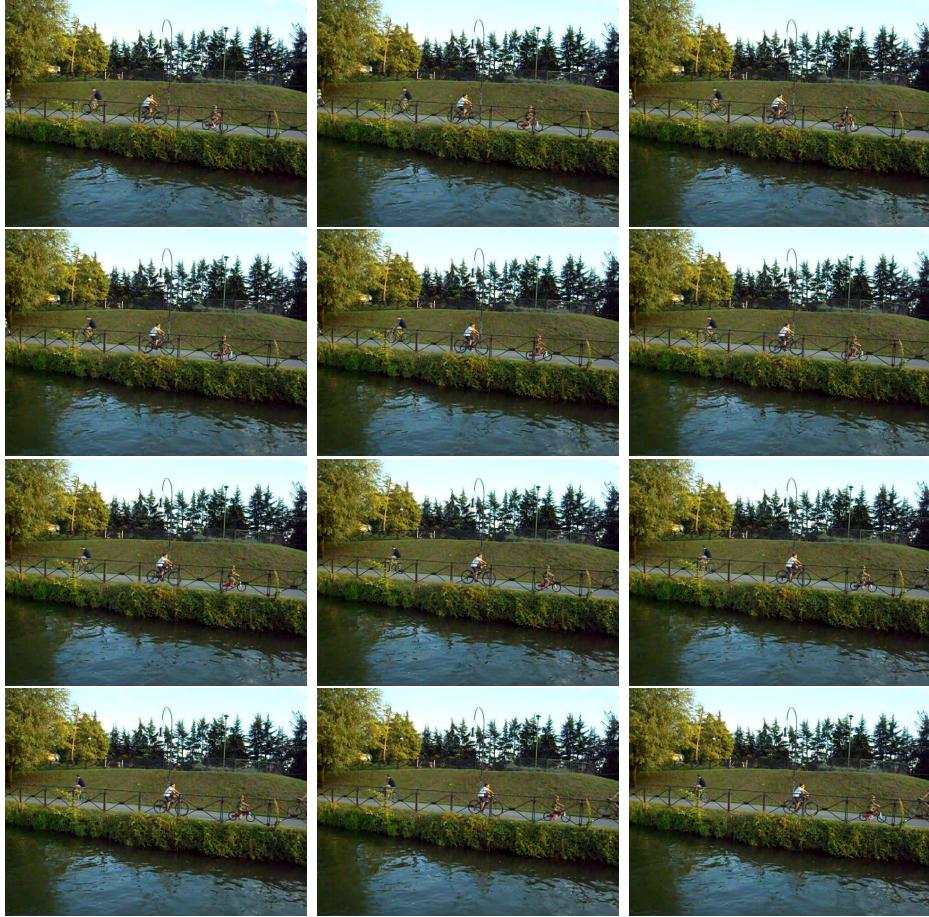


Figura 2.16: Sequenza di frame acquisiti a 30 fps

(Fig. 2.15(c)) abbiamo un crollo delle componenti ad alta frequenza nelle trasformate di Fourier (rispettivamente Fig. 2.15(b) e Fig. 2.15(d)). L’evento di tampering viene individuato quando l’*energia media* della trasformata di Fourier (o di quella wavelet) del frame corrente $E_{HF}(z_t)$ è Γ volte minore di quella del background $E_{HF}(B_t)$:

$$E_{HF}(z_t) \leq \Gamma \cdot E_{HF}(B_t),$$

dove $0 < \Gamma < 1$ è un valore di soglia scelto in base alla sensibilità che si vuole dare all’algoritmo.

Un altro approccio consiste nell’analizzare la perdita di dettagli confrontando i contorni (*edges*) del frame corrente con quelli del background. Questo metodo, utilizzato in [22], [20], [19] e [24], consiste nell’estrarre i contorni dalle immagini secondo il metodo di Sobel [27] o di Canny [28], e confrontare il numero di pixel dei contorni con quelli del background. Quando il numero di pixel dei contorni del frame corrente $\sum edges(z_t)$ è Γ volte più piccolo di

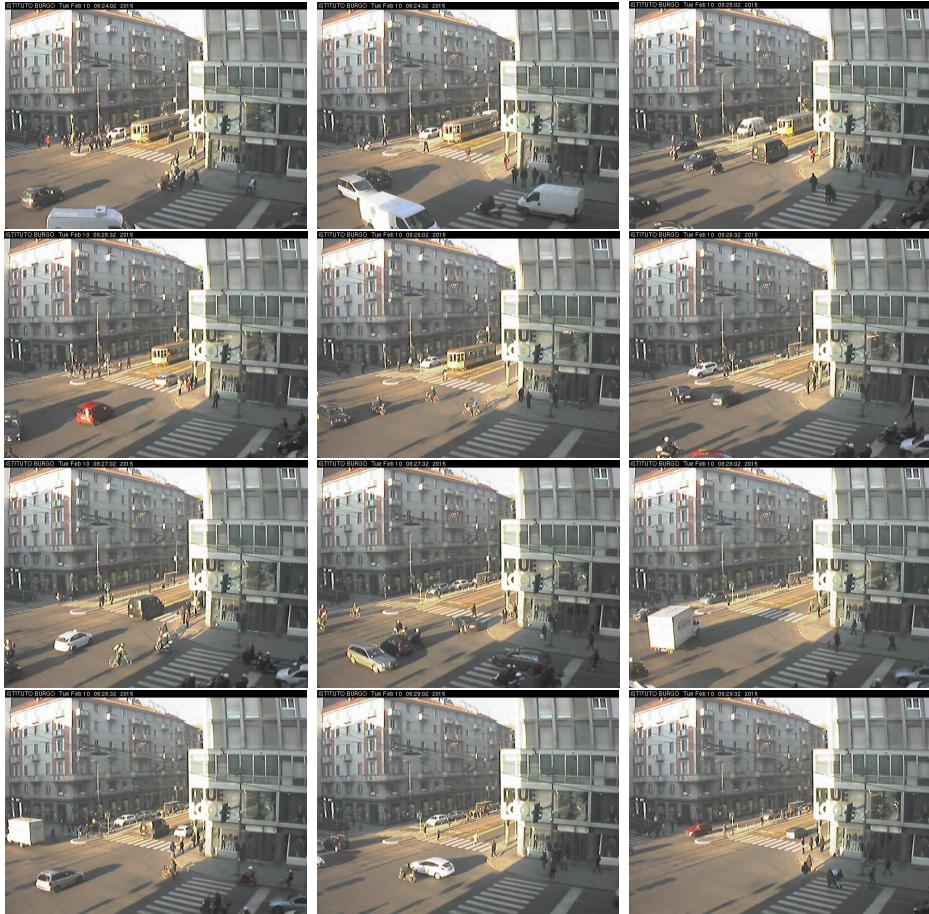


Figura 2.17: Sequenza di frame acquisiti ogni 30 secondi

quello del background $\sum edges(B_t)$:

$$\sum edges(z_t) \leq \Gamma \sum edges(B_t),$$

dove $0 < \Gamma < 1$ è un valore di soglia scelto in base alla sensibilità che si vuole dare all'algoritmo.

2.4.3 Tecniche basate su monitoraggio sequenziale

Le tecniche viste finora, come è stato detto nel Paragrafo 2.4.2, permettono di aggiornare ciascun frame con un modello della scena che viene calcolato in base alle osservazioni precedenti. Questo approccio risulta fattibile nel caso in cui la camera operi con un framerate *continuo*, solitamente tra i 30 frame per secondo (*fps*) e i 2 fps. In questi casi, infatti, possiamo considerare che, tra un frame e il successivo, non avvengano grandi cambiamenti all'interno della scena e non cambi eccessivamente la luminosità media. La Figura

2.16 mostra come, in un’acquisizione fatta a 30 fps, le differenze tra frame consecutivi siano quasi inesistenti. Un evento di tampering può, quindi, essere identificato in maniera molto semplice, in quanto una differenza molto elevata tra il frame analizzato e il background può essere dovuto solamente a un evento di tampering sulla camera. La Figura 2.17 mostra, invece, un esempio di frame acquisiti ogni 30 secondi. Notiamo come le differenze tra immagini consecutive, in questo caso, siano più marcate rispetto al caso dell’acquisizione continua. Questo non permette l’impiego di un modello di background per il problema del tampering detection. Evidenziamo, inoltre, che le tecniche viste finora richiedono che il sistema di monitoraggio possieda elevate capacità computazionali, in quanto l’aggiornamento del background e l’analisi di tampering detection non può rallentare la frequenza di acquisizione.

Un approccio alternativo consiste nel monitorare nel tempo il comportamento di alcuni indicatori estratti dalle singole immagini acquisite. Si presuppone che, quando il sistema di monitoraggio opera in condizioni di funzionamento *ottimali*, gli indicatori analizzati presentino una certa *stazionarietà*, ovvero siano considerati dei campioni *indipendenti* tra loro e distribuiti secondo una stessa funzione di ripartizione. L’evento di tampering viene considerato come un *cambiamento nella stazionarietà* di questi indicatori.

Il monitoraggio può avvenire utilizzando tecniche statistiche, come ad esempio *change-point method* (CPM) [7] o *change-detection test* [29].

Troviamo alcuni esempi nell’identificazione di sfocature: in [18] la soluzione consiste nel monitorare nel tempo il *numero di SURF* [30], in quanto tali descrittori decrementano in maniera considerevole il loro numero in presenza di sfocature. Notiamo, però, che l’utilizzo di una tecnica del genere richiede un elevato numero di calcoli per ricavare le SURF. Il metodo, quindi, si presta poco a essere utilizzato su sistemi di monitoraggio a basso consumo. Un altro esempio è dato da [31], dove le sfocature vengono identificate monitorando l’*energia media del gradiente* delle immagini acquisite:

$$m_t = \mathcal{M}[z_t] = \int_{\mathcal{X}} \|\nabla z_t(x)\|_1 dx,$$

dove $\|\cdot\|_1$ si riferisce alla *norma \mathcal{L}_1* . Per identificare il cambio di stazionarietà di questo indicatore vengono utilizzate tecniche di CDT basate su *somme cumulate* (CUSUM) [8].

Il test statistico utilizzato non richiede alcuna informazione *a priori* del processo che si sta monitorando, e sfrutta una sequenza iniziale $\{m_t\}, t = 1, \dots, T$ di indicatori estratti da frame non sfocati, in modo da configurare automaticamente i suoi parametri. Tale sequenza prende il nome di *training set* e, assumendo che i dati abbiano una distribuzione *gaussiana*, permette di stimare i parametri di m_t in assenza di sfocature che determinano l’*ipotesi nulla* (indicata con Θ^0), e di definire le *ipotesi alternative*, indicate con Θ^1 , che identificano qualsiasi cambiamento non stazionario.

Per garantire una stima accurata dei parametri, [31] consiglia di utilizzare un training set ampio, ad esempio $T > 400$. Il test opera su *sotto-sequenze* di misure m_t (ad esempio da 20 misure) e stima la transizione da Θ^0 a Θ^1 misurando la *log-verosimiglianza* tra la pdf in assenza di sfocature e le pdf delle varie ipotesi alternative nella sotto-sequenza τ , ovvero

$$r(\tau) = \ln \frac{N_{\Theta^1}(\phi_\tau)}{N_{\Theta^0}(\phi_\tau)},$$

dove ϕ_τ è il valore medio della misura m_t nella sotto-sequenza τ , e N_Θ è la *distribuzione gaussiana multivariata* parametrizzata in Θ .

Il metodo CUSUM considera la *somma cumulata*

$$S(\tau) = \sum_{t=1}^{\tau} r(t)$$

e identifica un cambiamento in m_t quando

$$g(\tau) = S(\tau) - \nu(\tau),$$

ovvero la differenza tra il valore della somma cumulata all'istante τ e il suo valore minimo nel tempo

$$\nu(\tau) = \min_{t=1,\dots,\tau} S(t),$$

superà una certa soglia h .

L'utilizzo di un descrittore leggero come l'energia media del gradiente permette di operare con una bassa complessità computazionale, tanto da poter essere utilizzato come soluzione a livello di nodo per una *wireless multimedia sensor network* (WMSN) [32]. D'altro canto, confrontato con le tecniche descritte nel Paragrafo 2.4.2, l'utilizzo di tecniche sequenziali genera un numero più alto di *falsi allarmi*, il che si traduce, all'interno di una rete di sensori, in un aumento dei messaggi di allarme inviati, che devono essere considerati, quindi, come un costo.

Capitolo 3

Impostazione del problema di ricerca

In questo capitolo descriviamo il problema affrontato dall'algoritmo di tampering detection, in maniera formale e rigorosa.

Nel Paragrafo 3.1 illustriamo il modello delle osservazioni e gli eventi che siamo interessati a identificare.

Nel Paragrafo 3.2 formalizziamo il concetto di tampering detection.

3.1 Modello delle osservazioni

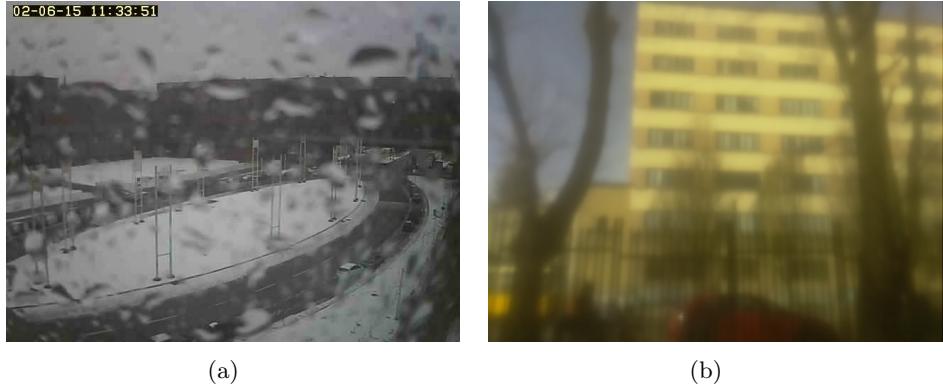
Il nostro campo di osservazione si concentra su quegli eventi che si interpongono tra la scena ripresa da una camera e il sensore che acquisisce le immagini. Non vogliamo, cioè, identificare degli eventi particolari che avvengono nella scena, come un oggetto lasciato incustodito [15], bensì vogliamo identificare quegli eventi che non permettano al sensore di riprendere in maniera ottimale la scena, quali ad esempio sfocature o spostamenti della camera. Nel seguito cerchiamo di dare una definizione formale di questi eventi.

3.1.1 Sfocatura

Il fenomeno della sfocatura avviene quando un elemento trasparente o semi-trasparente si interpone tra la lente della camera e la scena ripresa, oppure quando viene cambiata la messa a fuoco, causando una perdita nei dettagli della scena ripresa.

Nella Figura 3.1 sono mostrati degli esempi in cui sono presenti delle sfocature. Queste possono essere di origine diversa:

- dovute a *cause naturali*, come ad esempio dell'acqua piovana che si deposita sulla lente (Figura 3.1(a)), o la condensa dovuta all'umidità e



(a)

(b)

Figura 3.1: Esempi di sfocature

alle basse temperature, oppure un raggio di sole incidente sull'obiettivo della camera;

- per *intervento dell'uomo*, che a sua volta può avvenire in maniera intenzionale (e in questo caso si può parlare di *manomissione*) oppure non intenzionale. Ad esempio, si può direttamente intervenire sulla messa a fuoco, nel caso sia possibile cambiarla manualmente; oppure (come nel caso della Figura 3.1(b)) è possibile applicare una sostanza semitrasparente sulla lente della camera, come il gas di un deodorante spray.

Inoltre, possiamo notare come nella Figura 3.1(b) la sfocatura riguardi tutta l'immagine, mentre nella Figura 3.1(a) la sfocatura si concentri solo in alcune aree (quelle dove sono presenti le gocce). Nel primo caso si parlerà, quindi, di sfocatura *totale*, mentre nel secondo caso di sfocatura *parziale*.

Riprendendo la trattazione presente in [31], questo fenomeno può essere modellato come un operatore di *degradazione* \mathcal{D} applicato a un'immagine y , considerata priva di errori, i.e.,

$$z = \mathcal{D}[y]. \quad (3.1)$$

In particolare, all'interno dell'operatore \mathcal{D} si può considerare il contributo dovuto a un operatore di *sfocatura* \mathcal{B} (dall'inglese *blur*) e un termine aleatorio η corrispondente al rumore, i.e.,

$$z(x) = \mathcal{D}[y](x) = \mathcal{B}[y](x) + \eta(x), \quad x \in \mathcal{X} \quad (3.2)$$

dove, come abbiamo specificato nel Paragrafo 2.4.1, indichiamo con x le coordinate dei *pixel* dell'immagine e con \mathcal{X} l'insieme dei pixel che formano l'immagine. Per praticità possiamo assumere che l'operatore di blur sia *lineare*. Quindi, considerando l'immagine come un dato *continuo*,

$$\mathcal{B}[y](x) = \int_{\mathcal{X}} y(s)h(x, s)ds, \quad (3.3)$$



Figura 3.2: Sequenza di otto frame consecutivi acquisiti ogni minuto

dove $h(x, \cdot)$ rappresenta la *risposta all'impulso*, *point spread function* (PSF), della sfocatura sul pixel x . L'effetto di $h(x, \cdot)$ è quello di rendere le differenze di intensità, tra pixel adiacenti, più morbide (*smooth*). Nel caso in cui la sfocatura sia applicata sulla totalità dell'immagine e fosse *spazio invariante* (come nel caso della Figura 3.1(b)), allora è possibile modellare l'operatore di blur come una *convoluzione*¹:

$$\mathcal{B}[y](x) = \int_{\mathcal{X}} y(s)h(s - x)ds, \quad (3.4)$$

dove $h(\cdot)$ è un filtro gaussiano o uniforme.

Nel caso più generale possiamo considerare che la camera acquisisca una sequenza di N osservazioni (dove N può tendere all'infinito) $\{z_t\}, t = 1, \dots, N$, quindi la formula (3.2) si può riscrivere come

$$z_t(x) = \mathcal{D}_t[y_t](x) = \mathcal{B}_t[y_t](x) + \eta(x), \quad x \in \mathcal{X}. \quad (3.5)$$

La sequenza delle immagini $\{y_t\}, t = 1, \dots, N$, può variare in maniera significativa nel suo contenuto, anche nel caso in cui la vista sia la stessa. Un esempio è illustrato nella Figura 3.2, in cui le immagini riprese dalla camera sono acquisite ogni minuto. Nonostante l'inquadratura non cambi tra le acquisizioni, il contenuto delle singole immagini varia parecchio, a causa del continuo passaggio di automobili e pedoni. Questo problema fa sì che l'identificazione delle sfocature non possa avvenire facendo un semplice confronto tra frame consecutivi, in quanto avremmo un numero troppo elevato di falsi positivi. Infatti, nel caso in cui avessimo un riscontro negativo dal confronto tra due frame ($z_t \neq z_{t+1}$), sarebbe difficile capire se è cambiato il contenuto delle immagini ($y_t \neq y_{t+1}$) o l'operatore di sfocatura ($\mathcal{B}_t \neq \mathcal{B}_{t+1}$).

3.1.2 Spostamento della camera

Lo spostamento della camera avviene quando cambia la sua inquadratura. Le cause possono essere, ancora una volta, di tipo naturale, ad esempio una

¹Il blur convoluzionale è quello che abbiamo utilizzato per generare, in maniera sintetica, sequenze con immagini sfocate.



(a)

(b)

Figura 3.3: Esempio di spostamento della camera

raffica di vento che sposta la camera, oppure dovute a un intervento umano, uno spostamento intenzionale per evitare che la scena venga ripresa. Un esempio è mostrato nella Figura 3.3, dove vediamo che tra il frame riportato in Figura 3.3(a) e quello in Figura 3.3(b) c'è stato un evento che ha spostato la camera. Possiamo formalizzare il concetto di spostamento della camera nel modo seguente: consideriamo la sequenza $\{y_t\}$ di immagini generate da una camera in una certa posizione, e la sequenza $\{w_t\}$ di immagini generate dalla stessa camera in una posizione differente.

Possiamo modellare la sequenza di immagini $\{z_t\}$ in cui avviene uno spostamento della camera all'istante T^* nel seguente modo:

$$z_t(x) = \begin{cases} y_t(x) + \eta(x) & \text{per } t < T^* \\ w_t(x) + \eta(x) & \text{per } t \geq T^* \end{cases}, \quad (3.6)$$

dove $\eta(x)$ è un rumore stazionario.

In questa formulazione abbiamo considerato lo spostamento come un fenomeno *istantaneo*; in generale, possiamo considerare una fase *transitoria* in cui l'inquadratura della camera cambia a ogni frame acquisito, fino a raggiungere la posizione finale. Dato che, nella nostra applicazione, la camera opera con framerate molto bassi (come ad esempio un'immagine al minuto), possiamo considerare lo spostamento come istantaneo e, quindi, tenere come riferimento il modello (3.6).

Anche per lo spostamento della camera vale la considerazione fatta nel caso della sfocatura: il contenuto delle immagini varia con il passare del tempo, quindi identificare lo spostamento confrontando frame consecutivi nel tempo genera un alto numero di falsi allarmi, come verrà mostrato dalla fase sperimentale.



Figura 3.4: Esempi di occlusione

3.1.3 Occlusione e guasti della camera

Il fenomeno dell’occlusione avviene quando un oggetto opaco si pone a ridosso della camera, impedendo la visione di una parte, se non la totalità, della scena. Esempi di occlusione sono illustrati in Figura 3.4. Può succedere (Fig. 3.4(a)) che un oggetto venga posto intenzionalmente davanti alla lente della camera, in modo da coprire la scena, oppure (Fig. 3.4(b)), a causa di una nevicata, può avvenire che della neve si depositi sulla lente e, quindi, venga compromessa la corretta acquisizione.

Quando parliamo di guasti, invece, possiamo considerare due casi:

- guasto della *trasmissione*;
- guasto del *sensore*.

Il primo caso può avvenire quando considerano delle WMSN, in particolare quando i frame vengono trasmessi attraverso la rete dai sensori verso un nodo centrale. Quando avviene un guasto nella rete parte dell’immagine non viene trasmessa, come illustrato nella Figura 3.5(a). Il caso del sensore guasto, come ad esempio quello nella Figura 3.5(b), comporta generalmente un aumento del rumore presente nel frame.

Questi problemi non sono stati affrontati durante il corso della tesi, comunque le tecniche messe a punto per rilevare spostamenti e sfocature possono essere utilizzate anche per rilevare questo tipo di eventi.

3.2 Tampering detection

Nel Paragrafo 2.4 abbiamo introdotto in maniera molto generale il concetto di tampering detection. In questo paragrafo diamo una definizione più formale al problema. L’algoritmo di tampering detection consiste nell’analizzare la sequenza dei frame generati dalla camera in modo da determinare



Figura 3.5: Esempi di guasti

un possibile cambiamento nell'inquadratura della scena o una sfocatura. Consideriamo il caso generale di una sequenza di frame $\{z_t\}, t = 1, \dots, \infty$. Per $t < T^*$ abbiamo che i frame vengono acquisiti in condizioni di funzionamento normale:

$$z_t(x) = y_t(x) + \eta(x), \forall x \in \mathcal{X}, \text{ per } t = 1, \dots, T^* - 1.$$

All'istante di tempo $t = T^*$ avviene un evento di tampering, il quale compromette i frame per $t \geq T^*$. In particolare, nel caso di una sfocatura avremo

$$z_t = \mathcal{B}_t[y_t](x) + \eta(x), \forall x \in \mathcal{X}, \text{ per } t \geq T^*,$$

mentre nel caso di uno spostamento della camera avremo

$$z_t = w_t(x) + \eta(x), \forall x \in \mathcal{X}, \text{ per } t \geq T^*.$$

L'obiettivo dell'algoritmo è quello di stimare l'istante T^* .

Come abbiamo illustrato precedentemente, discriminare tra cambiamenti avvenuti a livello di contenuto della scena e cambiamenti dovuti a tampering può diventare complicato confrontando direttamente i frame tra loro, in quanto stiamo operando con framerate bassi. Non utilizzeremo, quindi, le tecniche basate su background che sono state descritte nel Paragrafo 2.4.2. Nel Capitolo 4 vedremo la soluzione che abbiamo deciso di utilizzare per lo sviluppo dell'algoritmo di tampering detection.

Capitolo 4

Soluzione proposta

In questo capitolo descriviamo la soluzione che abbiamo proposto per risolvere il problema del tampering detection, formulato nel Capitolo 3.

Nel Paragrafo 4.1 descriviamo quali sono gli indicatori che abbiamo deciso di monitorare per individuare gli eventi di tampering.

Nel Paragrafo 4.2 illustriamo in che modo vengono monitorati questi indicatori in modo da individuare degli eventi specifici.

Nel Paragrafo 4.3 descriviamo, infine, l'algoritmo di segmentazione che utilizziamo, assieme al monitoraggio, per individuare eventi di spostamento della camera.

4.1 Indicatori utilizzati per identificare gli eventi di tampering

L'approccio che abbiamo considerato per risolvere il problema del tampering detection consiste nell'estrarrre, da ciascun frame ripreso dalla camera, degli indicatori *scalari* che possano essere monitorati nel tempo, in modo da individuare un cambio di distribuzione o un *outlier* associabile a un evento di tampering. In particolare abbiamo deciso di considerare un indicatore in grado di verificare la presenza di sfocature globali all'interno della scena e un altro in grado di identificare gli eventi di spostamento della camera. Il resto del paragrafo è dedicato alla descrizione di questi indicatori.

4.1.1 Misura della sfocatura nell'immagine

Nel Paragrafo 3.1.1 abbiamo modellizzato il fenomeno della sfocatura secondo la formula (3.5):

$$z_t(x) = \begin{cases} y_t(x) + \eta(x) & \text{per } t < T^* \\ \mathcal{B}_t[y_t](x) + \eta(x) & \text{per } t \geq T^* \end{cases}, \forall x \in \mathcal{X}.$$

Ricavare un indicatore in grado di misurare direttamente il grado di sfocatura di un’immagine è difficile. Quello che è possibile fare, come proposto in [31], è misurare *indirettamente* il grado di sfocatura di un’immagine z_t basandosi sulle frequenze dell’immagine.

Come abbiamo visto nel Paragrafo 3.1.1, l’operatore di sfocatura \mathcal{B} ha come effetto principale quello di rendere le differenze di intensità tra pixel adiacenti più morbide (*smooth*). In base a questa considerazione è possibile identificare un evento di sfocatura andando a monitorare l’*energia media del gradiente* di ciascuna immagine:

$$g(t) = \mathcal{G}[z_t] = \frac{\sum_{\mathcal{X}} \|\nabla z_t(x)\|_2^2}{|\mathcal{X}|}, \quad (4.1)$$

dove abbiamo indicato con $|\mathcal{X}|$ la *cardinalità* dell’insieme dei pixel \mathcal{X} , e con $\|\cdot\|_2$ la norma di tipo \mathcal{L}_2 ¹.

Lavorando nel dominio discreto delle immagini digitali, possiamo calcolare le derivate dell’intensità luminosa (*luma*) per mezzo di convoluzioni con filtri derivativi. In particolare, per il calcolo delle derivate orizzontali abbiamo utilizzato il seguente filtro f_h :

$$f_h = f \circledast \begin{bmatrix} 1 & 0 & -1 \end{bmatrix},$$

mentre per il calcolo delle derivate verticali abbiamo utilizzato il seguente filtro f_v :

$$f_v = f \circledast \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix},$$

dove abbiamo indicato con \circledast l’operatore di convoluzione. Il filtro f , invece, è ottenuto tramite un campionamento della *funzione gaussiana* con media 0 e deviazione standard σ

$$h(i, j) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{i^2 + j^2}{2\sigma^2}\right), \quad (4.2)$$

e ponendo il valore massimo di questa funzione nel centro del filtro. Con questi filtri è possibile calcolare la *norma del gradiente* nel seguente modo:

$$\|\nabla z_t(x)\|_2^2 = (z_t \circledast f_h)(x)^2 + (z_t \circledast f_v)(x)^2. \quad (4.3)$$

Una volta calcolata la norma del gradiente è possibile farne la media come specificato in (4.1). Il risultato finale è un indicatore *scalare* per ciascun frame acquisito, che può essere monitorato per individuare eventi di sfocature. In particolare ci aspettiamo che l’evento di sfocatura provochi un abbattimento del valore di g .

¹ $\|x\|_2 = \sqrt{\sum_t x_t^2}$



(a)

(b)

Figura 4.1: Esempio di cambi di luminosità tra il giorno e la notte

4.1.2 Misura dello spostamento della camera

Nel Paragrafo 3.1.2 abbiamo modellizzato il fenomeno dello spostamento della camera secondo (3.6)

$$z_t(x) = \begin{cases} y_t(x) + \eta(x) & \text{per } t < T^* \\ w_t(x) + \eta(x) & \text{per } t \geq T^* \end{cases}, \forall x \in \mathcal{X}$$

dove T^* indica l'istante in cui avviene il cambiamento.

Uno spostamento della camera, quindi, può essere rilevato come un cambiamento *globale* dei valori di intensità luminosa (*luma*) nei pixel dell'immagine. In base a questo è possibile identificare un evento di spostamento della camera andando a monitorare l'*energia media della luma* di ciascuna immagine:

$$l(t) = \mathcal{L}[z_t] = \frac{\sum_{\mathcal{X}} z_t(x)}{|\mathcal{X}|}, \quad (4.4)$$

dove abbiamo indicato con $|\mathcal{X}|$ la *cardinalità* dell'insieme dei pixel \mathcal{X} .

Il risultato finale è un indicatore *scalare* per ciascun frame acquisito, che può essere monitorato per individuare eventi di spostamenti della camera.

4.1.3 Comportamento degli indicatori nel tempo

Analizzando alcune sequenze video abbiamo notato che, anche in assenza di eventi di tampering, vi sono alcuni fattori che sono in grado di far variare il valore degli indicatori estratti. Tra i più importanti abbiamo:

- *Cambiamenti di luminosità* che avvengono nel corso della giornata. Se consideriamo l'esempio in Figura 4.1, possiamo notare come, nel passaggio dal giorno (Figura 4.1(a)) alla notte (Figura 4.1(b)), le differenze di luminosità siano elevate.



(a)

(b)

Figura 4.2: Esempio di presenza di sfocature dovute all'aumento del tempo di esposizione della camera

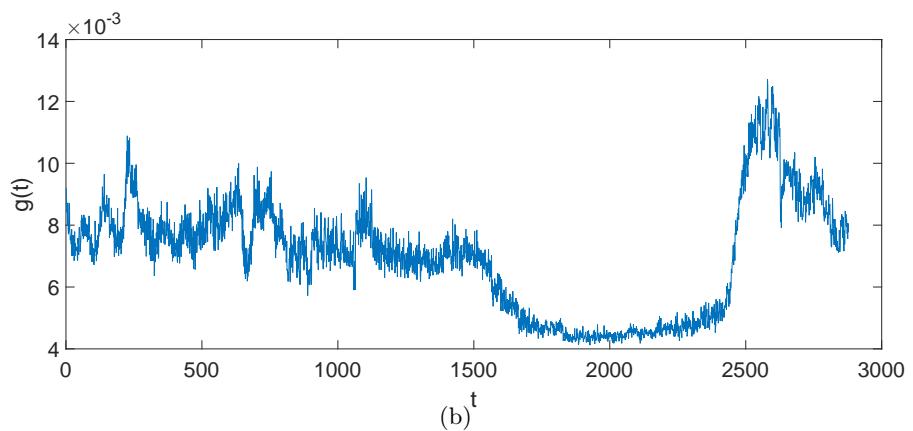
- *Dinamicità della scena.* La ripresa di una scena dinamica, come ad esempio una strada, ha come risultato che ciascun frame sia diverso dagli altri. Ciò si traduce in una variabilità elevata degli indicatori che abbiamo utilizzato. Inoltre, col passare del tempo, può succedere che cambi anche il *grado di dinamicità* della scena. Considerando ancora l'esempio della strada avremo dei momenti in cui il traffico è più intenso (nelle cosiddette *ore di punta*) e altri in cui le macchine passano meno spesso (tipicamente durante la notte).
- *Tempo di esposizione della camera.* Solitamente le camere sono in grado di configurare in maniera automatica alcuni parametri che regolano l'acquisizione dell'immagine, in base alle condizioni di luminosità esterne. Ad esempio, durante la ripresa di scene notturne la camera solitamente aumenta il *tempo di esposizione* del sensore, in modo che esso possa ricevere più luce possibile. Ciò porta alla *presenza di sfocature* nel caso vengano immortalati degli oggetti in movimento (come ad esempio le macchine che si muovono nella Figura 4.2(b)). Nel caso in cui il tempo di esposizione non fosse sufficiente, invece, i cambiamenti di luminosità potrebbero portare a un aumento del rumore nei frame acquisiti nelle ore più scure, in quanto il sensore riceverebbe meno luce.

Questi fenomeni fanno sì che i nostri indicatori abbiano una dinamica *dificilmente prevedibile* e che *non siano stazionari*², come possiamo vedere dal comportamento dell'energia del gradiente nella Figura 4.3(b) e da quello dell'energia media della luma nella Figura 4.4(b). I due grafici riportano l'andamento degli indicatori durante una ripresa di 24 ore, acquisendo un frame ogni minuto, in cui non sono avvenuti eventi di tampering. Il fatto

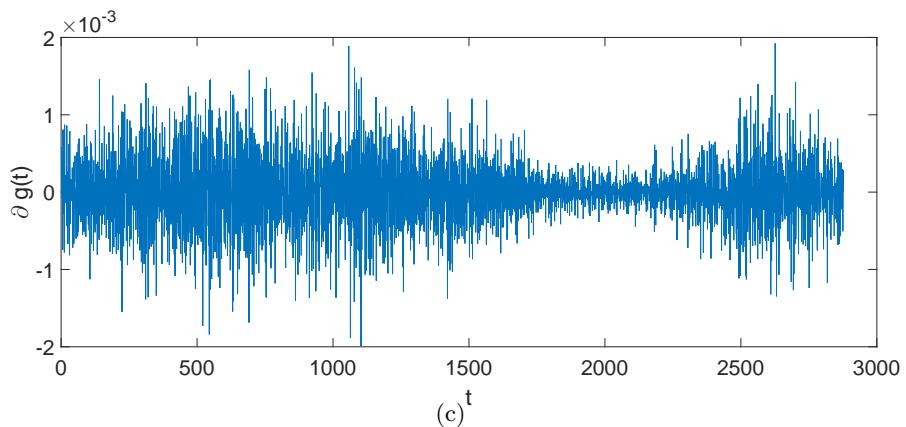
²Ovvero non sono realizzazioni i.i.d. di una stessa variabile aleatoria



(a)



(b)

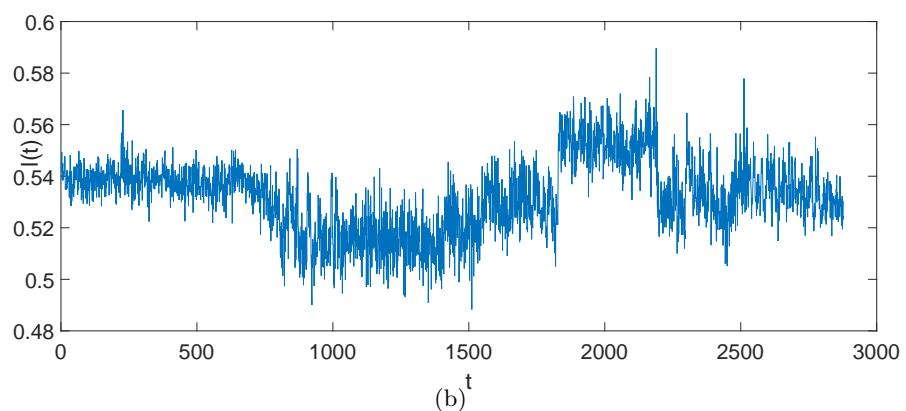


(c)

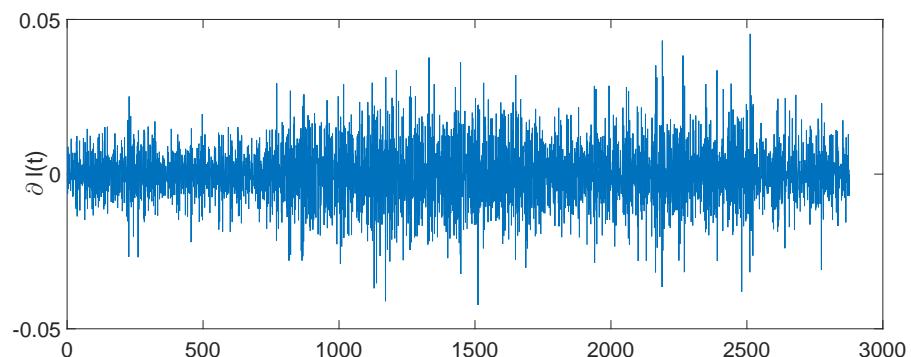
Figura 4.3: Energia media del gradiente lungo un'acquisizione di 24 ore (a) e suo detrending (b)



(a)



(b)



(c)

Figura 4.4: Energia media della luma lungo un'acquisizione di 24 ore (a) e suo detrending (b)

che questi indicatori non si comportino in maniera stazionare anche nel caso in cui non avvengano eventi di tampering rende impossibile applicare direttamente le tecniche di CDT come in [31].

4.1.4 Detrending degli indicatori

Osservando l’andamento degli indicatori nelle Figure 4.3(b) e 4.4(b) possiamo notare la presenza di componenti ad alta frequenza, dovute dal rumore nelle immagini, e di componenti in bassa frequenza, dovute prevalentemente ai cambi di luce e dalla dinamicità della scena. Per eliminare le componenti in bassa frequenza, che sono quelle che rendono i nostri indicatori non prevedibili, possiamo fare un *detrending* di ciascun segnale calcolando la *differenza* tra l’indicatore all’istante corrente e quello precedente. Nel caso dell’energia media del gradiente avremo

$$\frac{\partial g}{\partial t}(t) = g(t) - g(t-1), \quad (4.5)$$

mentre per l’energia media della luma avremo

$$\frac{\partial l}{\partial t}(t) = l(t) - l(t-1). \quad (4.6)$$

Vediamo un esempio di come si comporta il detrending sui nostri indicatori nelle Figure 4.3(c) e 4.4(c). Possiamo notare come le fluttuazioni in bassa frequenza vengano rimosse dal detrending, visto che gli indicatori di due frame consecutivi sono pressoché costanti.

Come abbiamo detto prima, lo scopo del monitoraggio di questi indicatori è quello di individuare degli eventi di tampering. Nelle Figure 4.5 e 4.7 possiamo vedere il comportamento dell’energia media di luma e gradiente rispettivamente per un caso di sfocatura e per un caso di spostamento della camera. In particolare, in entrambi i casi l’evento di tampering avviene al frame 1000³. Il detrending di questi segnali è visualizzato nelle Figure 4.6 e 4.8.

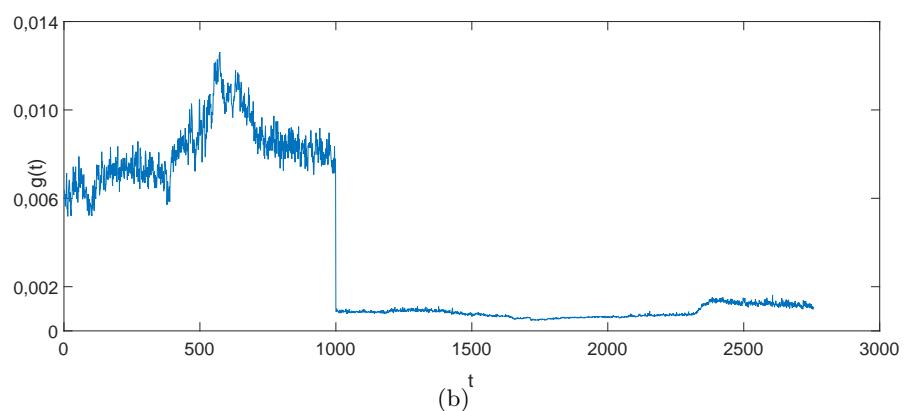
Possiamo vedere come, in generale, l’evento di tampering sia associato a un brusco salto o a un crollo *istantaneo* del valore di uno o entrambi gli indicatori. Vanno notate alcune cose:

- L’evento di sfocatura non si traduce in un cambiamento nell’energia media della luma. Questo avviene perché questo tipo di tampering abbatte le alte frquenze presenti nell’immagine, mantenendo comunque il valore medio della luma invariato.

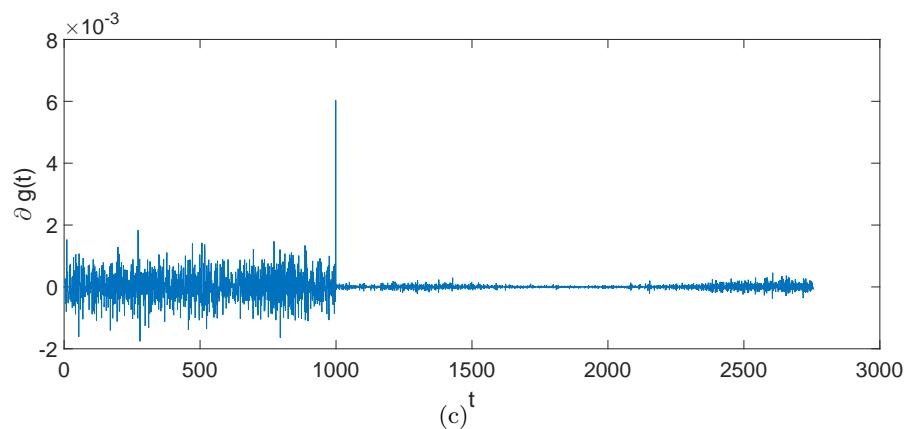
³Il modo in cui sono stati ottenuti gli eventi di tampering è descritto nel Capitolo 5



(a)



(b)

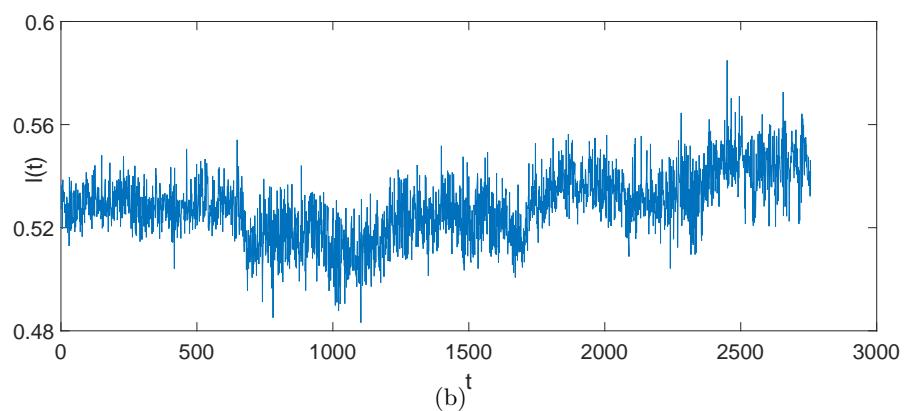


(c)

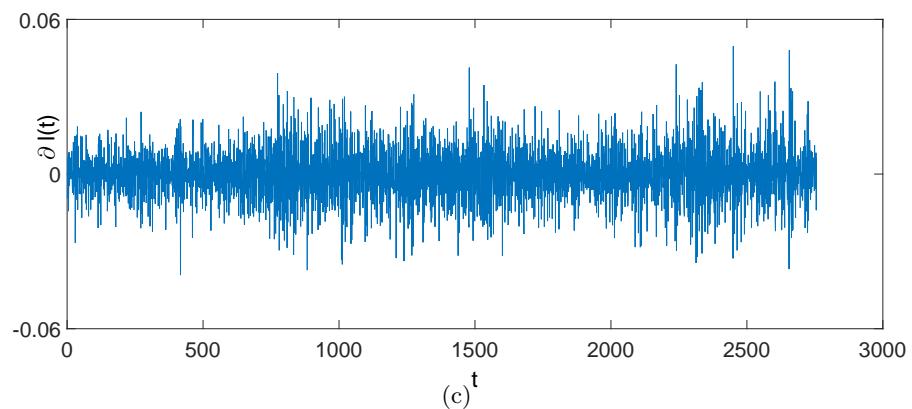
Figura 4.5: Energia media del gradiente (a) e suo detrending (b) nel caso di una sfocatura



(a)



(b)

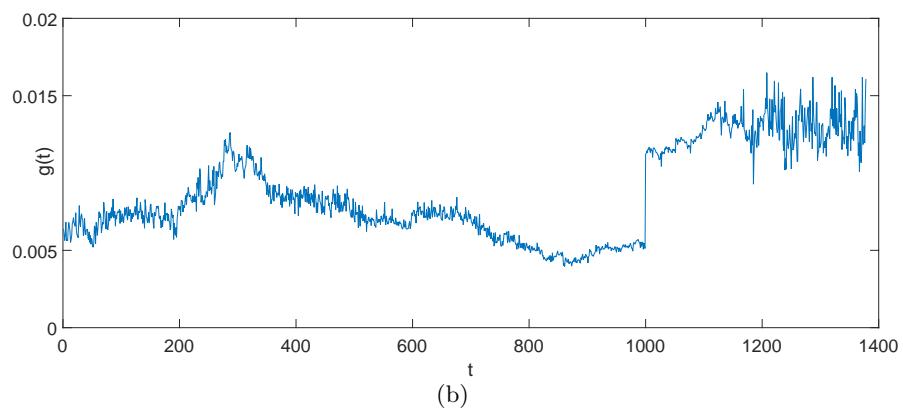


(c)

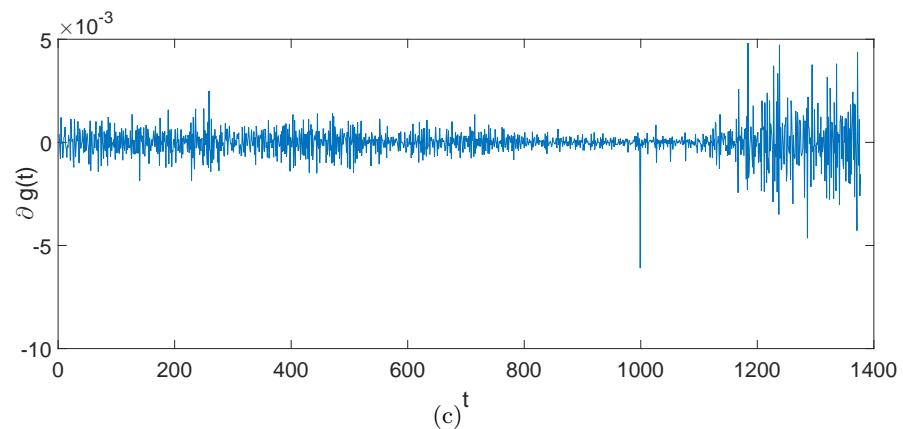
Figura 4.6: Energia media della luma (a) e suo detrending (b) nel caso di una sfocatura



(a)



(b)

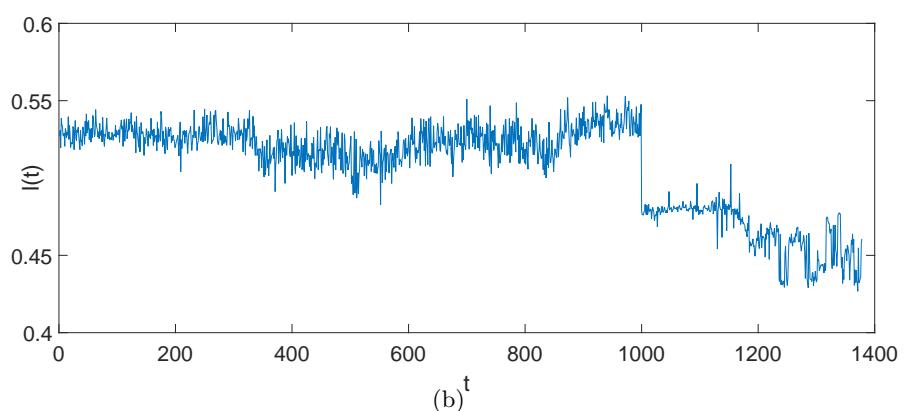


(c)

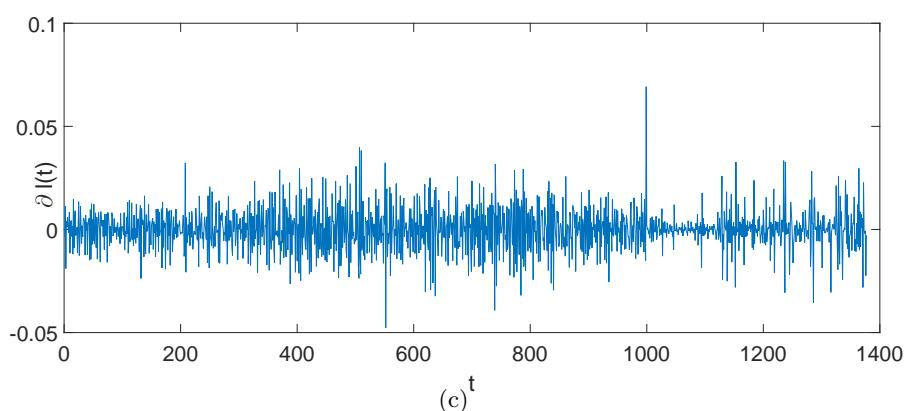
Figura 4.7: Energia media del gradiente (a) e suo detrending (b) nel caso di uno spostamento della camera



(a)



(b)



(c)

Figura 4.8: Energia media della luma (a) e suo detrending (b) nel caso di uno spostamento della camera

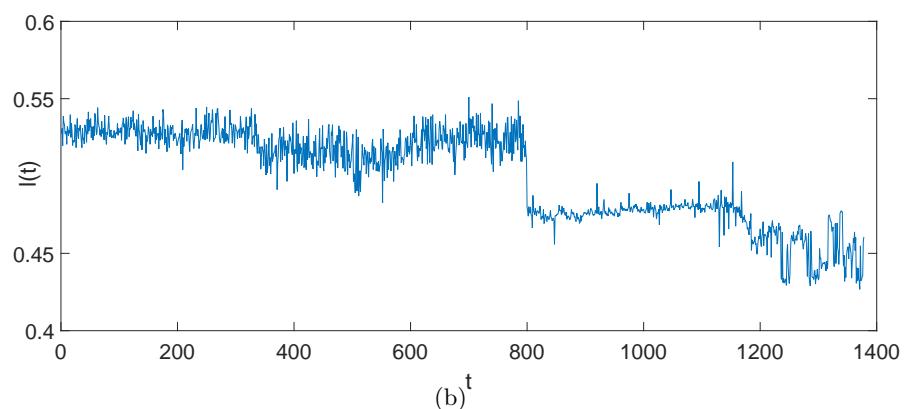
- Facendo il detrending della sequenza abbiamo dei valori che oscillano attorno allo zero, un picco in corrispondenza dell'istante in cui avviene il tampering (Figure 4.5(c), 4.7(c), 4.8(c)) e infine altri valori che oscillano attorno allo zero. Questo è dato dal fatto che il detrending considera le differenze tra dati consecutivi e, quindi, la differenza maggiore si ha proprio nell'istante in cui inizia l'evento di tampering, mentre solitamente le differenze tra misure consecutive sono minime. Questo fa sì che monitorare il detrending delle sequenze degli indicatori permetta di identificare più facilmente gli eventi di tampering rispetto ad analizzare la sequenza originale. Il risvolto della medaglia è che i cambiamenti *persistenti*, come quelli che stiamo considerando noi, diventano dei cambiamenti *istantanei*.
- Considerando l'evento di sfocatura, possiamo notare (Figura 4.5(b)) come, in seguito all'evento, oltre ad aver un crollo istantaneo del valore dell'energia del gradiente, abbiamo anche un abbassamento *persistente* della sua varianza. Questo permette di utilizzare tecniche di monitoraggio sequenziale sull'energia del gradiente, con dei CDT sulla varianza, per identificare eventi di sfocatura.

In definitiva, è possibile identificare un evento di tampering andando a monitorare il detrending degli indicatori descritti da (4.1) e (4.4), utilizzando semplicemente delle soglie in modo da individuare il picco causato dall'evento di tampering. In particolare possiamo monitorare l'energia media del gradiente per individuare eventi di sfocatura, e l'energia media della luma per individuare eventi di spostamento della camera. Inoltre, per rendere più robusta l'identificazione di sfocature, è possibile usare un test sequenziale sull'energia media del gradiente in grado di individuare dei cambiamenti nella varianza. Dato il *basso carico computazionale* richiesto da queste tecniche, è possibile integrarle su dispositivi embedded a basso consumo di energia. Dobbiamo fare un'ultima considerazione sullo spostamento della camera. Ci possono essere dei casi in cui l'evento è difficilmente individuabile monitorando la scena nella sua totalità. Un esempio è illustrato nella Figura 4.9, dove vediamo un caso di spostamento della camera che avviene al frame 800. Questo evento, però, non si traduce, nel segnale $\frac{\partial l}{\partial t}$, in un picco che si eleva rispetto agli altri, come nel caso della Figura 4.8(c). Questo problema capita perché stiamo monitorando l'energia media della luma calcolata sulla *totalità* della scena. Possiamo avere situazioni, come nel caso della Figura 4.9, in cui lo spostamento della camera non determina un cambiamento sostanziale nella luminosità media della scena.

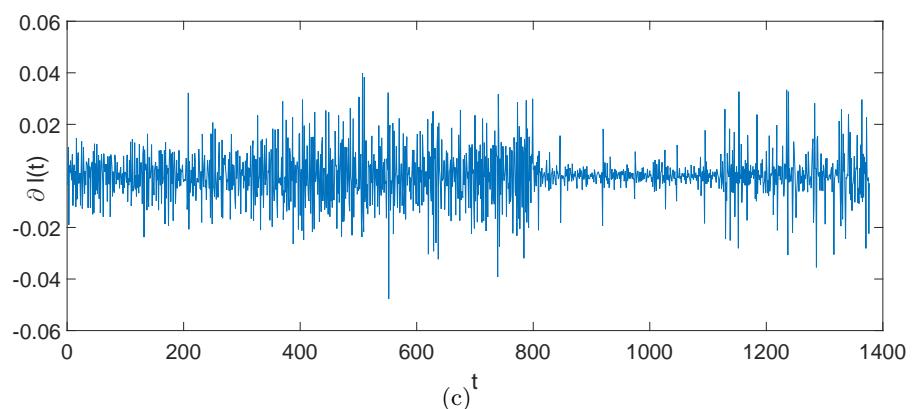
Questo problema viene meno se consideriamo il contributo dell'energia della luma mediato non sulla totalità della scena, bensì *separatamente* su *regioni* specifiche. Infatti, se consideriamo un'area particolare della scena, come ad esempio quella occupata dal palazzo nella Figura 4.10(a), durante uno spostamento della camera la variazione della luma mediata solo sui pixel



(a)



(b)



(c)

Figura 4.9: Esempio di sequenza dell'energia media della luma (a) e del suo detrending (b) con un displacement difficile da identificare



Figura 4.10: Esempio di spostamento della camera

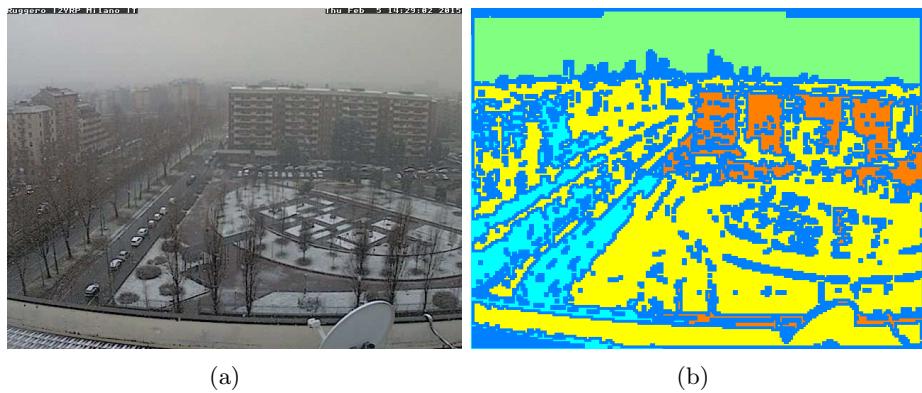


Figura 4.11: Esempio di segmentazione della scena

appartenenti a quella regione sarà più marcata rispetto a quella della luminosità mediata su tutta l'immagine. Questo ci ha suggerito di inserire una fase di *segmentazione* in cui vengono estratte le regioni della scena che la camera deve inquadrare. Un esempio è illustrato nella Figura 4.11. Nel Paragrafo 4.3 vedremo in dettaglio come vengono estratte le regioni dalla scena.

4.2 Algoritmo di identificazione di sfocature e di spostamenti della camera

In base alle considerazioni fatte nel Paragrafo 4.1 possiamo dividere l'algoritmo di tampering detection in due *thread*: il primo in grado di individuare la presenza di sfocature, il secondo lo spostamento della camera. In particolare la presenza di sfocature può essere identificata monitorando l'energia media del gradiente, descritta in (4.1), mentre l'evento di spostamento della

```

Configurazione:
1. for  $t = 1, \dots, T_o$  do
2.   Estraggo il frame  $z_t$ 
3.   Calcolo  $g(t)$ ,  $\frac{\partial g}{\partial t}(t)$ 
end
4. Definisco le soglie  $\Gamma_{min}^g$  e  $\Gamma_{max}^g$ 
5. Definisco i parametri per il CDT sulla varianza di  $g(t)$ 

Fase operativa:
6. for  $t = T_o, \dots, \infty$  do
7.   Estraggo il frame  $z_t$ 
8.   Calcolo  $g(t)$ ,  $\frac{\partial g}{\partial t}(t)$ 
9.   if  $\frac{\partial g}{\partial t}(t) < \Gamma_{min}^g \vee \frac{\partial g}{\partial t}(t) > \Gamma_{max}^g$  then
10.    |  $z_t$  è un frame in cui è avvenuto una sfocatura
end
11.  if CDT identifica un cambiamento in  $g(t)$  then
12.    |  $z_t$  è un frame in cui è avvenuto uno spostamento della camera
end
end

```

Algoritmo 1: Algoritmo di identificazione di sfocature

camera può essere identificato monitorando l'energia media della luma, descritta in (4.4).

In questo paragrafo illustriamo come sono state sviluppate le due tecniche.

4.2.1 Identificazione delle sfocature

L'Algoritmo 1 mostra il funzionamento, ad alto livello, dell'algoritmo per identificare la presenza di sfocature nei frame. Dato che la sfocatura causa un crollo dell'energia media del gradiente e anche della sua varianza, è possibile lanciare due monitoraggi in grado di identificare ciascuno di questi cambiamenti, uno di tipo *one-shot* e l'altro di tipo *sequenziale*.

Il monitoraggio one-shot prevede un'analisi del detrending dell'energia media del gradiente, secondo (4.1) e (4.5), in modo da identificare il picco dovuto al cambiamento (si veda la Figura 4.5). L'identificazione viene fatta attraverso la definizione di due *soglie*, calcolate facendo riferimento alle prime T_o osservazioni, ritenute prive di tampering. Tale sequenza prende il nome di *training set*. Le due soglie Γ_{min}^g e Γ_{max}^g vengono calcolate nel seguente modo:

$$\begin{aligned}\Gamma_{min}^g &= \hat{\mu}_g(t) - \gamma \hat{\sigma}_g(t) \\ \Gamma_{max}^g &= \hat{\mu}_g(t) + \gamma \hat{\sigma}_g(t)\end{aligned}, \quad (4.7)$$

dove $\hat{\mu}_g(t)$ indica il valore medio delle osservazioni del training set

$$\hat{\mu}_g(t) = \frac{\sum_{\tau=1}^{T_o} \frac{\partial g}{\partial t}(\tau)}{T_o},$$

$\hat{\sigma}_g(t)$ indica la deviazione standard delle osservazioni del training set

$$\hat{\sigma}_g(t) = \sqrt{\frac{1}{T_o - 1} \sum_{\tau=1}^{T_o} \left(\frac{\partial g}{\partial t}(\tau) - \hat{\mu}_g(\tau) \right)^2}$$

e $\gamma > 1$ è un parametro moltiplicativo ottenuto sperimentalmente⁴.

Le soglie definite in (4.7) forniscono un limite superiore e uno inferiore ai valori di $\{\frac{\partial g}{\partial t}\}$ ammissibili. Qualsiasi valore al di sotto di Γ_{min} o al di sopra di Γ_{max} viene considerato come un evento di sfocatura nel frame corrispondente.

Oltre all'analisi sul detrending è possibile fare un monitoraggio sequenziale sulla varianza di $\{g(t)\}$: infatti, come abbiamo visto nel Paragrafo 4.1.3, la presenza di una sfocatura comporta una diminuzione della varianza di questa sequenza. Il monitoraggio sequenziale viene fatto utilizzando un CDT basato su ICI-rule⁵ sulla varianza di $\{g(t)\}$, usando il training set per configurare i suoi parametri.

4.2.2 Identificazione degli spostamenti della camera

L'Algoritmo 2 mostra il funzionamento, ad alto livello, dell'algoritmo per identificare un evento di spostamento della camera.

Una prima fase dell'algoritmo consiste nella *segmentazione* della scena inquadrata dalla camera in un insieme di regioni $\{R_k\}$, $k = 1, \dots, K$. Il metodo con cui vengono estratte queste regioni verrà analizzato in dettaglio nel paragrafo 4.3.

L'analisi che viene fatta è simile a quella vista nel monitoraggio del detrending dell'energia del gradiente: in questo caso l'analisi è fatta in maniera indipendente per ciascuna regione R_k , calcolando l'energia della luma *mediata per i pixel appartenenti alla regione*:

$$\begin{aligned} l^k(t) &= \mathcal{L}^k[z_t] = \frac{\sum_{x \in R_k} z_t(x)}{|R_k|}, \\ \frac{\partial l^k}{\partial t}(t) &= l_t^k - l_{t-1}^k, \end{aligned} \quad (4.8)$$

dove abbiamo indicato con $|R_k|$ il numero totale dei pixel appartenenti alla regione R_k .

⁴Per maggiori dettagli si rimanda al Capitolo 5

⁵Lo schema di funzionamento del CDT basato su ICI-rule è spiegato nel Paragrafo 2.3.

Configurazione:

1. Estrago le regioni $\{R_k\}, k = 1, \dots, K$
2. **for** $t = 1, \dots, T_o$ **do**
3. Estrago il frame z_t
4. **for** $k = 1, \dots, K$ **do**
5. Calcolo $l^k(t), \frac{\partial l^k}{\partial t}(t)$ per la regione R_k
6. **end**
7. **end**
8. **for** $k = 1, \dots, K$ **do**
9. Definisco le soglie Γ_{min}^k e Γ_{max}^k
10. **end**

Fase operativa:

11. **for** $t = T_o, \dots, \infty$ **do**
12. Estrago il frame z_t
13. $n = 0$
14. **for** $k = 1, \dots, K$ **do**
15. Calcolo $l^k(t), \frac{\partial l^k}{\partial t}(t)$ per la regione R_k
16. **if** $\frac{\partial l^k}{\partial t}(t) < \Gamma_{min}^k \vee \frac{\partial l^k}{\partial t}(t) > \Gamma_{max}^k$ **then**
17. $n = n + 1$
18. **end**
19. **end**
20. **if** $n \geq K - 1$ **then**
21. z_t è un frame in cui è avvenuto uno spostamento della camera
22. **end**
23. **end**

Algoritmo 2: Algoritmo di identificazione di spostamenti della camera

L'identificazione viene fatta attraverso la definizione di due *soglie*, calcolate facendo riferimento alle prime T_o osservazioni, ritenute prive di tampering. Le due soglie Γ_{min}^k e Γ_{max}^k vengono calcolate per ciascuna regione nel seguente modo:

$$\begin{aligned}\Gamma_{min}^k &= \hat{\mu}_l^k(t) - \gamma \hat{\sigma}_l^k(t) \\ \Gamma_{max}^k &= \hat{\mu}_l^k(t) + \gamma \hat{\sigma}_l^k(t)\end{aligned}, \quad (4.9)$$

dove $\hat{\mu}_l^k(t)$ indica il valore medio delle osservazioni del training set

$$\hat{\mu}_l^k(t) = \frac{\sum_{\tau=1}^{T_o} \frac{\partial l^k}{\partial t}(\tau)}{T_o},$$

$\hat{\sigma}_l^k(t)$ indica la deviazione standard delle osservazioni del training set

$$\hat{\sigma}_l^k(t) = \sqrt{\frac{1}{T_o - 1} \sum_{\tau=1}^{T_o} \left(\frac{\partial l^k}{\partial t}(\tau) - \hat{\mu}_l^k(\tau) \right)^2}$$

e $\gamma > 1$ è un parametro moltiplicativo ottenuto sperimentalmente.

Le soglie definite nella Formula (4.9) forniscono un limite superiore e uno inferiore ai valori di $\{\frac{\partial l^k}{\partial t}\}$ ammissibili.

Dato che l'evento di spostamento della camera viene inteso come un cambiamento globale dell'immagine, ci aspettiamo che tale cambiamento sia percepibile in tutte le regioni contemporaneamente. Nella pratica, però, può capitare che l'energia media della luma all'interno di una specifica regione non cambi, anche in caso di spostamento della camera. Se consideriamo, ad esempio, l'evento di spostamento nella Figura 4.9 e la segmentazione della scena in Figura 4.11(b), infatti, possiamo notare come la luma nella regione del cielo non subisca cambiamenti sostanziali dato che, anche in seguito all'evento di tampering, questa regione riprende sempre una porzione di cielo. In generale, quindi, possiamo considerare un monitoraggio indipendente di ogni sequenza $\{\Delta l_t^k\}$ per $k = 1, \dots, K$. Se, all'istante T^* , abbiamo che, per $K - 1$ regioni sulle K totali, il valore di $\Delta l_{T^*}^k$ è maggiore di Γ_{max}^k o minore di Γ_{min}^k , allora possiamo considerare avvenuto un evento di sfocatura nel frame z_{T^*} .

Dato che l'evento di spostamento della camera, in genere, non comporta un cambiamento persistente nella varianza dell'energia media della luma, non ha senso aggiungere un monitoraggio sequenziale come abbiamo fatto nell'Algoritmo 1.

4.3 Algoritmo di segmentazione

In molte applicazioni di visione artificiale è spesso necessario estrarre, da un'immagine o un video, una o più regioni specifiche. Quando si vanno a trattare dei video, i tipi di segmentazione che possono essere applicati sono solitamente due:

- una segmentazione di tipo *spaziale*, in cui vengono estratti dei pixel specifici per ogni frame costituente il video;
- una segmentazione di tipo *temporale*, in cui vengono estratti dei frame significativi (*key-frames*).

Segmentazioni di tipo spaziale permettono, ad esempio, di seguire il movimento di uno specifico oggetto all'interno della scena ripresa [33, 34], oppure di identificare un particolare evento all'interno di una scena [35]. Alcuni standard video, come ad esempio l'MPEG-4, utilizzano una codifica *object-based* in cui viene utilizzata una segmentazione spaziale dei frame [36]. Segmentazioni di tipo temporale, invece, permettono di dividere un filmato in *shot* oppure di creare una sintesi del video con dei frame significativi. In quest'ultimo caso si parla di *key-frames extraction* o di *video summarization* [37, 38, 39, 40]. Quando si vanno a trattare immagini la segmentazione è di

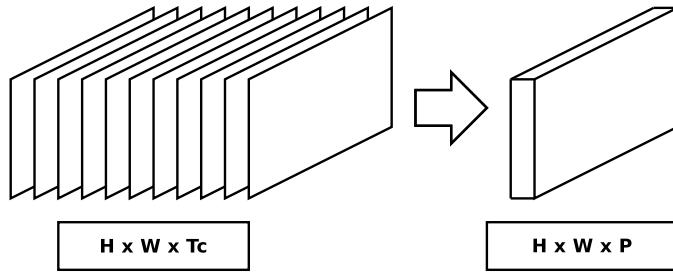


Figura 4.12: Passaggio dallo spazio dei frame allo spazio dei feature vector

tipo spaziale.

Considerando la segmentazione di tipo spaziale, nella maggior parte dei casi la tecnica consiste nell'utilizzare dei descrittori per ciascun pixel e applicare delle tecniche di *clustering*, ovvero delle metodologie atte a partizionare un insieme di dati [41], sugli stessi.

Nel Paragrafo 4.1.3 abbiamo introdotto il fatto di utilizzare una segmentazione in regioni della scena ripresa dalla camera, in modo da rendere più efficiente l'identificazione di spostamenti della camera. Vediamo ora il modo in cui questa segmentazione viene fatta.

Durante la fase di estrazione delle regioni acquisiamo un certo numero di frame in cui non avviene nessun evento di tampering. Da questi frame estraiamo un vettore descrittore (*feature vector*) per ciascun pixel della scena inquadrata. Passiamo, quindi, da una matrice di dimensioni $H \times W \times T_c$, dove H e W sono le dimensioni in pixel dell'immagine acquisita dalla camera e T_c è il numero di frame usato per la creazione della segmentazione, a una matrice di dimensioni $H \times W \times P$, dove P è il numero di descrittori calcolati per fare la segmentazione.

Una volta ottenuta la matrice, viene fatto un *partizionamento* dei vari feature vector, in modo da raggruppare i pixel in base a parametri comuni.

La fase di segmentazione della scena in regioni avviene durante la configurazione del sistema, separatamente rispetto alla fase di identificazione degli eventi di tampering. La segmentazione, quindi, non comporta ulteriore costo computazionale all'algoritmo, che la considera semplicemente come una variabile d'ingresso. Inoltre, la segmentazione può essere eseguita *offline* su un dispositivo esterno.

4.3.1 Calcolo dei descrittori utilizzati per la segmentazione

Consideriamo la sequenza $\{z_t\}$ di frame acquisiti dalla camera, con $t = 1, \dots, T_c$. Per ciascun pixel $x \in \mathcal{X}$ calcoliamo un vettore $\mathbf{d}(x)$ di 5 elementi

$$\mathbf{d}(x) = [r(x); c(x); \mu_\nabla(x); \sigma_\nabla(x); \bar{z}(x)], \mathbf{d}(x) \in \mathbb{R}^5 \quad (4.10)$$

dove:

- $r(x)$ rappresenta il numero di riga del pixel x .
- $c(x)$ rappresenta il numero di colonna del pixel x .
- $\mu_\nabla(x)$ rappresenta il valore del gradiente nel pixel x mediato nel tempo:

$$\mu_\nabla(x) = \frac{\sum_{t=1}^{T_c} (\|\nabla z_t\|_2^2 \circledast f)(x)}{T_c}, \quad (4.11)$$

dove abbiamo indicato con $\|\nabla z_t\|_2^2$ la norma del gradiente per l'immagine z_t , definita in (4.3), e con f il filtro gaussiano discreto derivato dal campionamento di (4.2).

- $\sigma_\nabla(x)$ rappresenta la deviazione standard nel tempo del gradiente nel pixel x :

$$\sigma_\nabla(x) = \sqrt{\frac{1}{T_c - 1} \sum_{t=1}^{T_c} ((\|\nabla z_t\|_2^2 \circledast f)(x) - \mu_\nabla(x))^2}. \quad (4.12)$$

- $\bar{z}(x)$ rappresenta il valore della luma del pixel x mediato nel tempo:

$$\bar{z}(x) = \frac{\sum_{t=1}^{T_c} (z_t \circledast f)(x)}{T_c}. \quad (4.13)$$

Come possiamo vedere in (4.11), (4.12) e (4.13), nel calcolo degli indicatori viene fatta una convoluzione con un filtro gaussiano. Questo è equivalente a mediare il contributo di $\|\nabla z_t\|_2^2$ e di z_t in un *intorno* nello spazio delle coordinate.

Nella Figura 4.13 possiamo vedere un esempio di come si comportano questi indicatori su una specifica scena (Figura 4.13(a)). Il loro utilizzo permette di avere un'informazione completa sul comportamento dei pixel durante la ripresa della scena e il loro rapporto con i pixel appartenenti al suo intorno, definito dal filtro gaussiano f .

In particolare, l'indicatore $\mu_\nabla(x)$ mette in relazione il valore di luma del pixel x in rapporto con quelli nel suo intorno, definito dal filtro f . Valori alti di $\mu_\nabla(x)$ corrispondono a zone della scena con molti dettagli (*textured*), mentre valori bassi corrispondono a zone della scena poco dettagliate (*flat*). Ad esempio, nella Figura 4.13(b) possiamo vedere come i valori più alti di $\mu_\nabla(x)$ corrispondano alle zone dove le variazioni di luma sono più elevate (ad esempio nelle aree di contorno tra le foglie degli alberi e il cielo).

L'indicatore $\sigma_\nabla(x)$, invece, ci fornisce un'informazione riguardo all'evoluzione di ciascun pixel nel tempo. Valori alti di $\sigma_\nabla(x)$ corrisponderanno a pixel appartenenti a zone *dinamiche* della scena, mentre valori bassi dell'indicatore corrisponderanno a pixel appartenenti a zone *statiche*. Nell'esempio in

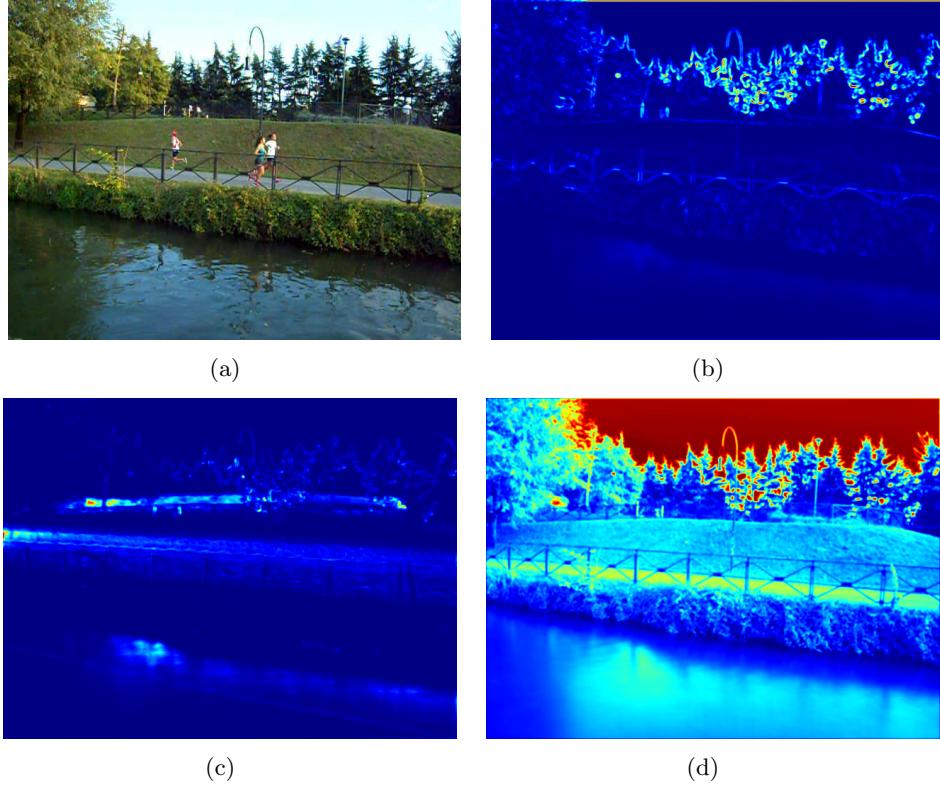


Figura 4.13: Esempio grafico di come si comportano i descrittori utilizzati per la segmentazione di una specifica scena. La scena di esempio è rappresentata in (a). In (b), dove sono illustrati il comportamento di $\mu_{\nabla}(x)$ per ciascun pixel, possiamo vedere come i valori più alti siamo in prossimità delle zone più dettagliate, come per esempio il contorno tra le foglie e il cielo. In (c), invece, vediamo i valori di $\sigma_{\nabla}(x)$ per ciascun pixel. Possiamo notare come i valori più alti del descrittore siano in prossimità delle zone a maggior dinamicità. In (d), infine, mostriamo i valori di $\bar{z}(x)$ per ciascun pixel.

Figura 4.13(c) possiamo vedere come i valori più alti dell’indicatore corrispondano alle zone dove passano i pedoni e i ciclisti.

L’indicatore \bar{z} (Figura 4.13(d)) ci fornisce un’indicazione su come è distribuita mediamente la luminosità nella scena.

L’utilizzo delle coordinate spaziali, infine, permette di avere un’informazione sulla posizione del pixel all’interno della scena.

4.3.2 Partizionamento dei feature vector per l’estrazione delle regioni

Una volta calcolati i feature vector di tutti i pixel, possiamo raggrupparli tra loro usando un algoritmo di *clustering*. In realtà, per alleggerire il peso computazionale nella fase di segmentazione, possiamo fare un *campionamento*

dei feature vector totali. Come abbiamo visto nel Paragrafo 4.3.1, infatti, ciascun feature vector è associato non al singolo pixel, ma a un intorno del pixel definito dal filtro gaussiano f . Possiamo, quindi, dividere la scena totale in *blocchi*, le cui dimensioni sono quelle del filtro f , e prendere il feature vector del pixel centrale di ciascun blocco.

Per partizionare i feature vector selezionati, abbiamo deciso di utilizzare il classico algoritmo di clustering *k-means* [41, 42].

K-means permette di partizionare un insieme di dati in K sottoinsiemi. L'algoritmo, nella sua forma più semplice, riceve in ingresso un insieme $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ di N dati da partizionare e il numero K di partizioni (cluster) $\{C_i\}, i = 1, \dots, K$ che si vuole creare. Per semplicità consideriamo ciascun dato come un vettore $\mathbf{d}(\mathbf{x})$ di P elementi, $\mathbf{d}(\mathbf{x}) \in \mathbb{R}^P, \mathbf{x} \in \mathcal{X}$.

Come prima cosa vengono creati i *centroidi* $\{\mathbf{c}_i\}, i = 1, \dots, K, \mathbf{c}_i \in \mathbb{R}^P$ associati ai cluster $\{C_i\}, i = 1, \dots, K$. Nella versione più semplice dell'algoritmo essi vengono scelti in maniera casuale dai dati appartenenti a \mathcal{X} . Una tecnica più raffinata è data dall'algoritmo *k-means++* [43], che utilizza funzioni probabilistiche più accurate e permette di distribuire meglio i centroidi su tutto lo spazio dei dati. In particolare, k-means++ sceglie il primo centroide \mathbf{c}_1 in maniera casuale da \mathcal{X} con distribuzione probabilistica uniforme. I successivi centroidi vengono scelti iterativamente prendendo $\mathbf{c}_i = \mathbf{x}' \in \mathcal{X}$ con probabilità $\frac{D(\mathbf{x}')^2}{\sum_{\mathbf{x} \in \mathcal{X}} D(\mathbf{x})^2}$, dove $D(\mathbf{x})$ indica la minima distanza di \mathbf{x} dal centroide più vicino tra quelli già calcolati.

Una volta ottenuti i centroidi vengono calcolate le *distanze euclidi* di ciascun dato da ogni centroide:

$$\text{dist}(\mathbf{d}(\mathbf{x}), \mathbf{c}_i) = (\mathbf{d}(\mathbf{x}) - \mathbf{c}_i)^T \cdot (\mathbf{d}(\mathbf{x}) - \mathbf{c}_i).$$

Ciascun dato $\mathbf{d}(\mathbf{x})$, quindi, viene associato al cluster con centroide $\mathbf{c}^*(\mathbf{x})$ con distanza *minima* da esso:

$$\mathbf{c}^*(\mathbf{x}) = \underset{\mathbf{c}_i}{\operatorname{argmin}} \text{dist}(\mathbf{d}(\mathbf{x}), \mathbf{c}_i).$$

Una volta associato ciascun dato al cluster più vicino, vengono aggiornati i centroidi di ciascun cluster in base ai loro elementi:

$$\mathbf{c}_i = \frac{\sum_{\mathbf{x} \in C_i} \mathbf{x}}{|C_i|}$$

La procedura viene ripetuta con la nuova assegnazione dei centroidi. L'algoritmo viene richiamato fino a quando non si raggiunge la *convergenza*, ovvero il momento in cui la posizione dei centroidi non cambia più.

Per utilizzare questa tecnica nel nostro algoritmo di segmentazione abbiamo dovuto fare alcune modifiche alla sua versione originale, in modo da risolvere alcuni problemi.

Il primo problema, utilizzando la versione base del k-means, è quello di determinare il numero di cluster da creare. Per valutare in maniera automatica

il numero di cluster ottimale per ciascun insieme di dati abbiamo utilizzato il *criterio di Calinski-Harabasz*[44], che consiste nel provare l'algoritmo di k-means per un insieme di valori $k = K_{min}, \dots, K_{max}$. Per ciascun k utilizzato viene calcolata una cifra di merito chiamata *variance ratio criterion* (VRC):

$$\text{VRC}_k = \frac{SS_B}{SS_W} \times \frac{N - k}{k - 1},$$

dove SS_B rappresenta la *varianza totale tra cluster* (*overall between-cluster variance*):

$$SS_B = \sum_{i=1}^k |C_i| \| \mathbf{c}_i - \bar{\mathbf{d}} \|^2,$$

con $|C_i|$ che rappresenta il numero di dati appartenenti al cluster C_i e $\bar{\mathbf{d}}$ che rappresenta il valore medio di tutti i dati:

$$\bar{\mathbf{d}} = \frac{\sum_{\mathbf{x} \in \mathcal{X}} \mathbf{x}}{|\mathcal{X}|}.$$

SS_W , invece, rappresenta il grado di dispersione medio di ciascun cluster (*overall within-cluster variance*):

$$SS_W = \sum_{i=1}^k \sum_{\mathbf{x} \in C_i} \| \mathbf{d}(\mathbf{x}) - \mathbf{c}_i \|^2.$$

Viene scelto, infine, il valore k con il valore VRC_k più alto:

$$K^* = \operatorname{argmax}_k \text{VRC}_k.$$

Un altro problema è dato dal peso che ha ciascun elemento dei dati durante la segmentazione. Nel nostro caso, ad esempio, i valori delle coordinate possiedono una varianza talmente elevata rispetto a quella degli altri elementi, che il loro contributo domina quello delle altre componenti, come possiamo vedere nell'esempio in Figura 4.14(b). Per risolvere il problema abbiamo utilizzato, quindi una variante *pesata* del k-means [34], in cui il calcolo delle distanze di ciascun dato da ogni centroide viene modificato associando un *peso* a ciascuna componente:

$$\text{dist}_w(\mathbf{d}(\mathbf{x}), \mathbf{c}_i) = (\mathbf{d}(\mathbf{x}) - \mathbf{c}_i)^T W^i (\mathbf{d}(\mathbf{x}) - \mathbf{c}_i),$$

dove W^i è una matrice di pesi associata a ciascun cluster C_i :

$$W^i = \begin{bmatrix} w_1^i & 0 & 0 & 0 & 0 \\ 0 & w_2^i & 0 & 0 & 0 \\ 0 & 0 & w_3^i & 0 & 0 \\ 0 & 0 & 0 & w_4^i & 0 \\ 0 & 0 & 0 & 0 & w_5^i \end{bmatrix},$$

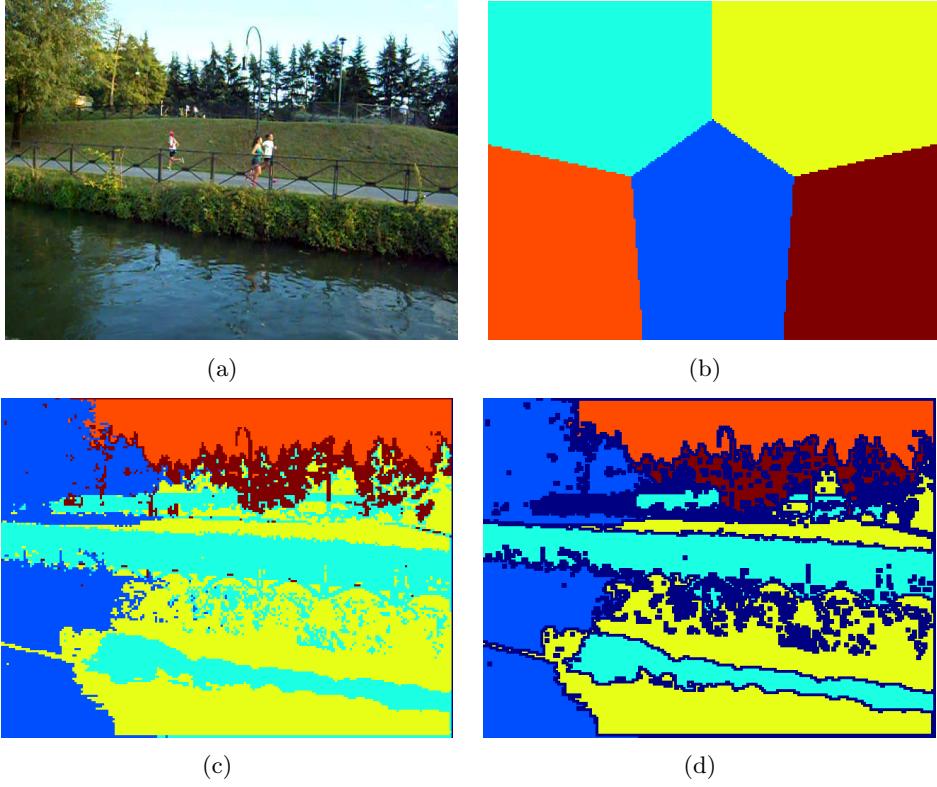


Figura 4.14: Un esempio di segmentazione di una particolare scena. La scena di esempio è rappresentata in (a). Possiamo vedere in (b) il risultato della segmentazione utilizzando l'algoritmo di k-means nella sua versione originale, dove l'unico contributo è dato dalle coordinate dei pixel. Il risultato ottenuto utilizzando l'algoritmo di k-means pesato è illustrato invece in (c), dove possiamo vedere come la segmentazione sia legata a particolari regioni della scena. Infine, in (d) possiamo vedere il risultato finale della segmentazione, che consiste nella separazione delle regioni e nell'eliminazione delle aree più piccole.

dove w_s^i corrisponde al peso dato all'elemento s del feature vector associato al cluster C_i :

$$w_s^i = \frac{(\sigma_1^i \cdot \sigma_2^i \cdot \sigma_3^i \cdot \sigma_4^i \cdot \sigma_5^i)^{1/5}}{\sigma_s^i}, s = 1, \dots, 5,$$

dove σ_s^i rappresenta la deviazione standard dell'elemento s dei dati appartenenti al cluster C_i .

4.3.3 Separazione delle regioni

L'algoritmo di clustering utilizzato produce un partizionamento della scena ripresa dalla camera come quello in Figura 4.14(c). Come possiamo vedere

le regioni estratte dal nostro algoritmo sono legate a particolari aree presenti nella scena, in base alla loro dinamicità, alla presenza di dettagli e all'intensità media della luma.

Una fase finale della segmentazione consiste nell'eliminare le regioni più piccole e separare quelle rimanenti per mezzo di *operatori morfologici*.

La prima operazione che facciamo consiste nel separare le varie regioni tra di loro utilizzando l'operatore morfologico di *erosione*. Dato che l'operatore funziona su immagini *binarie* creiamo, per ciascun cluster, una *maschera binaria*:

$$M_i(x) = \begin{cases} 1 & \text{se } x \in C_i \\ 0 & \text{se } x \notin C_i \end{cases}, \forall x \in \mathcal{X}.$$

L'operatore morfologico di erosione esegue un filtraggio non lineare che rimpiazza il valore di ciascun pixel della maschera M_i con il valore minimo tra quelli presenti in un suo intorno.

$$E_i(x) = \begin{cases} 1 & \text{se } \forall \xi \in U_x \quad M_i(\xi) = 1 \\ 0 & \text{se } \exists \xi \in U_x \quad M_i(\xi) = 0 \end{cases}, \forall x \in \mathcal{X},$$

dove abbiamo indicato con U_x l'intorno del pixel x . Il risultato di questo operatore è quello di ridurre i bordi delle immagini binarie.

L'immagine binaria E_i conterrà al suo interno un certo numero di *regioni connesse*. Le regioni più piccole, ovvero quelle il cui numero di pixel è inferiore allo 0,01% dei pixel totali, vengono eliminate ponendo a 0 il valore dei loro pixel. Una volta eseguite queste operazioni su tutte le maschere M_i , i risultati finali vengono composti assieme in una nuova immagine:

$$S(x) = \begin{cases} i & \text{se } x \in E_i, \quad i = 1, \dots, K \\ 0 & \text{altrimenti} \end{cases}, \forall x \in \mathcal{X}.$$

Infine, allarghiamo le aree di separazione tra le regioni utilizzando un altro operatore di *erosione*:

$$S'(x) = \begin{cases} i & \text{se } \forall \xi \in U_x \quad S(\xi) = i \\ 0 & \text{se } \exists \xi \in U_x \quad S(\xi) = 0 \end{cases}, \forall x \in \mathcal{X},$$

e vengono eliminati i cluster più piccoli, ovvero quelli il cui numero di pixel è inferiore al 2% dei pixel totali. Il risultato finale è quello mostrato in Figura 4.14(d), dove le zone colorate in blu scuro rappresentano i pixel che non vengono considerati durante l'analisi successiva.

Capitolo 5

Prove sperimentali e valutazione

Nel Capitolo 4 abbiamo illustrato gli algoritmi proposti per la soluzione del tampering detection. Tali algoritmi sono stati implementati nel linguaggio di programmazione MATLAB, in modo valutarne le prestazioni, confrontandoci con BOOOOM. In questo capitolo illustriamo come sono stati condotti questi esperimenti.

Il Paragrafo 5.2 è dedicato alla descrizione dei dataset utilizzati come riferimento, mostrando quali sistemi di acquisizione sono stati utilizzati e quali sono le metriche che abbiamo tenuto in considerazione.

Nel Paragrafo ??, invece, entriamo nel dettaglio su quali sono stati i risultati a valle di tutta l'analisi sperimentale.

5.1 Metodi considerati e obiettivi

- Performance con metodi più diretti (frame difference come soluzione in acquisizione continua)

$$d(t) = \frac{\sum_{x \in \mathcal{X}} (z_t(x) - z_{t-1}(x))^2}{|\mathcal{X}|},$$

- utilità di una segmentazione per lo spostamento della camera
- monitoraggio sequenziale per la sfocatura

5.2 Acquisizione dei dataset

Durante lo svolgimento della tesi abbiamo acquisito diversi dataset video, in modo da poter testare le soluzioni proposte per l'identificazione di eventi

di tampering e l'impatto della segmentazione sulle sue performance.

Un primo dataset è stato acquisito utilizzando una consumer camera *Nikon Coolpix*, in grado di registrare video. Questo primo insieme è stato utilizzato come *benchmark* per l'algoritmo di segmentazione, e comprendeva scene riprese all'aperto con un framerate continuo (30 fps). Ciascun video comprendeva un minuto di ripresa (1800 frame), e ciò lo rendeva inutilizzabile per validare le scelte fatte per l'algoritmo di tampering detection.

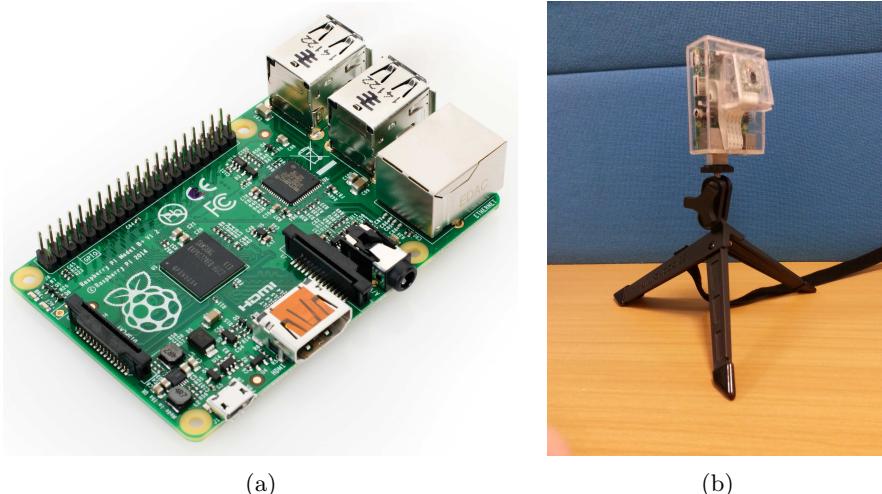
Per validare il nostro algoritmo, infatti, è necessario che le sequenze utilizzate comprendano un elevato numero di frame acquisito con framerate bassi, ad esempio un'immagine ogni minuto. Dato che la consumer camera non dava la possibilità di variare il framerate, abbiamo utilizzato un secondo sistema di acquisizione basato su un *Raspberry Pi modello B+* [45] con relativo *modulo camera* [46], in cui è possibile variare la frequenza di acquisizione dei frame a piacimento. Il Raspberry Pi (Fig. 5.1(a)) è un *single-board computer* (ovvero un computer implementato su una singola scheda elettronica) basato su un *system-on-chip* (SoC) *Broadcom BCM2835* [47], che incorpora un processore *ARM1176JZF-S* [48], una GPU *VideoCore IV* [49] e 512 MB di memoria. Utilizza un sistema operativo *Debian Linux* realizzato per processori ARM chiamato *Raspbian* [50].

Il modulo camera permette di acquisire immagini a diverse risoluzioni e a diverso framerate, fornendo inoltre la possibilità di salvarle in vari formati. Abbiamo deciso, quindi, di acquisire i frame in formato non compresso *yuv*, in modo da avere la maggior qualità possibile.

Il sistema realizzato è illustrato nella Figura 5.1(b). Le ridotte dimensioni e il basso consumo di potenza del sistema hanno permesso il suo utilizzo per fare le acquisizioni in ambienti esterni, utilizzando una batteria per alimentare il tutto. Per interfacciarsi con il dispositivo abbiamo creato una *connessione SSH*, tramite USB, con uno smartphone: in questo modo abbiamo potuto lanciare i comandi necessari per far partire l'acquisizione. Per l'acquisizione abbiamo creato uno script in *Python*, utilizzando la libreria *picamera* [51] per interfacciarsi con il modulo camera.

Abbiamo acquisito, infine, un terzo dataset utilizzando il sito *ilMeteo.it*. Questo portale di previsioni del tempo fornisce l'accesso ai frame acquisiti dalle webcam presenti in varie città e in varie località turistiche. A ciascuna webcam è associato un unico URL, in cui è presente un'immagine in formato *jpeg* rappresentante il frame corrente acquisito dalla camera. Queste webcam, in genere, acquisiscono un frame ogni minuto, e ciò ha permesso di utilizzarle come caso d'uso reale.

Attraverso uno script in Python e il tool *curl* [52], che permette il trasferimento di dati da indirizzi URL, abbiamo creato delle sequenze di frame che coprono un arco di 24 ore ciascuna, con frame acquisiti ogni minuto. In questo modo abbiamo potuto valutare il comportamento degli indicatori in condizioni di framerate basso e su lunghi periodi.



(a)

(b)

Figura 5.1: Il Raspberry Pi (a) e il sistema di acquisizione basato su di esso (b)

5.2.1 Eventi di tampering

Per testare i nostri algoritmi, abbiamo dovuto creare degli eventi in cui venga compromessa l’acquisizione corretta della scena ripresa. Nel caso del dataset acquisito con il Raspberry Pi abbiamo potuto introdurre degli eventi di tampering *reali*, ad esempio spostando la camera durante la ripresa, oppure buttando dell’acqua o spruzzando del deodorante spray sull’obiettivo della camera, in modo da creare la sfocatura.

Nel caso dei frame presi dalle webcam su internet abbiamo dovuto creare degli eventi di tampering in maniera *artificiale*. La sfocatura è stata introdotta utilizzando delle convoluzioni con filtri *gaussiani* di varie dimensioni, mentre lo spostamento della camera è stato simulato concatenando sequenze di frame differenti.

5.2.2 Definizione dei ground truth

Per valutare le prestazioni del nostro metodo, abbiamo dovuto definire un *ground truth* (GT) per ciascuna sequenza video, in modo da poter disporre di alcune metriche per poter confrontare i risultati ottenuti.

Dato che il nostro algoritmo identifica gli istanti in cui evento di tampering inizia, nel GT di ciascuna sequenza sarà presente, quindi, il numero del frame in cui questo evento inizia effettivamente. In particolare avremo un’indicazione del frame in cui avviene una sfocatura (se presente) e un’indicazione del frame in cui avviene lo spostamento della camera (se presente). Ricordiamo, inoltre, che nella nostra analisi stiamo considerando che il passaggio da una situazione prima di tampering a una con tampering come un evento *istantaneo*.

5.3 Figure di merito

5.4 Esperimento 1: tampering sintetico

5.4.1 Creazione dei tampering

5.4.2 Risultati

5.5 Esperimento 2: tampering reale

Capitolo 6

Direzioni future di ricerca e conclusioni

Bibliografia

- [1] [http://www.ilmeteo.it/webcam/.](http://www.ilmeteo.it/webcam/)
- [2] Antonio Manetti, Domenico De Robertis, and Giuliano Tanturli. *Vita di Filippo Brunelleschi: preceduta da La novella del Grasso*, volume 2. Milano: Il Polifilo, 1976.
- [3] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.
- [4] David A Forsyth and Jean Ponce. Computer vision: A modern approach. 2002.
- [5] Cesare Alippi. *Intelligence for Embedded Systems*. Springer, 2014.
- [6] Sheldon M. Ross. *Introduction to probability and statistics for engineers and scientists*. Academic Press, 2009.
- [7] Gordon J. Ross, Dimitris K. Tasoulis, and Niall M. Adams. Nonparametric monitoring of data streams for changes in location and scale. *Technometrics*, 53(4):379–389, 2011.
- [8] Cesare Alippi and Manuel Roveri. Just-in-time adaptive classifiers—part I: Detecting nonstationary changes. *Neural Networks, IEEE Transactions on*, 19(7):1145–1153, 2008.
- [9] Cesare Alippi, Giacomo Boracchi, and Manuel Roveri. A just-in-time adaptive classification system based on the intersection of confidence intervals rule. *Neural Networks*, 24(8):791–800, 2011.
- [10] A. Goldenshluger and A. Nemirovski. On spatially adaptive estimation of nonparametric regression. *Mathematical methods of Statistics*, 6(2):135–170, 1997.
- [11] George E.P. Box and David R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 211–252, 1964.

- [12] BFJ Manly. Exponential data transformations. *The Statistician*, pages 37–42, 1976.
- [13] Giacomo Boracchi and Manuel Roveri. A reconfigurable and element-wise ici-based change-detection test for streaming data. *Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA), 2014 IEEE International Conference on*, pages 58–63, 2014.
- [14] Cesare Alippi, Giacomo Boracchi, and Manuel Roveri. A hierarchical, nonparametric, sequential change-detection test. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 2889–2896. IEEE, 2011.
- [15] <http://www.mitan.it/security-solution/videosorveglianza/sistemi-di-videosorveglianza-e-registrazione/>. Visitato il giorno 09/03/2015.
- [16] Anil Aksay, Alptekin Temizel, and A. Enis Cetin. Camera tamper detection using wavelet analysis for video surveillance. In *Advanced Video and Signal Based Surveillance, 2007. AVSS 2007. IEEE Conference on*, pages 558–562. IEEE, 2007.
- [17] Ali Saglam and Alptekin Temizel. Real-time adaptive camera tamper detection for video surveillance. In *Advanced Video and Signal Based Surveillance, 2009. AVSS'09. Sixth IEEE International Conference on*, pages 430–435. IEEE, 2009.
- [18] Theodore Tsesmelis, Lars Christensen, Preben Fihl, and Thomas B Moeslund. Tamper detection for active surveillance systems. In *Advanced Video and Signal Based Surveillance (AVSS), 2013 10th IEEE International Conference on*, pages 57–62. IEEE, 2013.
- [19] Sebastien Harasse, Laurent Bonnaud, Alice Caplier, and Michel Desvignes. Automated camera dysfunctions detection. In *Image Analysis and Interpretation, 2004. 6th IEEE Southwest Symposium on*, pages 36–40. IEEE, 2004.
- [20] Pedro Gil-Jiménez, R. López-Sastre, Philip Siegmann, Javier Acevedo-Rodríguez, and Saturnino Maldonado-Bascón. Automatic control of video surveillance camera sabotage. *Nature Inspired Problem-Solving Methods in Knowledge Engineering*, pages 222–231, 2007.
- [21] Evan Ribnick, Stefan Atev, Osama Masoud, Nikolaos Papanikolopoulos, and Richard Voyles. Real-time detection of camera tampering. In *Video and Signal Based Surveillance, 2006. AVSS'06. IEEE International Conference on*, pages 10–10. IEEE, 2006.

- [22] Damian Ellwart, Piotr Szczyzko, and Andrzej Czyżewski. Camera sabotage detection for surveillance systems. In *Security and Intelligent Information Systems*, pages 45–53. Springer, 2012.
- [23] Nuno Roma, José Santos-Victor, and José Tomé. A comparative analysis of cross-correlation matching algorithms using a pyramidal resolution approach. 2002.
- [24] Tomasz Kryjak, Mateusz Komorkiewicz, and Marek Gorgon. Fpga implementation of real-time head-shoulder detection using local binary patterns, svm and foreground object detection. In *Design and Architectures for Signal and Image Processing (DASIP), 2012 Conference on*, pages 1–8. IEEE, 2012.
- [25] Stephane G Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 11(7):674–693, 1989.
- [26] Ronald N Bracewell. The fourier transform and its applications. 1978.
- [27] Irwin Sobel and Gary Feldman. A 3x3 isotropic gradient operator for image processing. 1968.
- [28] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.
- [29] Marco A.F. Pimentel, David A. Clifton, Lei Clifton, and Lionel Tarassenko. A review of novelty detection. *Signal Processing*, 99:215–249, 2014.
- [30] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer vision–ECCV 2006*, pages 404–417. Springer, 2006.
- [31] Cesare Alippi, Giacomo Boracchi, Romolo Camplani, and Manuel Roveri. Detecting external disturbances on the camera lens in wireless multimedia sensor networks. *Instrumentation and Measurement, IEEE Transactions on*, 59(11):2982–2990, 2010.
- [32] Ian F Akyildiz, Tommaso Melodia, and Kaushik R Chowdhury. A survey on wireless multimedia sensor networks. *Computer networks*, 51(4):921–960, 2007.
- [33] Jong Bae Kim and Hang Joon Kim. Efficient region-based motion segmentation for a video monitoring system. *Pattern Recognition Letters*, 24(1):113–128, 2003.

- [34] Dane P Kottke and Ying Sun. Motion estimation via cluster matching. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(11):1128–1132, 1994.
- [35] Yan Ke, Rahul Sukthankar, and Martial Hebert. Event detection in crowded videos. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.
- [36] Yining Deng and BS Manjunath. Unsupervised segmentation of color-texture regions in images and video. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(8):800–810, 2001.
- [37] Yihong Gong and Xin Liu. Video summarization using singular value decomposition. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 2, pages 174–180. IEEE, 2000.
- [38] Richard M Jiang, Abdul H Sadka, and Danny Crookes. Hierarchical video summarization in reference subspace. *Consumer Electronics, IEEE Transactions on*, 55(3):1551–1557, 2009.
- [39] Alessandro Sentinelli and Luca Celetto. Live key frame extraction in user generated content scenarios for embedded mobile platforms. In *MultiMedia Modeling*, pages 291–302. Springer International Publishing, 2014.
- [40] Wayne Wolf. Key frame selection by motion analysis. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, volume 2, pages 1228–1231. IEEE, 1996.
- [41] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan kaufmann, 2011.
- [42] Stuart Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.
- [43] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.
- [44] Tadeusz Caliński and Jerzy Harabasz. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27, 1974.
- [45] <http://www.raspberrypi.org/products/model-b-plus/>.

- [46] [http://www.raspberrypi.org/products/camera-module/.](http://www.raspberrypi.org/products/camera-module/)
- [47] [http://www.broadcom.com/products/BCM2835/.](http://www.broadcom.com/products/BCM2835/)
- [48] [http://www.arm.com/products/processors/classic/arm11/arm1176.php/.](http://www.arm.com/products/processors/classic/arm11/arm1176.php/)
- [49] [http://www.broadcom.com/products/technology/mobmm_videocore.php/.](http://www.broadcom.com/products/technology/mobmm_videocore.php/)
- [50] [http://www.raspbian.org/.](http://www.raspbian.org/)
- [51] [http://picamera.readthedocs.org/en/release-1.9/.](http://picamera.readthedocs.org/en/release-1.9/)
- [52] [http://curl.haxx.se/.](http://curl.haxx.se/)