UNIVERSITÀ DEGLI STUDI DI PADOVA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA DELLE TELECOMUNICAZIONI

Algoritmi per il rilevamento di alterazioni su sequenze video digitali

LAUREANDA: *Alessia Tiberto*

RELATORE: Prof. Simone Milani

CORRELATORE: Ing. Martino Jerian

Padova, 19 Aprile 2016

Indice

Sommario

Immagini, audio e video digitali sono largamente diffusi grazie allo sviluppo recente di dispositivi mobili in grado di registrare una traccia audio o una sequenza video ad alta qualità e condividerli sulle piattaforme social con un solo click. La facilità di condivisione di questi dati ha portato alla nascita del cosiddetto giornalismo collettivo, dove chiunque può fare notizia attraverso una propria immagine o un proprio video. Questo ha anche contribuito alla diffusione di notizie false poiché il meccanismo distribuito con il quale vengono generate e raccolte le notizie non permette sufficienti controlli sulle fonti o sull'autenticità dei dati utilizzati.

Immagini e video alterati vengono utilizzati consapevolmente per manipolare la propensione all'acquisto dei consumatori. La pubblicità ingannevole e la diffusione di notizie false sono tutt'oggi reati legalmente perseguibili.

I contenuti multimediali inoltre possono diventare fondamentali nell'ambito di investigazioni e processi giudiziari come elementi probanti a carico dell'accusa o della difesa. Per questo motivo sono necessari dei strumenti che permettano di stabilire l'integrità dei dati digitali in modo inequivocabile.

Una particolare attenzione viene dedicata ad immagini, audio e video: la loro potenza di comunicazione li rende dati importanti, in grado di influenzare notevolmente l'opinione pubblica.

Per immagini ed audio sono stati proposti molti metodi efficaci per valutarne l'autenticità ed esistono in commercio software per permettere anche a persone poco esperte di effettuare delle analisi su questo tipo di contenuti, in particolare in ambito legale. Per l'analisi di sequenze video invece non esistono ancora software in grado di eseguire una analisi veloce ed efficiente: finora la valutazione è stata affidata ad esperti che analizzano il contenuto in base alla propria esperienza e ad una valutazione visiva.

In questo elaborato viene descritto il processo che ha portato allo sviluppo di un software per l'analisi e localizzazione di video tampering, in collaborazione con l'azienda AMPED SRL che opera nel settore di analisi dati per applicazione forense da otto anni. Il metodo proposto è basato sul lavoro di Bestagini *et al.*[1] e si concentra su due modifiche possibili: *copy-paste* nel caso in cui una porzione di video sia sostituita con un'immagine fissa, *copy-move* nel caso in cui una porzione

di video sia sostituita con un'altra porzione. Per individuare un attacco copy-paste si sfrutta il fatto che da un frame all'altro l'area modificata risulta sempre identica. Per individuare un attacco copy-move invece si utilizza la correlazione di fase tra le diverse porzioni di video, che risulterà essere alta tra l'area modificata e l'area da cui è stata copiata.

Nel primo capitolo viene fatta un'introduzione alla digital forensics e alla video forensics, in particolare ai diversi tipi di modifiche che possono essere apportate ad un video.
Nel secondo capitolo sono illustrati i principali metodi esistenti per l'identificazione di video tampering, suddivisi per tipo di approccio.
Nel terzo capitolo viene descritto il metodo proposto, suddiviso nei due diversi approcci per l'attacco copy-paste e copy-move, e la sua implementazione ottenuta con l'utilizzo del linguaggio C++ e la libreria esterna OpenCV.
Nel quarto capitolo sono esposti i risultati ottenuti e le conclusioni riguardo questo progetto.

Capitolo 1

Introduzione

I dati digitali rappresentano sempre più spesso delle prove da utilizzare in ambito legale ma, a causa della loro flessibilità e facilità di manipolazione, la loro veridicità è spesso messa in discussione.
Nelle pagine di cronaca degli ultimi anni si è spesso letto come l'ultimo accesso ad certo file sul personal computer, l'aggancio ad una particolare cella effettuato dal proprio cellulare, o ancora un mezzo di trasporto ripreso da camere di sorveglianza siano stati fattori determinanti per delle indagini di importanza nazionale.
Nel giornale *The Huffington Post* del 7 Agosto 2015, la giornalista Alexis Sobel Fitts afferma che questi stessi dati possono essere modificati con facilità e da chiunque abbia accesso ad essi con lo scopo di eliminare eventuali prove di reati[2]. In particolare, Fitts riporta il caso di un video ripreso da una volante della polizia statunitense riguardante l'arresto di una donna di nome Sandra Bland, morta suicida nella propria cella tre giorni dopo. Le circostanze dell'arresto e della morte della donna hanno fatto nascere dei dubbi sul ruolo che ha avuto la polizia in tutto ciò e questi dubbi sono aumentati quando sono stati notati dei *loop* (ripetizioni) sospetti nel video fornito dalla polizia. In seguito il video è stato ripubblicato senza loop, a conferma del fatto che si era trattato di un problema causato dal sito su cui era stato caricato. L'intera faccenda ha messo in luce come sia necessario avere dei mezzi con cui certificare l'originalità di ciò che vediamo e ascoltiamo.

1.1 Digital forensics

La Digital Forensics è una scienza che si pone come obiettivo la ricerca di dati digitali e la verifica della loro integrità per l'utilizzo in ambito legale. Ciò rappresenta un'ardua sfida vista la facilità con cui questi dati possono essere generati e modificati. Da qui la necessità di meccanismi che permettano di identificare modifiche illecite dei contenuti, i cui risultati abbiano un margine di incertezza molto basso e che possano essere facilmente utilizzati anche da utenti poco esperti. Questi meccanismi devono inoltre rispettare certi parametri per poter essere usati in indagini e processi. Un'indagine di tipo forense infatti deve seguire una metodologia valida affinché i suoi risultati possano essere ritenuti validi.
L'intero processo di estrazione e analisi dei dati consiste in cinque fasi[3]:

1. *Individuazione*: deve essere identificata la posizione del dato che si vuole analizzare.

2. *Conservazione*: deve essere garantita l'integrità del dato originale, perciò è consigliabile utilizzare una copia del dato per effettuare le analisi.

3. *Protezione*: il dato originale va protetto da ogni possibile minaccia di alterazione, sia essa umana o non umana.

4. *Estrazione*: l'estrazione del dato necessario dal dispositivo in cui si trova deve avvenire nel modo più sicuro e meno invasivo possibile.

5. *Documentazione*: l'intero processo deve essere documentato passo dopo passo, includendo anche un registro di chi ha avuto accesso al dato originale e alle sue copie.

Riassumendo quindi, è di fondamentale importanza che qualunque meccanismo venga utilizzato durante l'analisi, questo non modifichi i dati originali o il dispositivo su cui si trovano.
La digital forensics si divide in diverse categorie, a seconda del tipo di dati o dispositivi analizzati: *network forensics*, *forensic data analysis*, *mobile device forensics*, *image forensics*, *video forensics*. Gli argomenti discussi in questa tesi si possono collocare nella video forensics, in particolare in quella sezione che si pone come obiettivo determinare se un video è originale o se ha subito delle modifiche in seguito alla sua creazione.

1.2 Video tampering

Il video tampering è la manomissione di un video, cioè la sua modifica illecita. Esistono diversi metodi per modificare un video e per ciascuno di essi esistono diversi approcci per individuarli.
Una prima distinzione viene fatta in base al momento di cui il video viene modificato: si parla di *tampering live* se la modifica avviene durante la registrazione stessa del video e di *tampering a posteriori* se la modifica avviene in un momento successivo.

Camera tampering

Il tampering live viene anche detto *camera tampering* perché non agisce sul video vero e proprio ma sulla camera che registra il video. Il camera tampering consiste nella manomissione di una camera installata allo scopo di impedirne il corretto funzionamento: ad esempio coprire una parte della camera, spostarla o cambiare il suo focus. Sono stati sviluppati molti software, rivolti in particolare al settore della videosorveglianza, in grado di identificare il camera tampering e inviare un segnale di allarme: in tal modo non è necessario che un operatore osservi i monitor ininterrottamente.
L'idea su cui si basano questi software è quella di confrontare i pixel del frame corrente con quelli precedenti: se esistono delle differenze significative allora è molto probabile che ci sia stata una manomissione[4]. I parametri su cui basare l'analisi devono essere scelti accuratamente: bisogna tener conto infatti dei cambiamenti di illuminazione (naturale e artificiale), parziali oscuramenti involontari (ad esempi il ramo di un albero che a causa del vento si sposta), oggetti in movimento che possono essere considerati normali (come una persona o una macchina).

Tampering a posteriori

I tipi di modifiche che possono essere fatte a posteriori sono molteplici: possono essere a livello di pixel o blocchi di pixel quando una parte del frame viene sostituita con un'altra (*tampering spaziale*), oppure a livello di frame quando dei frame vengono duplicati, eliminati oppure spostati (*tampering temporale*). La modifica infine può anche essere di tipo spazio-temporale cioè parti di una sequenza di frame possono sostituite con altre parti non necessariamente appartenenti alla stessa sequenza.

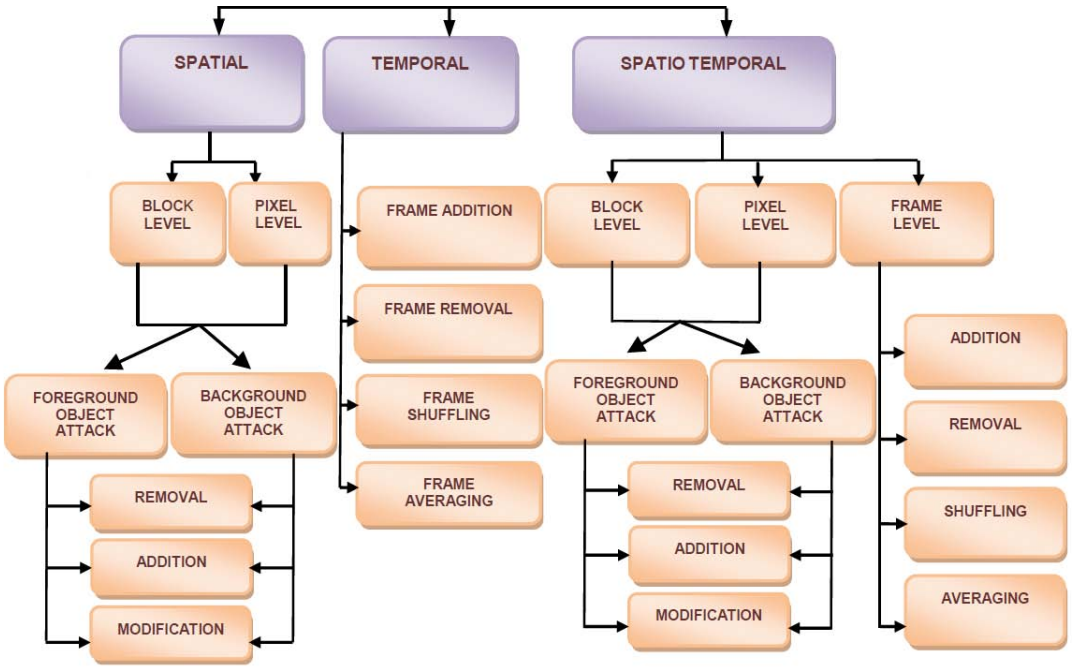


Figura 1.1: Distinzione dei tipi di tampering possibili[5].

Capitolo 2

Stato dell'arte

I meccanismi esistenti al giorno d'oggi si concentrano soprattutto sulla identificazione di tampering e raramente sulla localizzazione. I tipi di approccio inoltre possono essere molto differenti: alcuni optano per un approccio attivo, dove il file originale viene opportunamente modificato in modo da facilitare il compito dell'analista forense, altri adottano un approccio passivo, supponendo di avere a disposizione solo il file modificato. Inoltre molti di questi programmi partono da delle ipotesi non sempre valide nella Digital Forensics, come ad esempio conoscere il modello di camera con cui il video è stato registrato oppure conoscere a priori gli istanti in cui il video risulterà alterato. Tutte queste ipotesi non sono sempre attendibili: durante un'indagine forense è possibile utilizzare solamente un approccio *blind*, cioè minimizzando il numero delle condizioni poste a priori.

Nel seguente capitolo verranno presentati brevemente i metodi esistenti più importanti, suddivisi per tipo di approccio.

2.1 Approccio attivo

Gli algoritmi riconducibili a questo tipo di approccio consistono principalmente in strategie di autenticazione da applicare al file originale e che garantiscano la sua integrità.

Gli algoritmi appartenenti a questo tipo di approccio si basano sull'utilizzo di un algoritmo di crittografia asimmetrica, come ad esempio RSA. In particolare, ciò che garantisce l'autenticità dei dati inviati è l'utilizzo di chiavi da almeno 256 bit, che rendono questi meccanismi robusti a qualunque tipo di attacco. In

Figura 2.1 è rappresentato uno schema di crittografia asimmetrica in cui chi invia utilizza la sua chiave privata (segreta) e chi riceve utilizza la chiave pubblica del mittente per decifrare il messaggio.

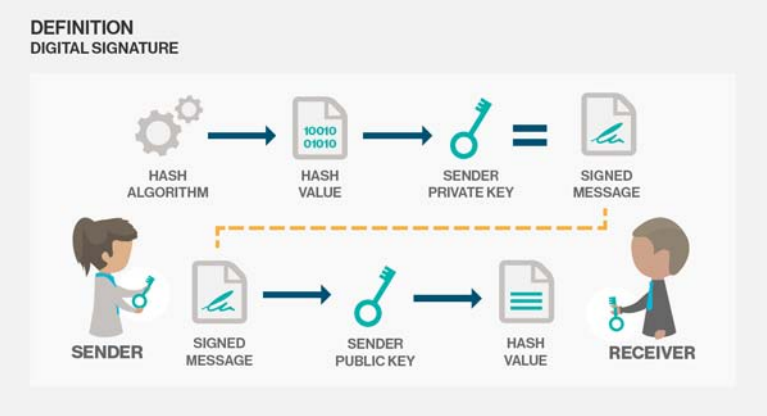


Figura 2.1: Schema di firma digitale[6].

Il ricevente può accorgersi di una prima manomissione se i dati che riceve sono stati crittografati con una chiave privata diversa da quella originale: in questo caso infatti il ricevente non sarà in grado di decifrare il messaggio. Inoltre, nel caso in cui il mittente abbia calcolato una sorta di firma (hashing) digitale sul dato originale, il ricevente può effettuare una verifica ricalcolando l'hashing sul dato ricevuto e confrontando il valore ottenuto con quello originale. Se non coincidono, il dato ricevuto è stato alterato.

I due principali algoritmi appartenenti a questa categoria sono il *watermarking* e la *digital signature*.

Watermarking

Il meccanismo di watermarking consiste nell'applicare una funzione hash a delle *feature* (zone di un frame poco ridondanti) e successivamente criptare il risultato dell'hashing assieme al valore dell'istante temporale in cui le feature sono state estratte[7]. La criptazione può essere fatta ad esempio utilizzando l'algoritmo a chiave asimmetrica RSA che garantisce un livello di sicurezza molto alta. La stringa risultante deve poi essere integrata nel video, assieme ad altre informazioni quali l'istante temporale riferito al GOP (*Group of Pictures*) in cui si trova.
Queste informazioni serviranno in fase di verifica, come mostrato in Figura 2.2: la feature serve ad identificare tampering spaziale mentre l'istante temporale serve

ad identificare il tampering temporale nel caso in cui il tampering sia inter-GOP, ciò se interessa solo un singolo GOP.
La procedura di watermark può inoltre essere applicata anche ad un intero GOP per identificare tampering a livello di GOP o tampering intra-GOP, cioè quel tipo di alterazione che coinvolge un intero GOP.

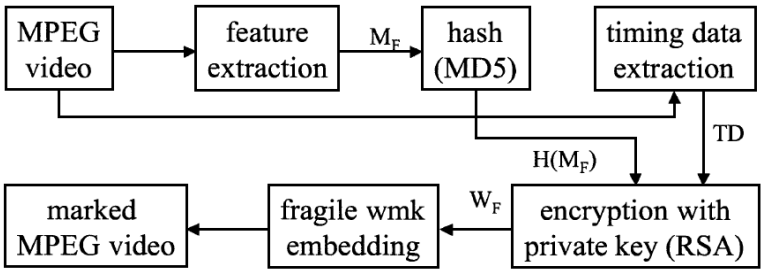


Figura 2.2: Schema di autenticazione con watermark[7].

I dati ottenuti dalla fase di autenticazione vengono inseriti all'interno del video stesso, preferibilmente nei frame di tipo I che sono codificati in maniera indipendente dal resto del video e hanno una maggiore capacità di embedding. La fase di verifica consiste nell'estrarre gli stessi dati utilizzati per il watermark (le informazioni riguardanti quali dati scegliere sono incluse nel video) e verificare che il risultato di hashing e criptazione coincidano con quelli contenuti del video. Se esiste una differenza nell'hashing si tratta di tampering spaziale, se invece riguarda la criptazione si tratta di tampering temporale (Figura 2.3).

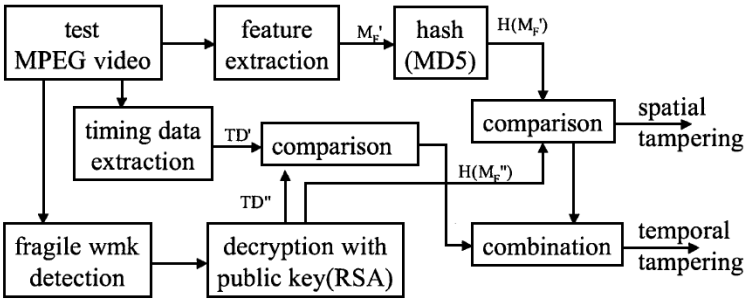


Figura 2.3: Schema di verifica con watermark[7].

Per i video codificati utilizzando lo standard di codifica MPEG vengono utilizzati come feature i coefficienti DCT (*Discrete Cosine Transformation*) su cui si basa la codifica stessa dal momento che essi possiedono una proprietà particolare: la differenza tra due coefficienti che si trovano nella stessa posizione ma calcolati

in istanti diversi resta pressoché invariata anche dopo una quantizzazione. Perciò i coefficienti DCT sono resistenti alle codifiche anche più robuste e rappresentano degli elementi ideali su cui utilizzare il watermarking.

Nel caso di codifica H.264/AVC (*Advanced Video Coding*) invece non è possibile utilizzare i coefficienti della DCT perché per essi non vale più la proprietà citata sopra. In [8] viene proposto di utilizzare dei parametri calcolati a partire dai coefficienti DCT. In particolare, per identificare il tampering spaziale i frame vengono suddivisi in macroblocchi e per ciascuno è calcolato un parametro a partire dai coefficienti DCT che viene poi criptato ed incluso nel video stesso. Per il tampering temporale invece l'indice di ciascun frame viene codificato ed inserito nel frame. In caso di alterazione dell'ordine dei frame, in fase di verifica ci sarà una discrepanza tra gli indici calcolati e quelli ricavati dallo stream video.

Digital signature

Il metodo di autenticazione basato su digital signature è molto simile al watermarking: in una prima fase vengono estratte le feature e nella seconda fase avviene la criptazione con un algoritmo a chiave pubblica. Questa procedura però viene ripetuta per ogni frame (a differenza del watermarking) e i dati criptati vengono inseriti in una parte specifica del video o addirittura in un file a parte.

Le feature devono essere scelte in modo da essere robuste rispetto ad una compressione lossy (con perdita di informazione). In [9] si decide di optare per i coefficienti DCT, sulla base del fatto che si sta considerando una codifica MPEG.

La digital signature rispetto al watermarking ha il vantaggio di poter generare dati destinati all'autenticazione senza grosse limitazioni: ciò è dovuto al fatto che i dati possono essere memorizzati a parte e non inseriti nel video stream come nel caso del watermarking. Inoltre, il fatto di non dover prevedere un meccanismo per l'aggiunta allo stream dei dati lo rende più semplice da realizzare.

2.2 Approccio passivo

I programmi che utilizzano un approccio passivo effettuano un'analisi sul video a posteriori, senza avere a disposizione i meccanismi di autenticazione presentati precedentemente. Nonostante ciò, molti di essi ipotizzano di avere a disposizione

alcuni metadati aggiuntivi come la camera utilizzata, il tipo di modifica che il video ha subito o in quali istanti questa è avvenuta. Alcuni metodi si pongono come obiettivo determinare se è avvenuta una modifica oppure no (*tampering detection*), altri cercano di identificare anche in quale punto del video e in che modo è stata applicata la modifica (*tampering localization*). Inoltre certi metodi si concentrano solamente sulle modifiche a livello di frame, altri solo su modifiche a livello spaziale ma pochi di questi si concentrano su casi misti, cioè modifiche sia a livello spaziale che temporale.
Questo approccio si basa sulla scelta di un parametro del video la cui continuità o discontinuità possa evidenziare un'alterazione con alta probabilità.

Camera identification

Ogni dispositivo che registra un video lascia delle "tracce", cioè un rumore definito *Photo Response Non-Uniformity* (PRNU). Dato un frame I_0 senza rumore, il frame corrispondente I acquisito dalla camera può essere descritto nel seguente modo:

$$I = I_0 + \gamma I_0 K + N$$

dove γ è un fattore moltiplicativo, K è il rumore PRNU e N rappresenta ulteriori rumori introdotti da sorgenti esterne. Ovviamente non è possibile disporre dei frame senza rumore, ma si può fare una stima di K a partire da un gruppo di frame con rumore ai quali si applica un apposito filtro di denoising. La scelta dei frame da utilizzare deve essere accurata, prediligendo scene ben illuminate.
Il PRNU può essere utilizzato per capire con quale dispositivo il video è stato acquisito ma sono possibili delle applicazioni anche nella video tampering detection[10].

In [11] viene proposto il seguente metodo: l'identificazione della camera viene eseguita al primo frame così da determinare le caratteristiche generali del rumore che corrompe i frame. Successivamente si procede al calcolo di tre diversi parametri per individuare altrettanti tipi di alterazione, sfruttando la correlazione esistente tra frame successivi. Per individuare una frame insertion si confronta il rumore del frame in esame con quello atteso: una discrepanza (cioè una bassa correlazione) può indicare che il frame non appartiene alla sequenza video. Nel caso di attacco cut-and-paste, cioè quanto una porzione di frame è stata copiata da un altro frame, il rumore del frame corrente viene confrontato con quello del frame

successivo: in presenza dell'area alterata ci si aspetta un valore di correlazione molto basso. Infine per la duplicazione di frame si confrontano i frame tra di loro: la coppia formata da due frame esattamente uguali avrà un'alta correlazione. Questo metodo presenta problemi nel caso di sequenze codificate.

Multiple compression detection

Un passo importante ed obbligato nell'editing di video è la compressione: la quantità di dati che un video mediamente contiene infatti è troppo elevata per poter essere mantenuta intatta. Si ricorre quindi alla compressione che ha il vantaggio di rendere i video facilmente trasmissibili e memorizzabili ma d'altra parte introduce una perdita di informazione.

Una prima compressione avviene nel dispositivo di acquisizione (la videocamera o il cellulare con i quali viene registrata la sequenza video), ma possono essere effettuate successive compressioni da parte dell'utente proprietario del video (ad esempio, qualora questo voglia ridurre la dimensione del file codificandolo ad un bit rate più basso). Al fine di modificare il contenuto dei frame è necessario decodificare il video, effettuare la modifica e poi comprimerlo nuovamente. Inoltre quando un video viene caricato online, subisce un'ulteriore compressione da parte dei codificatori del sito che effettua l'hosting (ad esempio YouTube, Facebook, Vimeo, ecc.)

Il numero di compressioni a cui un video è stato sottoposto può essere utilizzato come parametro per determinare l'autenticità del video. In molti casi ci si è concentrati sulla double detection, ovvero determinare se ci siano state due compressioni ma, come già detto in precedenza, un video originale potrebbe essere stato sottoposto anche a più compressioni per esigenze di dimensione.

Farid *et al.*[12] hanno proposto per primi un metodo per la double MPEG compression detection basato sullo studio dei coefficienti DCT. Supponendo che il video sia stato compresso due volte e che il livello di quantizzazione sia diverso nei due casi, è ragionevole supporre di conoscere il parametro (passo di quantizzazione) dell'ultima quantizzazione effettuata $q2$ (in quanto il video è disponibile in formato compresso e tale informazione è decodificabile dal bit stream). La distribuzione di probabilità dei coefficienti DCT permette di determinare il parametro della prima quantizzazione $q1$. Questa analisi viene fatta a livello di macroblocco, quindi può essere utile per individuare tampering in cui vengono inseriti

dei pixel con un livello di quantizzazione diverso da quello del video, oppure che sono stati generati dall'utente e quindi non hanno subito alcuna compressione in precedenza.

Sun *et al.*[13] hanno poi ampliato questo metodo con l'utilizzo della legge di Benford. Inizialmente viene estratta la cifra più significativa (*first digit*) dei coefficienti AC della DCT (cioè di tutti i coefficienti della DCT eccetto il primo) dai frame di tipo I (meno sensibili alla codifica rispetto ai frame di tipo P e B) e si verifica se queste seguono la relazione data dalla legge di Benford:

$$p(m) = Nlog_{10}(1 + \frac{1}{m})$$

dove N è una costante di normalizzazione. In caso affermativo, il video è stato compresso una sola volta mentre, in caso negativo, è possibile che sia avvenuta una seconda compressione. Ci sono infatti dei casi in cui i coefficienti non seguono questa legge nonostante il video sia autentico quindi per prendere una decisione finale viene utilizzato un classificatore *Support Vector Machine* (SVM) che utilizza un training set composto da video originali.

Milani *et al.*[14] hanno proposto un metodo più efficace che può determinare il numero compressioni effettuate senza limiti nel numero, indipendentemente dal tipo di codec utilizzato. L'analisi fatta sui coefficienti DCT può infatti essere estesa anche al caso in cui il codec utilizzi la motion estimation. Per determinare il numero di compressioni sono necessarie delle SVM più sofisticate rispetto all'esempio precedente, è necessario inoltre avere a disposizione dei training set composti da video compressi fino a N volte, dove N rappresenta il numero massimo di compressioni che possono essere identificate dal metodo. Nell'articolo in questione ci si è limitati a considerare il caso $N = 3$, visto che un numero di compressioni uguale a 2 può essere considerato normale mentre $N = 3$ può far già nascere dei sospetti.

La compression detection di per sè non rappresenta un metodo efficace per l'identificazione di tampering perché non garantisce che ci sia stata effettivamente una modifica o quale essa sia stata. Potrebbe essere utile per identificare quei tipi di tampering in cui vengono aggiunti dei frame o parti di frame provenienti da altri video e con una compressione differente.

Histogram of Oriented Gradients

In [15] viene proposto un metodo per identificare un attacco di tipo copy-paste ovvero quando una porzione di un frame viene sostituita con un'altra porzione fissa che può provenire dallo stesso frame (copy-paste spaziale) oppure da un altro frame (copy-paste temporale), per una certa durata di tempo. Il metodo si basa sulla statistica definita *Histogram of Oriented Gradients* (HOG), calcolata a partire dai frame. Questo parametro viene generato per i diversi blocchi dei frame, i quali verranno confrontati tra di loro per identificare un attacco copy-paste spaziale in presenza di forte somiglianza.
Il calcolo avviene nel seguente modo:

1. $$G_x(x,y) = [-1\ 0\ 1] * I(x,y)$$
$$G_y(x,y) = [-1\ 0\ 1]^T * I(x,y)$$
$$\Theta(x,y) = arctan(G_y(x,y)/G_x(x,y))$$
rappresentano rispettivamente gradiente orizzontale, gradiente verticale e orientazione per il pixel in posizione (x,y).

2. Ciascun frame è suddiviso in NxN blocchi e ciascun blocco a sua volta è suddiviso in MxM celle, con M $\leq$ N che dipende dalle caratteristiche della sequenza.

3. Per ogni pixel in posizione (x, y), si definisce un array di lunghezza L in cui l-esimo valore è definito in questo modo:
$$\Omega_l(x,y) = \begin{cases} G(x,y) & \text{se } \Theta(x,y) \in \text{bin}_l \\ 0 & \text{altrimenti} \end{cases}$$
dove $G(x,y) = \sqrt{G_x(x,y)^2 + G_y(x,y)^2}$ e bin rappresenta un array di L valori equi-spaziati compresi tra 0° e 180° nel caso di gradienti senza segno o compresi tra 0° e 360° nel caso di gradienti con segno.

4. Per ogni cella vengono calcolate delle feature di dimensione uguale a bin, in cui l-esimo elemento è normalizzato così:
$$f_l = \frac{\sum_{(x,y)\in Cell} \Omega_l(x,y) + \epsilon}{\sum_{(x,y)\in Block} G(x,y) + \epsilon}$$
dove ϵ è un valore molto piccolo.

5. Le feature di tutte le celle appartenenti ad uno stesso blocco formano un vettore d che rappresenta il descrittore HOG per quel blocco.

L'algoritmo poi procede in due modi differenti a seconda del tipo di alterazione che intende identificare. Per il copy-paste spaziale, confronta i descrittori dei blocchi appartenenti allo stesso frame mentre per copy-paste temporale confronta blocchi appartenenti a frame diversi. Se c'è una forte somiglianza tra due blocchi, allora c'è stata un'alterazione.

Adjustable Width Object Boundary

Richao *et al.*[16] hanno proposto un metodo che si concentra sul caso di aggiunta o cancellazione di oggetti da un video. Come prima fase, l'oggetto deve essere identificato: il metodo più semplice per fare ciò è il background subtraction, con l'ausilio di un modello statistico come il Gaussian Mixture Model. Dopodiché viene definito l'*Adjustable Width Object Boundary* (AWOB), cioè il contorno dell'oggetto ottenuto nella fase precedente, che ha prima subito un numero n di dilatazioni.
Sul contorno dell'oggetto e sull'area da esso identificata viene applicata una trasformata wavelet, i cui coefficienti presentano delle discontinuità in presenza di tampering. A partire da questi coefficienti vengono calcolate sei diverse feature che sono utilizzate da un classificatore Support Vector Machine per ottenere il risultato finale.
Il metodo descritto funziona solamente in caso di scena statica e l'utilizzo di SVM richiede che l'utente conosca a priori quali sequenze sono originali e quali sono state invece alterate, quindi questo non è un metodo blind.

Motion Compensated Edge Artifact

In [17] viene proposto un metodo per l'identificazione di tampering a livello di frame (frame insertion, frame deletion e frame shuffling) basato sui coefficienti DCT e applicabile su sequenze compresse con MPEG. In particolare, a partire da questi coefficienti viene calcolato il parametro MCEA (*Motion Compensated Edge Artifact*) per i frame di tipo P. In questo caso infatti MCEA dipende strettamente dalla motion estimation eseguita a livello di codifica e, in caso di tampering temporale, comporta dei cambiamenti di questo parametro. Nel metodo proposto la procedura per l'identificazione di tampering prevede di osservare la differenza

dei MCEA di frame adiacenti: se nella trasformata di Fourier di questi valori compaiono dei picchi, allora c'è stato tampering (Figura 2.4). Inoltre osservando lo spettro della trasformata di Fourier è possibile capire di che tipo di modifica si tratta e, nel caso di frame shuffling, ricostruire la struttura originaria del GOP.

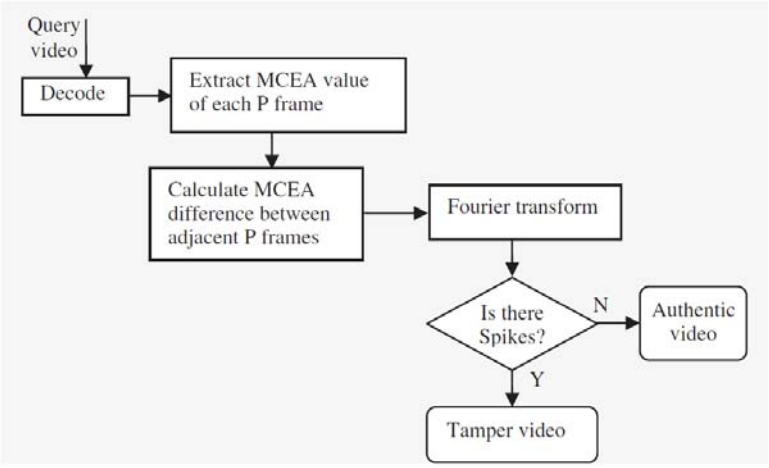


Figura 2.4: Schema del metodo basato su MCEA[17].

Il calcolo del parametro MCEA avviene nel seguente modo:

1. Viene calcolato il parametro

$$C_\tau^n = \sum_{(i,j)\in N} (c_\tau^n(i,j))^2$$

dove τ è l'indice del blocco di dimensione 8x8 considerato, N rappresenta l'insieme dei coefficienti DCT AC e $c_\tau^n(i,j)$ sono i coefficienti DCT residui ricevuti.

2. Viene calcolato poi

$$M_\tau^n = \sum_{(i,j)\in N} (m_\tau^n(i,j))^2$$

dove $m_\tau^n(i,j)$ sono i coefficienti DCT ottenuti a partire dai frame ricostruiti.

3. Un terzo parametro è

$$P_\tau^n = \sum_{(i,j)\in N} (m_\tau^n(i,j) - c_\tau^n(i,j))^2$$

4. Infine viene stimato il parametro

$$E_\tau^n = \sum_{\beta\in\sigma(\tau)} \omega(\beta) \sum_{(i,j)\in N} (m_\beta^{n-1}(i,j))^2$$

dove $\sigma(\tau)$ è l'insieme di quattro blocchi nel frame n-1 utilizzati per predire il blocco di indice τ del frame n e $\omega(\beta)$ è un coefficiente attribuito al blocco β.

5. I quattro parametri precedenti vengono combinati in

$$\mu_\tau^n = (P_\tau^n - E_\tau^n) - C_\tau^n + \sum_{\beta \in \sigma(\tau)} \omega(\beta)\mu_\beta^{n-1}$$

dove μ_τ^n rappresenta MCEA per il singolo blocco τ.

6. Viene calcolato la media del parametro μ_τ^n sui vari blocchi ottenendo

$$MCEA = \frac{\sum_{\tau \in T} \mu_\tau^n}{M^n} \quad .$$

Come già detto, il metodo prende in considerazione solamente i frame di tipo P e funziona solamente con video compressi utilizzando il codec MPEG. Inoltre non sono forniti dati precisi sull'effettiva percentuale di successo dell'algoritmo.

Noise Correlation

Nel metodo proposto in [18] invece ci si basa sullo studio del rumore presente nel video. La prima fase del metodo proposto consiste nell'ottenere il rumore di ogni singolo frame a partire dai coefficienti ad alta frequenza della trasformata wavelet. Successivamente, i frame sono suddivisi in blocchi e si calcola la correlazione tra un blocco e quello nella stessa posizione nel frame successivo.

In questo paper vengono presi in considerazione due tipi di alterazione: copy-paste temporale (una regione fissa è copiata per un certo numero di frame) e la tecnica *example-based texture synthesis* che consiste nell'utilizzare i pixel attorno alla regione per cancellare uno specifico oggetto. Per queste due tecniche differenti, ci si aspetta dei valori diversi di correlazione tra blocchi successivi (vedi Figura 2.5).

Nel caso di copy-paste temporale, nella regione copiata il rumore resterà sempre identico (salvo alcune differenze dovute alla successiva compressione) quindi ci si aspetta un valore di correlazione tra blocchi molto vicino ad 1. Nell'altro caso invece, in ciascun frame la regione viene modificata in modo indipendente dai frame adiacenti quindi ci si aspetta correlazione molto bassa.

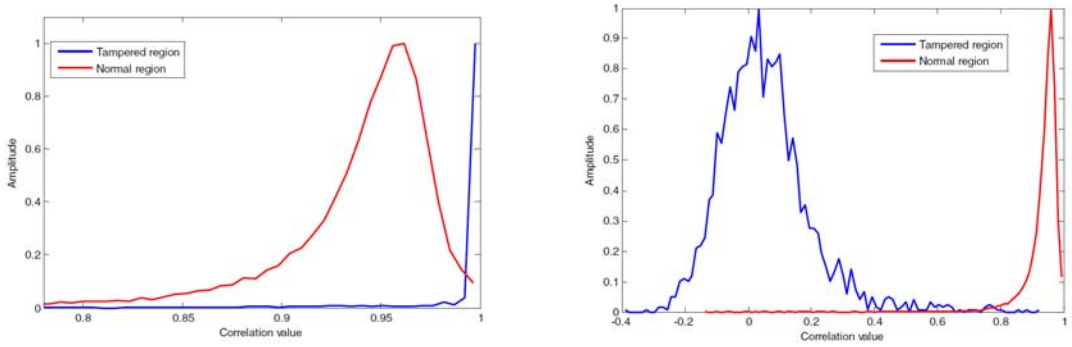


Figura 2.5: Correlazione del rumore di blocchi successivi nel caso di copy-paste temporale (figura a destra) e example-based texture synthesis (figura a sinistra)[18].

Nella fase successiva si distinguono i blocchi certamente originali da quelli che potrebbero aver subito un'alterazione ponendo una soglia sulla loro correlazione. Infine per quei frame il cui numero di blocchi modificati è al di sopra di un valore fissato si utilizza il *Metodo Generalizzato dei Momenti* (GMM), supponendo che nella sequenza ci siano due distribuzioni gaussiane corrispondenti alle due diverse tecniche di alterazione considerate.

La percentuale di successo è al di sopra del 90% se le sequenze video non sono state ricompresse dopo la modifica ma questo valore cala in maniera esponenziale all'aumentare del valore di quantizzazione utilizzato o in caso di scene in movimento.

Color Correlation

La correlazione viene utilizzata anche da Wang e Farid[19], i quali però la applicano non sul rumore ma direttamente sui valori delle componenti di colore dei frame del video. Propongono due procedure per identificare copy-move a livello di frame e copy-paste a livello di regione.

Per il caso di copy-move a livello di frame, si procede innanzitutto con l'identificare delle sotto-sequenze di lunghezza n, che si sovrappongono per la lunghezza di un frame. Questo implica ipotizzare che la duplicazione avvenga per un numero di frame successivi uguale o superiore ad n. Per ogni sotto-sequenza S_τ si calcola la correlazione per ogni coppia di frame appartenente alla sequenza stessa e i valori ottenuti vengono salvati nella matrice T_τ. Successivamente, viene calcolata la correlazione tra le due matrici T_{τ_1} e T_{τ_2} per ogni coppia τ_1 e τ_2 e se questa supera una certa soglia vicina ad 1, è possibile che ci sia un frame duplicato in una

delle due sotto-sequenze. Come mostrato in Figura 2.6, si pone un'altra soglia sul mimino valore della matrice T_{τ_1}: questo serve per escludere quelle sequenze completamente statiche che quindi presentano valori molto alti di correlazione e posso portare a falsi positivi.

Sono quindi calcolate le matrici $B_{\tau_1,k}$ e $B_{\tau_2,k}$ per $k \in [0, n-1]$ e, se tutti i valori ottenuti sono al di sopra di una certa soglia, la sotto-sequenza τ_1 è considerata alterata.

```
FRAMEDUP(f(x, y, t))
 1  ▷ f(x, y, t): video sequence of length N
 2
 3  ▷ n: sub-sequence length
 4  ▷ γ_m: minimum temporal correlation threshold
 5  ▷ γ_t: temporal correlation threshold
 6  ▷ γ_s: spatial correlation threshold
 7
 8  for τ = 1 : N − (n − 1)
 9        do S_τ = {f(x, y, t + τ) | t ∈ [0, n − 1]}
10           build T_τ ▷ temporal correlation
11
12  for τ_1 = 1 : N − (2n − 1)
13        do for τ_2 = τ_1 + n : N − (n − 1)
14                 do if (min(T_τ1) > γ_m & C(T_τ1, T_τ2) > γ_t)
15                       build B_τ1,k ▷ spatial correlation
16                       build B_τ2,k ▷ spatial correlation
17                       if ( C(B_τ1,k, B_τ2,k) > γ_s ) ▷ ∀k
18                             do ▷ Frame Duplication at τ_1
```

Figura 2.6: Pseudo-codice per l'identificazione di copy-move a livello di frame[19].

Per identificare copy-paste a livello di regione invece si utilizza la correlazione di fase, la quale permette di localizzare all'interno del frame in quale posizione si trova il valore maggiore di correlazione. Vengono distinti due casi: nel primo si suppone che la scena sia statica e nel secondo che la scena sia in movimento.

Se la scena è statica, innanzitutto per ogni coppia di frame $f(x, y, \tau_1)$ e $f(x, y, \tau_2)$ è calcolata la correlazione di fase e se in posizione (δ_x, δ_y) della matrice di correlazione è presente un valore maggiore di una soglia fissata, (δ_x, δ_y) rappresenta di quanto la regione duplicata risulterà spostata rispetto all'originale.

Si dividono quindi i frame in blocchi di 16x16 (sovrapposti di 1 pixel) e per ciascun blocco del frame nell'istante τ_1 viene calcolata la correlazione con il blocco corrispondente nell'istante τ_2, tenendo conto dell'offset calcolato nella fase precedente. Se la correlazione è sufficientemente alta, il blocco viene segnato come

alterato e se nel frame in τ_1 sono presenti un certo numero minimo di blocchi alterati, allora i risultati trovati sono confermati. L'intera procedura è decritta in Figura 2.7.

```
REGIONDUP1(f(x,y,τ1), f(x,y,τ2))
 1  ▷ f(x,y,τ1), f(x,y,τ2): two video frames
 2
 3  ▷ γp: phase correlation threshold
 4  ▷ γo: phase correlation offset threshold
 5  ▷ b: block neighborhood size
 6  ▷ γb: block correlation threshold
 7  ▷ γa: minimum area threshold
 8
 9  compute p(x,y) ▷ phase correlation
10  for each (Δx,Δy), s.t. p(Δx,Δy) > γp and
                            (|Δx| > γo or |Δy| > γo)
11        do for each i,j
12              do i' = i + Δx
13                 j' = j + Δy
14                 b1 = f(i : i+b-1, j : j+b-1, τ1)
15                 b2 = f(i' : i'+b-1, j' : j'+b-1, τ2)
16                 if ( C(b1,b2) > γb ) ▷ correlation
17                       do mask(i,j) = 1
18           mask = connected_components(mask)
19           if ( area( mask(x,y) ) > γa )
20                 do ▷ Region Duplication at (x,y)
```

Figura 2.7: Pseudo-codice per l'identificazione di copy-paste a livello di regione in caso di scena statica[19].

In caso di scena in movimento, l'algoritmo appena descritto non funziona in modo corretto in quanto appaiono come alterati dei blocchi i quali però di fatto si sono solamente spostati a causa del movimento della camera. Nel secondo metodo proposto quindi si calcola lo spostamento della camera e si procede con l'algoritmo precedente, richiedendo però che l'offset (δ_x, δ_y) sia superiore all'offset dovuto alla camera (Figura 2.8).

L'algoritmo per il copy-move a livello di frame è risultato robusto alla compressione e con percentuale di successo attorno a 85% nel caso di video statici e superiore all' 87% nel caso di video in movimento. I tempi di esecuzione però sono elevati: su un video di dimensione 480x720 pixel e 10000 frame l'algoritmo impiega 45 minuti per effettuare i calcoli necessari.

Il metodo per l'identificazione di copy-paste invece è risultato molto sensibile alla dimensione dell'area alterata: l'algoritmo ha ottime prestazioni con regioni

```
REGIONDUP2(f(x, y, t), τ1, τ2)
 1  ▷ f(x, y, t): video sequence
 2  ▷ τ1, τ2: two frame indices
 3  ▷ frame size: Nx × Ny pixels
 4
 5  ▷ s: motion camera offset threshold
 6
 7  for τ = τ1 : τ2 − 1
 8        do compute p(x, y) for f(x, y, τ) and f(x, y, τ + 1)
 9           (δx, δy) = arg max_{x,y}(p(x, y))
10           (Δx, Δy) = (Δx, Δy) + (δx, δy)
11  if ( Δx > sNx and Δy > sNy )
12        do REGIONDUP1( f(x, y, τ1), f(x, y, τ2) )
```

Figura 2.8: Pseudo-codice per l'identificazione di copy-paste a livello di regione in caso di scena in movimento[19].

grandi (256x256 pixel) ma pessime prestazioni per regioni più piccole (64x64 pixel). Il tempo di esecuzione inoltre è di gran lunga superiore al caso precedente: ciò è dovuto alla complessità dei calcoli necessari per la correlazione di fase.

Capitolo 3

Algoritmo proposto e implementazione

Come visto nel capitolo precedente, esistono molti metodi proposti per l'identificazione di tampering video. Pochi di essi però sono utilizzabili anche per la localizzazione e i risultati che forniscono sono spesso fortemente dipendenti dalle informazioni iniziali possedute o eccessivamente complessi da ottenere dal punto di vista computazionale.

Per la progettazione del software è stato scelto il metodo proposto da Paolo Bestagnini *et al.* in [1], il quale permette l'identificazione e la localizzazione di alterazione video con un approccio completamente blind. Il metodo viene suddiviso in due fasi, con l'obiettivo di riconoscere due tipi di attacco diversi definiti *copy-paste* e *copy-move*. In entrambe le fasi, si decide di lavorare con la sola componente di *luminanza* del video, cioè con frame in scala di grigio perché il colore non è un dato necessario per l'algoritmo.

3.1 Algoritmo per identificazione di alterazione copy-paste

Un attacco di tipo copy-paste consiste nel sostituire una porzione di alcuni frame del video con un'immagine fissa (Figura 3.1). L'immagine utilizzata per l'attacco può provenire dal video stesso oppure da una sorgente esterna (un'altra immagine o video). L'algoritmo è in grado di identificarla in entrambi i casi.

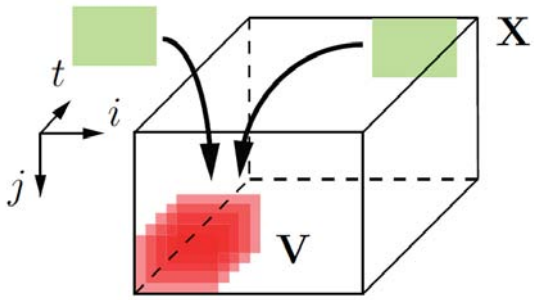


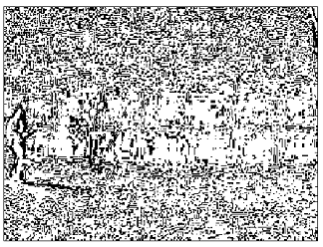
Figura 3.1: Schema dell'attacco copy-paste[1].

Ciò su cui si basa l'identificazione della modifica è l'immobilità data dall'immagine rispetto al resto del video. Anche in una scena statica esiste un movimento minimo o del rumore di acquisizione legato alla camera che rende pixel adiacenti di frame temporalmente differenti sempre diversi l'uno dall'altro, anche se queste differenze non sono visibili dall'occhio umano. Nella regione in cui è stata copiata l'immagine esterna tuttavia queste differenze vengono meno.

Tale effetto viene influenzato dal livello di compressione. Infatti l'operazione di compressione riduce la presenza di alte frequenze nell'immagine ricostruita (cioè opera come un filtro passa basso). Di conseguenza le piccole differenze relative ad esempio al rumore di acquisizione vengono mediate rendendo le regioni non alterate simili a quelle alterate. Pertanto tale proprietà è utilizzabile qualora il livello di compressione del video non sia eccessivamente alto. Le figure 3.2 e 3.3 mostrano questo effetto nel caso del video fuji_2800_outdoor(2).avi e della sua versione modificata forged_fuji_2800_outdoor(2).avi. L'immagine binaria (c) rappresenta la differenza assoluta tra due frame, in cui i pixel bianchi indicano dove tale differenza è esattamente nulla.

(a) Frame 20.

(b) Frame 21.



(c) Differenza tra frame 20 e frame 21.

Figura 3.2: Video fuji_2800_outdoor(2).avi .

(a) Frame 20. (b) Frame 21. (c) Differenza tra frame 20 e frame 21.

Figura 3.3: Video forged_fuji_2800_outdoor(2).avi .

Confrontando ciascun frame con il successivo, vengono prese in considerazione quelle zone in cui la differenza è esattamente nulla generando una mappa binaria dove ogni pixel vale 0 o 1 a seconda del valore della differenza. Viene pertanto applicata ad esse un'operazione di *erosione*.
La procedura di erosione consiste nell'applicare ad un'immagine binaria A un *elemento strutturante* B e impostare tutti gli elementi della finestra in A al minimo elemento contenuto nella finestra stessa.
Formalmente:

$$A \ominus B = \{z | B_z \cap A^c \neq \emptyset\}$$

dove B_z è la traslazione di B rispetto al vettore z, B è una matrice i cui elementi sono tutti uguali a 1 e A^c è il complementare di A. In pratica, ciascun elemento di A sarà settato al valore 0 se nella finestra definita dalle dimensioni di B tutti gli elementi (eccetto quello centrale) sono a 0. In questo caso A e B sono strutture in tre dimensioni e le dimensioni dell'elemento strutturante permettono di definire le dimensioni spaziali e la durata temporale minima di una zona per essere considerata alterata. L'erosione permette quindi di eliminare quelle zone che con poca probabilità sono risultato di una modifica (Figura 3.4).

Il software realizzato unisce i risultati ottenuti dall'erosione con tre diverse dimensioni di B: una prima erosione serve per escludere le zone non alterate, successive erosioni con elementi strutturanti più piccoli in dimensioni spaziali servono a definire in modo più preciso le zone restanti. Tra queste ultime, vengono scartate quelle al di sotto di una certa dimensione che con alta probabilità sono dei falsi positivi e quelle uniformi che potrebbero essere causate da un alta compressione in una regione uniforme come un muro o il cielo. La libreria OpenCV

(a) Frame 240 prima dell'erosione.

(b) Frame 240 dopo l'erosione con elementro strutturante di dimensione 16x16x30.

Figura 3.4: Video forged_flower_garden.264 .

contiene il metodo *getStructuringElement* che permette di costruire l'elemento strutturante di dimensione e forma desiderata e il metodo *erode* che esegue il processo di erosione.

Di seguito sono descritte le fasi del programma, ripetute per sequenze di frame di lunghezza massima pari a 600:

1. Conversione del video nel formato yuv utilizzando il software FFmpeg.
2. Lettura della componente di luminanza dei frame a partire dal file yuv.
3. Costruzione della matrice binaria tridimensionale R su cui si effettueranno i successivi calcoli. Un pixel uguale a 1 indica che la differenza tra due frame successivi è esattamente nulla, in caso contrario il pixel sarà settato a 0:

```
per ogni istante t e per ogni i,j
{
    r_temp = X(i,j,t) - X(i,j,t+1);
    if (r_temp == 0)
        R(i,j,t) =1;
    else
        R(i,j,t) =0;
}
```

4. Divisione della matrice in blocchi, in maniera proporzionale rispetto al parametro *scale_t* definito nel seguente modo:

```
WIDTH = larghezza frame;
```

```
if (WIDTH > 1900)
    scale_t = 4;
if (WIDTH > 1200)
    scale_t = 3;
if (WIDTH > 700)
    scale_t = 2;
if (WIDTH <= 700)
    scale_t = 1;
```

Il numero di blocchi in cui la matrice è divisa in ogni istante t e la loro dimensione è calcolato come di seguito:

```
numero blocchi = scale_t*scale_t * 2;
altezza blocco = floor(HEIGHT / scale_t);
larghezza blocco = floor(WIDTH / (scale_t * 2));
```

5. Ogni blocco viene analizzato come descritto nei seguenti passi:
   - si procede all'erosione in tre dimensioni con elemento strutturante di dimensione BxBxT con $B = 16$ e $T = 30$ (metodo *erodeBlock*);
   - per ogni istante $t > 0$ e multiplo di 5, se i pixel alterati in quell'istante sono di numero maggiore rispetto ad una soglia $W\_min = 300 * scale\_t$, allora verranno salvati in un'apposita mappa;
   - se nel punto precedente è stata rilevata un'alterazione, si ripete l'intera procedura per il blocco in esame diminuendo la dimensione spaziale dell'elemento strutturante, ponendo $B = 8$ e successivamente $B = 4$ per poter ottenere maggior precisione nei risultati finali.
6. Se nell'intera sequenza non sono state trovate alterazioni, si diminuisce la dimensione temporale dell'elemento strutturante, ponendo $T = 20$ o $T = 10$, e si ripete il punto 5.
7. Se risultano esserci dei frame alterati, la mappa binaria contenente i pixel alterati viene ulteriormente processata per eliminare possibili falsi positivi. La sequenza di operazioni effettuate viene riportata di seguito:
   - la mappa binaria viene erosa con elemento strutturante di dimensione 8x8 per eliminare una parte dei falsi positivi;

- viene applicato il metodo *deleteNoise*, che scarta le aree piccole utilizzando una soglia *area_min* proporzionale a *scale_t*:

```
if (scale_t==1)
     area_min = 200;
if (scale_t==2)
     area_min = 500;
if (scale_t==3)
     area_min = 800;
if (scale_t==4)
     area_min = 1000;

if (area rilevata < area_min)
    scarta l'area in esame;
```

- sulla mappa viene operata un'operazione di dilatazione con elemento strutturante di dimensione 6x6 per riempire le zone rimaste;
- è applicato nuovamente il metodo *deleteNoise*;
- viene ora utilizzato il metodo *connectArea*: se due pixel alterati sono abbastanza vicini, cioè se la loro distanza è minore o uguale a *distMin*, possono essere considerati alterati anche i pixel tra loro compresi. La distanza massima che due pixel possono avere è proporzionale a *scale_t*:

```
if (scale_t==1)
    distMin = 20;
else
    distMin = 40;
```

8. Infine viene eseguito il metodo *evaluateDraw*, che procede con lo scartare le aree con deviazione standard < 5. Aree di questo tipo infatti sono nella maggior parte dei casi dei falsi positivi individuati dal programma poiché la loro uniformità fa sì che da un frame all'altro non ci siano variazioni.
   Le aree rimaste invece vengono riportate con un bordo rosso nel frame originario.

3.2 Algoritmo per identificazione di alterazione copy-move

L'attacco di tipo copy-move consiste nel sostituire una porzione di alcuni frame con un'altra porzione (Figura 3.5). Si fanno alcune ipotesi: che le due porzioni siano prese in istanti diversi e che provengano dallo stesso video. Queste ipotesi sono legittime in quanto, in caso contrario, le modifiche possono essere visibili anche ad occhio nudo e il software non è più necessario.

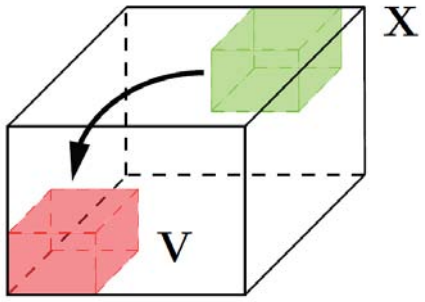


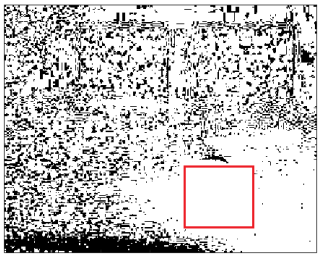
Figura 3.5: Schema dell'attacco copy-move[1].

Il parametro su cui si basa l'algoritmo è la *correlazione*, ovvero la somiglianza tra porzioni di frame (*blocchi*). A causa del rumore della camera e delle compressioni, la correlazione tra due blocchi qualunque è sempre bassa mentre, nel caso di modifica copy-move, la correlazione tra i due blocchi coinvolti sarà alta poiché affetti dallo stesso rumore (con alcune piccole differenze dovuto al rumore dell'ultima compressione).

In Figura 3.6 (c) è rappresentata la differenza assoluta tra il frame 70 e il frame 131 del video 02_forged.avi. In particolare, i pixel bianchi indicano che la differenza tra i due frame in quella posizione è esattamente nulla, cioè l'area evidenziata dal bordo rosso risulta essere identica nel frame 70 e nel frame 131. Nella Figura 3.7 (c) invece, nonostante l'apparente somiglianza tra i frame 70 e 232, l'area in questione risulta differente. Nel primo caso quindi è possibile che sia avvenuta un'alterazione, cioè che un'area del frame 70 sia stata copiata nel frame 131 o viceversa mentre nel secondo caso si può escludere che ci sia stata un'alterazione di tipo copy-move. Ovviamente per prendere una decisione definitiva è necessario osservare se l'area in questione appare modificata anche nei frame precedenti e/o successivi.

(a) Frame 70.

(b) Frame 131.



(c) Differenza tra frame 70 e frame 131.

Figura 3.6: Video 02_forged.avi, frame 70 e 131.

(a) Frame 70.

(b) Frame 232.

(c) Differenza tra frame 70 e frame 232.

Figura 3.7: Video 02_forged.avi, frame 70 e 232.

Il blocco potrebbe essere copiato così com'è oppure subire delle modifiche nel colore o nella luminosità. Per questo invece di lavorare direttamente sui dati della luminanza,il software realizzato elabora la differenza di questi ovvero per ciascun frame X viene generata una matrice delle differenze R i cui valori sono:

$$r^t_{i,j} = x^t_{i,j} - x^{t+1}_{i,j} \qquad \forall i,j \in X$$

dove $x^t_{i,j}$ è il valore del pixel in posizione i,j all'istante t.

Inoltre il blocco potrebbe essere stato ruotato o scalato: per questo motivo si sceglie di utilizzare la *correlazione di fase* invece della correlazione classica. La correlazione di fase per ogni coppia di residui R_{t_1} e R_{t_2} può essere calcolata utilizzando la loro rispettiva trasformata di Fourier $\mathcal{R}_{t_1}$ e $\mathcal{R}_{t_2}$ nel seguente modo:

$$\mathcal{C} = \frac{\mathcal{R}_{t_1} \circ \mathcal{R}^*_{t_2}}{|\mathcal{R}_{t_1} \circ \mathcal{R}^*_{t_2}|}$$

dove $\mathcal{C}$ è la trasformata di Fourier della matrice della correlazione.

Nel software la correlazione di fase viene calcolata tra ciascun blocco (definito in tre dimensioni) e l'intera sequenza video per individuare, in presenza di valori alti, l'istante temporale in cui potrebbe esserci stato un tampering e la posizione del blocco alterato. La Figura 3.8 mostra l'andamento tipico dei valori di correlazione per uno specifico blocco, in relazione a tutti gli istanti temporali del video. Quando un blocco è originale, questo è correlato in maniera mediamente omogenea con l'intera sequenza mentre se il blocco è stato alterato ci saranno due picchi più elevati in corrispondenza dell'istante in cui il blocco inizia e l'istante in cui è presente il blocco a lui identico. La correlazione con i restanti frame invece sarà mediamente bassa.

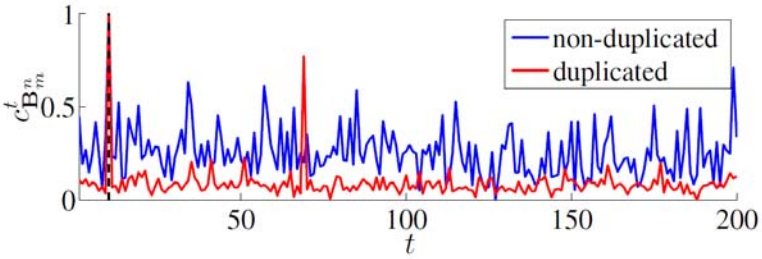


Figura 3.8: Esempio di valori di correlazione per un blocco nei vari istanti temporali[1].

Per distinguere i possibili blocchi modificati da quelli originali ci basiamo sulla proprietà che emerge dalla Figura 3.8, ovvero che il valore di correlazione media per un blocco alterato sarà più alto rispetto a quello di un blocco originale. Definiamo quindi il valore medio del picco massimo come:

$$p_{\mathbf{B}_m^n} = \frac{\max(c^t_{\mathbf{B}_m^n})}{\frac{1}{(T-1)}\sum_t c^t_{\mathbf{B}_m^n}}$$

dove $\mathbf{B}_m^n$ rappresenta il blocco di indice m (riferito alla sua posizione all'interno del singolo frame) che inizia all'istante n.

Per quei blocchi con valore $p_{\mathbf{B}_m^n}$ al di sopra di una certa soglia, è possibile che ci sia stata un'alterazione di tipo copy-move. Se questa procedura non è in grado di individuare un blocco con correlazione sufficientemente alta, allora siamo nel caso di un falso positivo e il blocco in questione viene scartato.

Di seguito sono descritte le fasi del programma:

1. Conversione del video nel formato yuv utilizzando il software FFmpeg.

2. Si definiscono alcuni parametri:

```
scale_t = 5; // parametro variabile
DT = 10;
if (larghezza frame <=700)
    DX = 8;
if (larghezza frame > 700)
    DX = 16;
if (larghezza frame > 1200)
    DX = 18;
if (larghezza frame > 1900)
    DX = 22;
T = floor((n_FR - DT - 3) / DT);

if (DX >=18)
    peak_threshold = 9.5;
else
    peak_threshold = 7.5;
```

$1/scale\_t$ rappresenta la scala da applicare ai frame, per poter ridurre la complessità computazionale del programma. Questo parametro determina in modo fondamentale la qualità dei risultati del programma. Dai test è emerso che porre $scale\_t$=5 è sufficiente per ottenere buoni risultati nel caso in cui i blocchi non siano molto dettagliati. Ma, in caso contrario, è necessario diminuire la scala per poter ottenere dei risultati più significativi.

DX e DT sono rispettivamente la dimensione spaziale e temporale dei blocchi su cui verrà eseguita l'analisi descritta precedentemente e rappresentano le dimensioni minime che un'area alterata deve possedere per poter essere individuata dall'algoritmo. Si è scelto di determinare DX in modo proporzionale in quanto ci si aspetta che l'area modificata sia adeguata alla dimensione del frame.

Infine $T*DT$ rappresenta l'istante massimo da considerare come indice n per un blocco. L'utilizzo di *peak_threshold* verrà definito in seguito.

3. Lettura della componente di luminanza dei frame a partire dal file yuv.

4. Costruzione della matrice tridimensionale R su cui si effettueranno i successivi calcoli:

```
per ogni istante t e per ogni i,j
     R(i,j,t) = | X(i,j,t) - X(i,j,t+1)|;
```

5. Calcolo della DFT (Discrete Fourier Transform) in tre dimensioni della matrice R.

6. Calcolo della correlazione di fase e localizzazione dei picchi per ciascun blocco di dimensione DXxDXxDT:

```
per ogni istante t=0:DT:T*DT
{  per ogni blocco
   {  calcolo della correlazione di fase;
      peak_mean = valore medio del picco massimo;
      if (peak_mean >= peak_threshold)
      {   localizzazione dei picchi con valore >=
          peak_rate * picco massimo;
          if (numero picchi trovati ==2 && istante del
              secondo picco compreso tra DT e T*DT )
          {   salva istanti e posizioni dei due picchi
              in una mappa;
          }
      }
   }
}
```

7. Vengono eliminati i blocchi isolati, in quanto possiamo ipotizzare che un'area alterata debba comparire in almeno 30 frame per considerare quei frame alterati con copy-move.

```
per ogni istante t=0:DT:T*DT
{  if (numero blocchi alterati in mappa(t) è compreso tra
       1 e 2)
```

```
        {  if (non ci sono blocchi alterati nei due frame
               precedenti e nei due successivi)
           {   elimina dai risultati tutti i blocchi in
               mappa(t);
           }
        }
}
```

8. I risultati vengono riportati alla dimensione originale e vengono salvati i frame corrispondenti evidenziando il contorno dell'area alterata con colore rosso.

3.3 Implementazione del programma

L'intero programma è stato implementato con l'utilizzo del linguaggio C++ e della sua libreria standard assieme alla libreria esterna OpenCV e il software FFmpeg.

La libreria OpenCV fornisce strutture e metodi per leggere, analizzare e creare immagini e video. La libreria è compatibile con le principali piattaforme (Windows, Linuc, Mac OS X, Android) e il suo linguaggio principale è il C++, sebbene sia compatibile anche con C, Python e Java. Da qui la scelta di sviluppare il programma interamente in C++, utilizzando sia strutture standard come i *vector* (vettori con caratteristiche simili agli array ma di più semplice utilizzo) sia strutture della libreria OpenCV come le *Mat* (matrici di due o più dimensioni). In particolare, si è deciso di optare per il solo utilizzo di vettori unidimensionali e Mat a due dimensioni. Durante l'implementazione del programma infatti è emerso che l'utilizzo di vettori tridimensionali, benché più intuitivi quando si lavora con video, è particolarmente costoso in termini di memoria. Della libreria OpenCV vengono utilizzati alcuni metodi che permettono di elaborare le Mat, come ad esempio *findContours* che permette di trovare i contorni su matrici binarie, il metodo *erode* per effettuare l'erosione e il metodo *dilate* per la dilatazione. Abbiamo inoltre utilizzato il metodo *dft* fornito dalla libreria per calcolare le trasformate di Fourier in tre dimensione.

Il software FFmpeg viene utilizzato per convertire il video nel formato YUV420p, di cui poi verrà utilizzata la sola componente di luminanza Y. L'uti-

lizzo di un software esterno per la conversione è dettato dall'esigenza di conoscere a priori il numero di frame di cui il video è composto in modo da ottimizzare le prestazioni del programma in termini di tempo di esecuzione. La libreria OpenCV offre un metodo per conoscere il numero di frame di un video ma non sempre questo restituisce il valore esatto mentre, avendo il file in formato raw, è possibile ricavare la lunghezza del video a partire dal numero di byte del file in questione.

Capitolo 4

Risultati e Conclusioni

Nel seguente capitolo verranno presentati i risultati ottenuti con l'algoritmo proposto, evidenziando le sue prestazioni in termini di *true positive rate* (TP), *true negative rate* (TN) e tempo di esecuzione.

Il true positive rate rappresenta la percentuale di pixel effettivamente alterati individuati dal programma mentre il true negative rate rappresenta la percentuale di pixel effettivamente originali individuati. Nel nostro caso questi due parametri non rappresentano la percentuale di successo del programma a livello di singolo frame ma a livello di intera sequenza. Scopo dell'algoritmo infatti non è la sola identificazione di tampering ma anche la sua precisa localizzazione nella sequenza e nel frame.

Il tempo di esecuzione è un parametro altrettanto importante per la valutazione del programma: un problema particolarmente difficile da risolvere nel campo dell'analisi video infatti è la gestione efficiente del gran numero di dati a disposizione. Questo implica dover scegliere con cura i tipi di struttura dato da utilizzare e ottimizzare il codice dove è possibile.

I risultati verranno suddivisi nelle due categorie di alterazioni su cui ci siamo concentrati, ovvero copy-paste e copy-move.

Per testare il programma sono stati utilizzati video di diversa dimensione, formato e provenienza. In Appendice A sono presenti tutti i video con la loro provenienza e le modifiche effettuate. Una parte di essi sono stati presi dal SULFA dataset[20] (*Surrey University Library for Forensic Analysis*), un database di video rivolto specificatamente al test di programmi per la digital forensics. Nello stesso sito sono presenti dei video in versione originale e modificata con tecnica copy-

move realizzati da P. Bestagini, S. Milani, M. Tagliasacchi e S. Tubaro. Tutti questi video sono di dimensione 320x240 pixel, di lunghezza variabile da 210 a 583 frame.

Per poter testare il programma su sequenze più grandi, sono stati utilizzati dei video scaricabili dal sito Xiph.org nella sezione *Derf's collection*[21], poi in seguito modificati sia con tecnica copy-paste che copy-move.

I test sono stati eseguiti su pc con la seguente configurazione hardware: Intel(R) Core(TM) i7 CPU, 2.93 GHz, RAM 8 GB, Windows 7.

4.1 Risultati programma copy-paste

Dal test è risultato che il programma è sensibile alla codifica subita dal video e alle regioni uniformi. Una codifica che comporti una riduzione sostanziale di informazione infatti riduce il rumore generato dalla camera al momento della registrazione su cui si basa l'algoritmo. In presenza di regioni praticamente uniformi come il cielo o in caso di scena statica, una codifica troppo elevata può eliminare quasi completamente questo rumore rendendo di fatto queste regioni dei falsi positivi per il programma. Per risolvere questo problema si è pensato all'eliminare dai risultati le regioni uniformi, per le quali l'algoritmo non è in grado di distinguere veri positivi da falsi positivi.

In Figura 4.1 sono riportati i valori di TP, TN e tempo di esecuzione per i video testati. I video contrassegnati con (*) sono quelli provenienti dal database SULFA, per i quali non era disponibile un *ground truth*, cioè una mappa con indicazione precisa di quali pixel fossero stati modificati. Si è scelto quindi di utilizzare una mappa ottenuta dalla semplice differenza con sogliatura tra il video originale e quello alterato. Questo però ha portato ad avere dei valori di TP che non rappresentano i veri valori in quanto dalla differenza non emergono quelle zone molto simili al background ma di fatto alterate.

I video contrassegnati con (**) invece provengono dal sito Xiph.org e sono caratterizzati da sfondi quasi uniformi che, nel caso di codifica con QP (*Quantization Parameter*) settato a 28, hanno introdotto molti falsi positivi (Figura 4.2). Dal sito non è stato possibile risalire alla esatta provenienza dei video ma è ipotizzabile che questi abbiano subito una precedente codifica a livello di frame

che quindi ha eliminato quasi completamente il rumore presente in queste zone particolari.

	TP	TN	Execution time (seconds)
can_220_van_car_Forge.MOV (320x240 , 361 frames) (*)	0.9388	0.9485	3
forged_fuji_2800_street.avi (320x240 , 266 frames) (*)	0.4542	0.9742	2.6
forged_fuji_2800_outdoor.avi (320x240 , 270 frames) (*)	0.4341	0.9633	2.6
forged_fuji_2800_road.avi (320x240 , 330 frames) (*)	0.9759	0.9981	4
forged_fuji_2800_man.avi (320x240 , 330 frames) (*)	0.9628	0.8688	2.5
forged_highway_qp20.264 (QP 20) (352x288 , 2001 frames)	0.8212	1	31.3
forged_highway.264 (QP 28) (352x288 , 2001 frames)	0.8422	0.9999	31
forged_flower_garden.264 (QP 28) (720x486 , 361 frames)	0.6528	0.9943	10.8
forged_johnny2_qp20.264 (QP 20) (1280x720 , 601 frames)	0.9411	0.9960	43
forged_johnny2.264 (**) (QP 28) (1280x720 , 601 frames)	0.9701	0.4495	67.5
forged_KristenAndSara_big2.264 (QP 20) (1280x720 , 601 frames)	0.9434	0.9514	44
forged_KristenAndSara_big.264(**) (QP 28) (1280x720 , 601 frames)	0.9648	0.5174	67
forged_KristenAndSara2.264 (QP 20) (1280x720 , 601 frames)	0.9231	0.9479	43
forged_KristenAndSara.264 (**) (QP 28) (1280x720 , 601 frames)	0.9563	0.5152	65
forged_vidyo3_qp20.264 (QP 20) (1280x720 , 601 frames)	0.9420	0.9877	37.6
forged_vidyo3.264 (**) (QP 28) (1280x720 , 601 frames)	0.9392	0.6616	67.5
forged_aspen_qp20.264 (QP 20) (1920x1080 , 570 frames)	0.7010	1	94
forged_aspen.264 (QP 28) (1920x1080 , 570 frames)	0.8555	0.9982	150
forged_blue_sky_qp20.264 (QP 20) (1920x1080 , 217 frames)	0.6987	1	29.2
forged_blue_sky.264 (QP 28) (1920x1080 , 217 frames)	0.7701	1	33
Valori medi:	**0.8343**	**0.8886**	

Figura 4.1: Prestazioni del programma copy-paste in termini di true positive rate (TP), true negative rate (TN) e tempo di esecuzione.

(a) Video originale, frame 375.

(b) Video alterato (QP 20), frame 375.

(c) Video alterato (QP 28), frame 375.

Figura 4.2: Video KristenAndSara_1280x720_60.y4m .

Un caso che può essere considerato come tentativo di anti-forensic per copy-paste è il video forged_fuji_2800_outdoor(2).avi . Il copy-paste per questo video è stato realizzato copiando un singolo frame per una breve durata (da 3 a 7 frame) e poi sostituendolo con un altro proveniente da un istante temporale diverso. Questo rende impossibile all'algoritmo originario di individuare la modifica in quanto si richiede che il tampering abbia una durata minima di 10 frame. Per risolvere questo problema, nell'algoritmo proposto è stato aggiunto un metodo che tiene conto di questa possibilità e permette quindi di individuare l'alterazione. Il metodo effettua una sorta di dilatazione nel tempo:

```
metodo dilateTime
 {   if (pixel corrente == 0)
        if (tre pixel precedenti e i due successivi == 1)
            pixel corrente = 1;
 }
```

Il tempo di esecuzione del programma dipende non solo dalla dimensione del video in esame ma anche dal numero di pixel individuati dal programma stesso. Come si può vedere dalla Figura 4.1, il tempo di esecuzione è sempre sotto al secondo, eccetto per il video forged_aspen.264 che risulta essere il video più grande da noi testato.

In tutti i casi da noi considerati, i video hanno una durata limitata tra i 10 e i 18 secondi e non rappresentano quindi un caso reale. Avendo un video di alcuni minuti, il tempo richiesto dal programma ovviamente aumenterebbe ma si può pensare di richiedere all'utente si selezionare una finestra temporale di interesse. Inoltre il programma legge ed esamina sequenze di 600 frame alla volta, salvando i risultati ottenuti alla fine di ogni sequenza e dando quindi la

possibilità all'utente di visualizzare in tempo reale tali risultati ed eventualmente terminare il programma qualora essi soddisfino le sue esigenze.

Un altro test per verificare il comportamento del programma rispetto a compressioni multiple è stato effettuato caricando sul sito YouTube tre video, sia in versione compressa (con FFmpeg, utilizzando il codec libx264) che in versione raw utilizzando un contenitore AVI. Ciò che si è potuto osservare è che la compressione effettuata dal sito è molto pesante ed introduce più falsi positivi rispetto alla compressione che abbiamo fatto nel test precedente. D'altra parte anche il numero di veri positivi è aumentato, nei casi di scena statica o zona uniforme, ad indicare il fatto che la compressione fatta dal sito elimina ulteriormente il rumore della camera e rende frame vicini altamente correlati.

	TP	**TN**
forged_aspen.avi compressed (*)	0.8673	0.9779
forged_aspen.avi not compressed (**)	0.8618	0.9706
forged_blue_sky.avi compressed (*)	0.7701	1
forged_blue_sky.avi not compressed (**)	0.7701	1
forged_flower_garden.avi compressed (*)	0.6528	0.9941
forged_flower_garden.avi not compressed (**)	0.6528	0.9931

Figura 4.3: Prestazioni del programma copy-paste con video scaricati dal sito YouTube.

4.2 Risultati programma copy-move

Come nel caso precedente, nella Figura 4.4 sono riportate le prestazioni del programma copy-move. Emerge in primis il tempo di esecuzione, decisamente maggiore rispetto al programma copy-paste. Questo è dovuto al fatto che sono necessari calcoli più complessi per la correlazione di fase e la scalatura, nonostante aiuti a diminuire la complessità, non è sufficiente a portare il tempo di esecuzione entro limiti accettabili.

	TP	TN	Execution time (seconds)
01_forged.avi (320x240, 210 frames)	0	1	48
02_forged.avi (320x240, 330 frames)	0.9338	0.9741	127
03_forged.avi (320x240, 314 frames)	0.8933	0.9525	108
04_forged.avi (320x240, 320 frames)	0.8716	0.9730	127
05_forged.avi (320x240, 583 frames)	0.7018	0.9837	480
06_forged.avi (320x240, 262 frames)	0.8712	0.9626	80
07_forged.avi (320x240, 412 frames)	0.7280	0.9777	215
08_forged.avi (320x240, 274 frames)	0.8837	0.9764	83
09_forged.avi (320x240, 554 frames)	0.6488	0.9865	435
10_forged.avi (320x240, 239 frames)	0.8073	0.9570	66
forged_gardenMove.264 (720x486, 361 frames)	0.8615	0.9400	978
forged_iceMove.264 (704x576, 480 frames)	0.4368	0.9955	2020
forged_parkrunMove.264 (1280x720, 504 frames)	0.8035	0.9683	6860
forged_blue_skyMove.264 (1920x1080, 217 frames)	0.7853	0.9663	6028
forged_aspenMove.264 (1920x1080, 570 frames)	0.6800	0.9907	50697
Valori medi:	0.7271	0.9736	

Figura 4.4: Prestazioni del programma copy-move per scale_t=5.

I valori di true negative rate (TN) sono abbastanza alti ma mai uguali ad 1 (eccetto quando il programma non individua alcuna alterazione): questo perché l'algoritmo non è in grado di distinguere il blocco originale dal blocco copiato e quindi per ogni vero positivo individua anche un falso positivo.

I valori di true positive rate invece (TP) variano tra 0.43 e 0.93. Un fattore che incide sulla percentuale di successo è l'utilizzo di una scala che, se da una parte fa guadagnare in termini di complessità dall'altra riduce la quantità di informazioni utilizzabili dall'algoritmo e quindi la precisione dei risultati. Inoltre il programma riesce ad individuare solo le zone con movimento all'interno di esse, poiché le zone statiche sono altamente correlate con l'intera sequenza video e quindi hanno un valore di picco mediato basso. Ciò porta a valori inferiori di TP per quelle sequenze in cui il copy-move è stato eseguito su zone statiche come in forged_iceMove.264. Altro fattore che incide sul TP è la dimensione dei blocchi, che produce dei risultati approssimativi.

In Appendice C sono riportate alcune figure riguardanti i test eseguiti con scale_t=5. Il programma si è dimostrato robusto in presenza di oggetti praticamente indistinguibili ad occhio nudo ma di fatto diversi, come nel caso del video 02_forged.avi.

Nel video 05_forged.avi i risultati sono pochi rispetto all'effettiva durata del tampering. Ciò è dovuto al fatto che la zona alterata è molto particolareggiata e i dettagli, a causa della scalatura, sono persi. Eseguendo il test con *scale_t*=2 infatti si ottengono risultati decisamente migliori ma a scapito del tempo di esecuzione, che risulta essere circa 3,5 volte maggiore rispetto al test con scala a 5. Stessa cosa accade con il video 01_forged.avi: nel caso scale_t = 5 il programma non individua nessuna alterazione mentre con scale_t=2 il programma dà buoni risultati ma con tempi più lunghi (Figura 4.5).

Una possibile soluzione potrebbe essere quella di lasciare all'utente la determinazione del valore di *scale_t*, in cui il valore 5 rappresenta un'analisi con media precisione e man mano che il valore diminuisce la precisione aumenta.

La complessità dei calcoli però resta ancora un problema: la precisione del programma è indirettamente proporzionale alla sua durata.
Per risolvere questo problema si potrebbe chiedere all'utente di selezionare un'area di interesse (sia a livello spaziale che a livello temporale) e dopodiché attuare un meccanismo automatico per la scelta del parametro *scale_t* a seconda della

quantità di dettagli presente in quell'area.

	TP	TN	Execution time (seconds)
01_forged.avi (320x240, 210 frames)	0.1464	0.9317	191
02_forged.avi (320x240, 330 frames)	0.9470	0.9548	311
03_forged.avi (320x240, 314 frames)	0.7842	0.9648	423
04_forged.avi (320x240, 320 frames)	0.9279	0.9273	489
05_forged.avi (320x240, 583 frames)	0.8223	0.9181	1773
06_forged.avi (320x240, 262 frames)	0.9107	0.9050	311
07_forged.avi (320x240, 412 frames)	0.7523	0.9544	859
08_forged.avi (320x240, 274 frames)	0.9114	0.9451	341
09_forged.avi (320x240, 554 frames)	0.7027	0.9619	1641
10_forged.avi (320x240, 239 frames)	0.8184	0.9243	254
forged_gardenMove.264 (720x486, 361 frames)	0.8644	0.9340	3701
forged_iceMove.264 (704x576, 480 frames)	0.4523	0.9816	9857
forged_parkrunMove.264 (1280x720, 504 frames)	0.8799	0.9353	39199
forged_blue_skyMove.264 (1920x1080, 217 frames)	0.7921	0.9731	37798
Valori medi:	0.7575	0.9436	

Figura 4.5: Prestazioni del programma copy-move per scale_t=2.

4.3 Conclusioni

In questo elaborato è stato presentato un metodo blind per l'identificazione e localizzazione di alterazione su sequenze video, rivolto ai tipi di attacchi copy-paste e copy-move sia a livello di frame che a livello di pixel.

L'algoritmo orginale è quello proposto da Paolo Bestagnini e Simone Milani in [1], al quale sono poi state apportate delle modifiche per migliorare i risultati ottenuti. L'algoritmo è stato implementato in C++ utilizzando le librerie standard, la libreria esterna OpenCV e il software FFmpeg.

Il programma per l'attacco copy-paste ha dato ottimi risultati in termini di true positive rate, true negative rate e tempo di esecuzione, con prestazioni peggiori nel caso di video con sfondi uniformi e che hanno subito una forte compressione.

True positive rate e true negative rate del programma copy-move sono risultati essere inferiori al caso precedente: ciò è dovuto alle approssimazioni fatte per diminuire i tempi di calcolo. Tempi che, nonostante ciò, sono rimasti abbastanza alti (sempre al di sopra del minuto e fino a 13 ore nel caso forged_aspenMove.264.). Inoltre in caso di alterazioni su aree molto dettagliate, la scalatura applicata risulta essere decisiva nei risultati finali.

Come sviluppo futuro quindi si potrebbe prevedere che l'utente selezioni un'area specifica di interesse e la scala da applicare potrebbe essere scelta automaticamente a seconda della quantità dei dettagli dell'area selezionata.

Appendice A

Video dataset

Dal SULFA dataset[20] sono stati presi i seguenti video:

- van_car.MOV
- can_220_van_car_Forge.MOV
- fuji_2800_man (2).avi
- forged_fuji_2800_man (2).avi
- fuji_2800_outdoor(2).avi
- forged_fuji_2800_outdoor(2).avi
- fuji_2800_road(1).avi
- forged_fuji_2800_road(1).avi
- fuji_2800_street(1).avi
- forged_fuji_2800_street.avi

Disponibili nello stesso sito[20]:

- 01_original.avi
- 01_forged.avi
- 02_original.avi
- 02_forged.avi

- 03_original.avi
- 03_forged.avi
- 04_original.avi
- 04_forged.avi
- 05_original.avi
- 05_forged.avi
- 06_original.avi
- 06_forged.avi
- 07_original.avi
- 07_forged.avi
- 08_original.avi
- 08_forged.avi
- 09_original.avi
- 09_forged.avi
- 10_original.avi
- 10_forged.avi

Dal sito Xiph.org nella sezione *Derf's collection*[21]:

- flower_garden_422_ntsc.y4m
- highway_cif.y4m
- aspen_1080p.y4m
- blue_sky_1080p25.y4m
- Johnny_1280x720_60.y4m
- KristenAndSara_1280x720_60.y4m

- vidyo3_720p_60fps.y4m
- ice_4cif.y4m
- 720p50_parkrun_ter.y4m

I primi sette video in formato *raw* (cioè che non hanno subito compressione lossy) sono stati modificati con la tecnica copy-paste e ricodificati con il codec H.264 ottenendo i seguenti video:

- forged_flower_garden.264
- forged_highway.264
- forged_highway_qp20.264
- forged_aspen.264
- forged_aspen_qp20.264
- forged_blue_sky.264
- forged_blue_sky_qp20.264
- forged_johnny2.264
- forged_johnny2_qp20.264
- forged_KristenAndSara.264
- forged_KristenAndSara2.264
- forged_KristenAndSara_big.264
- forged_KristenAndSara_big2.264
- forged_vidyo3.264
- forged_vidyo3_qp20.264

Con la stessa procedura, i restanti due video provenienti da Xiph.org assieme ad altri quattro video tra quelli precedenti sono stati modificati con la tecnica copy-move, ottenendo quindi i seguenti video:

- forged_gardenMove_qp20.264

- forged_blue_skyMove_qp20.264
- forged_aspenMove.264
- forged_KASMove_qp20.264
- forged_parkrunMove.264
- forged_iceMove.264

Appendice B

Figure risultati copy-paste

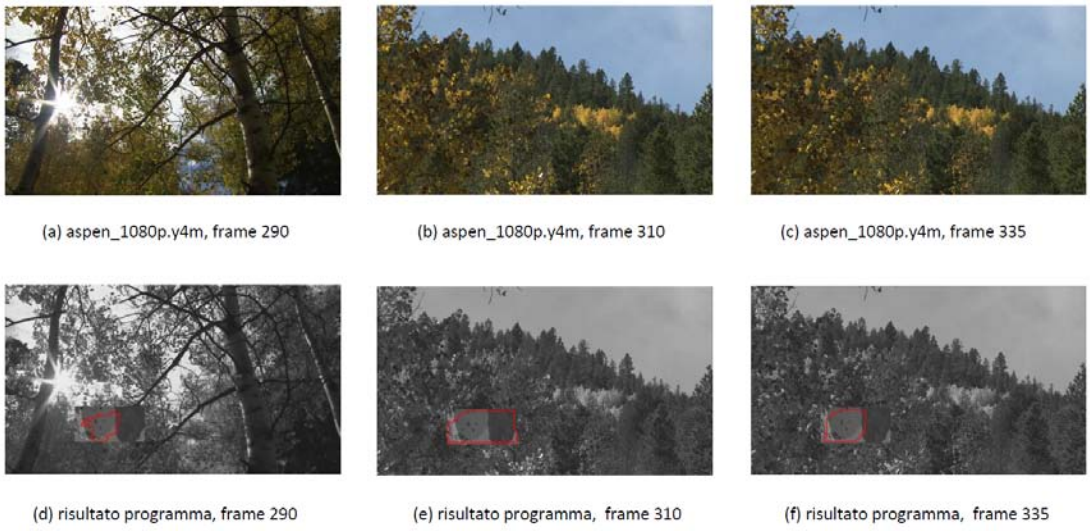


Figura B.1: aspen_1080p.y4m e forged_aspen_qp20.264

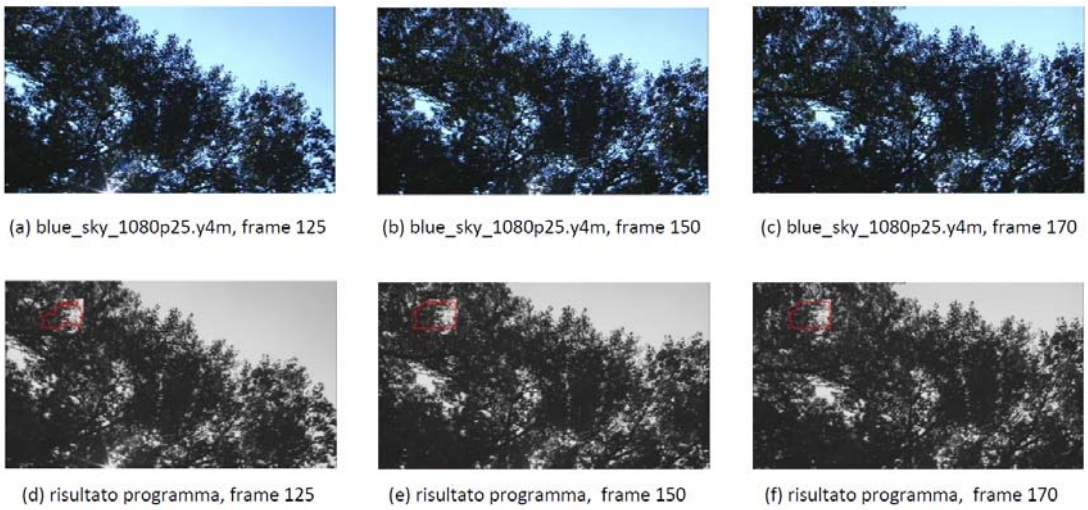


Figura B.2: blue_sky_1080p25.y4m e forged_blue_sky_qp20.264

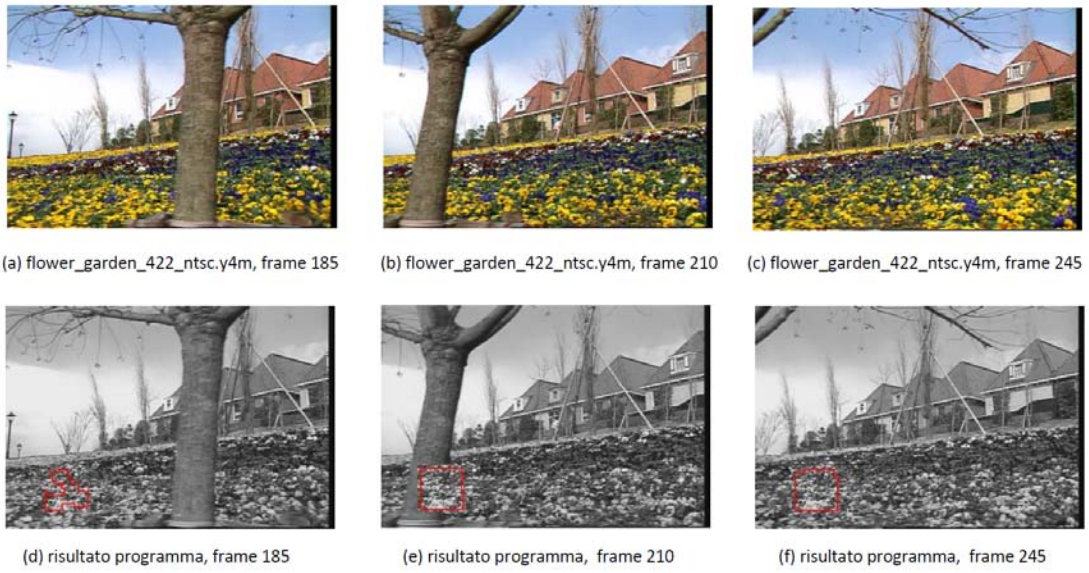


Figura B.3: flower_garden_422_ntsc.y4m e forged_flower_garden.264

Figura B.4: highway_cif.y4m e forged_highway_qp20.264

Figura B.5: Johnny_1280x720_60.y4m e forged_johnny2_qp20.264

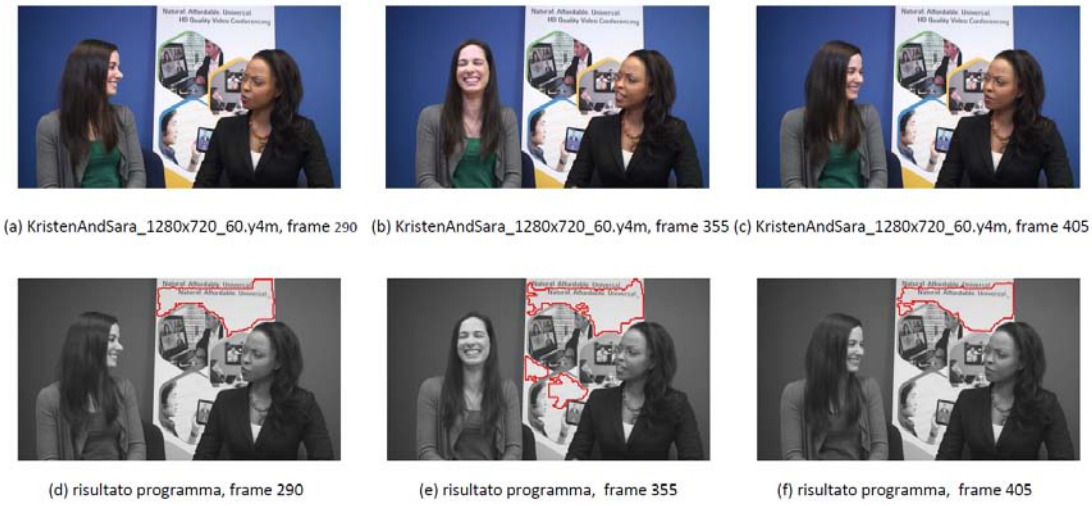


Figura B.6: KristenAndSara_1280x720_60.y4m e forged_KristenAndSara2.264

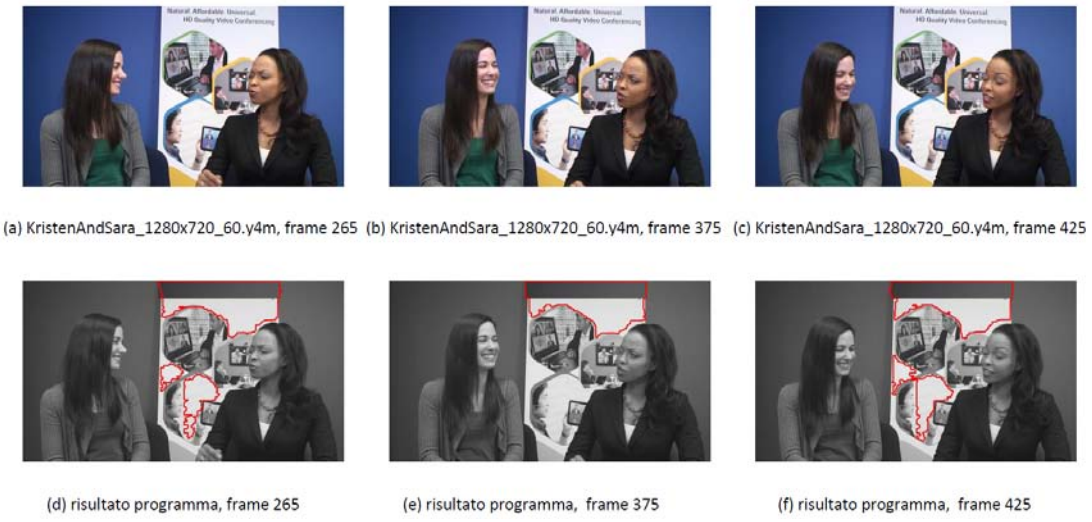


Figura B.7: KristenAndSara_1280x720_60.y4m e forged_KristenAndSara_big2.264

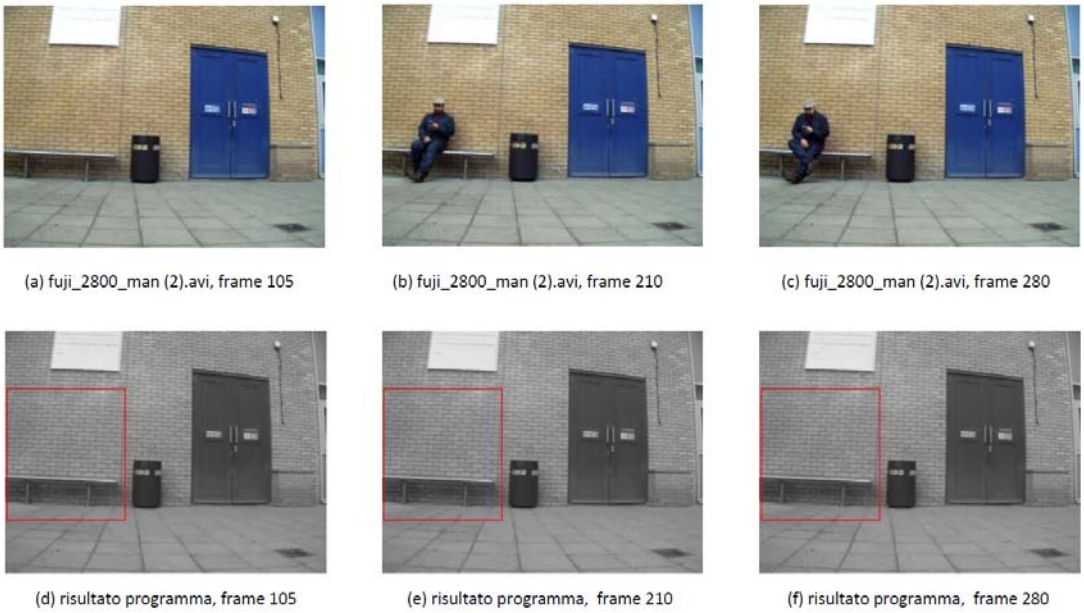


Figura B.8: fuji_2800_man (2).AVI e forged_fuji_2800_man (2).avi

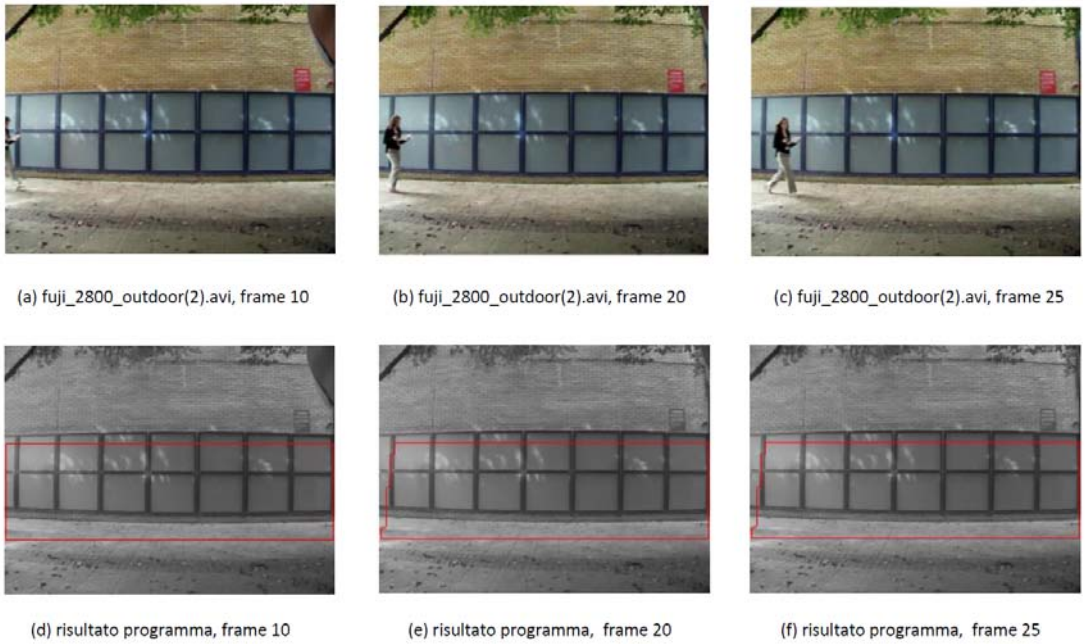


Figura B.9: fuji_2800_outdoor(2).AVI e forged_fuji_2800_outdoor(2).avi

(a) fuji_2800_road(1).avi, frame 5

(b) risultato programma, frame 5

Figura B.10: fuji_2800_road(1).AVI e forged_fuji_2800_road(1).avi

(a) fuji_2800_street(1).avi, frame 15

(b) fuji_2800_street(1).avi, frame 70

(c) fuji_2800_street(1).avi, frame 175

(d) risultato programma, frame 15

(e) risultato programma, frame 70

(f) risultato programma, frame 175

Figura B.11: fuji_2800_street.AVI e forged_fuji_2800_street.avi

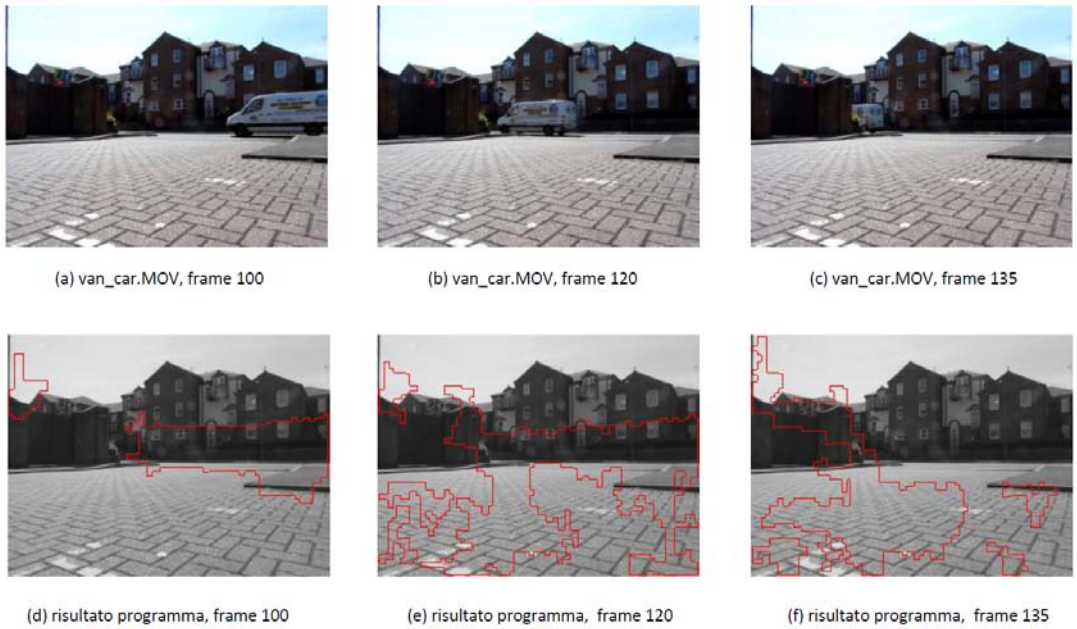


Figura B.12: van_car.MOV e can_220_van_car_Forge.MOV

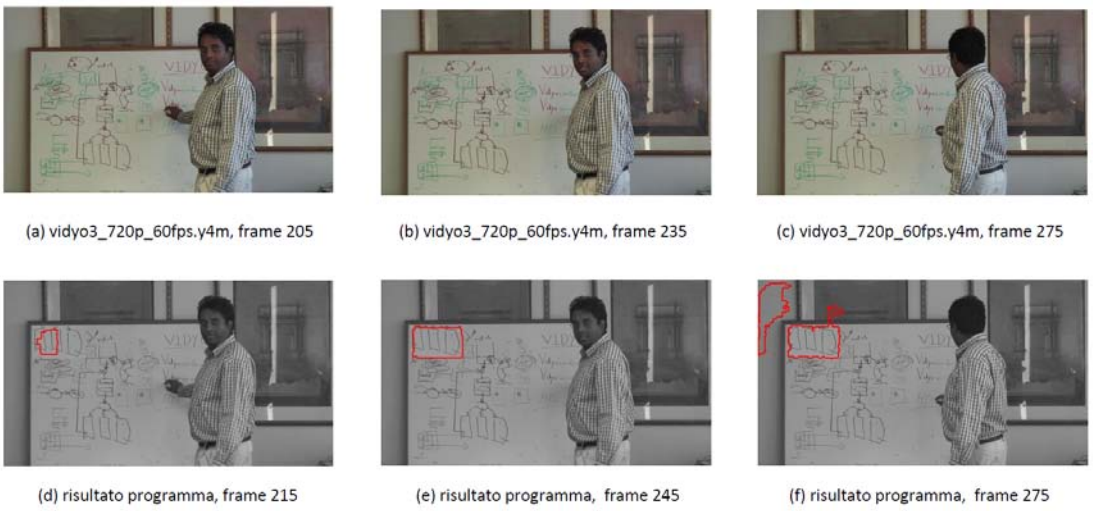


Figura B.13: vidyo3_720p_60fps.y4m e forged_vidyo3_qp20.264

Appendice C

Figure risultati copy-move

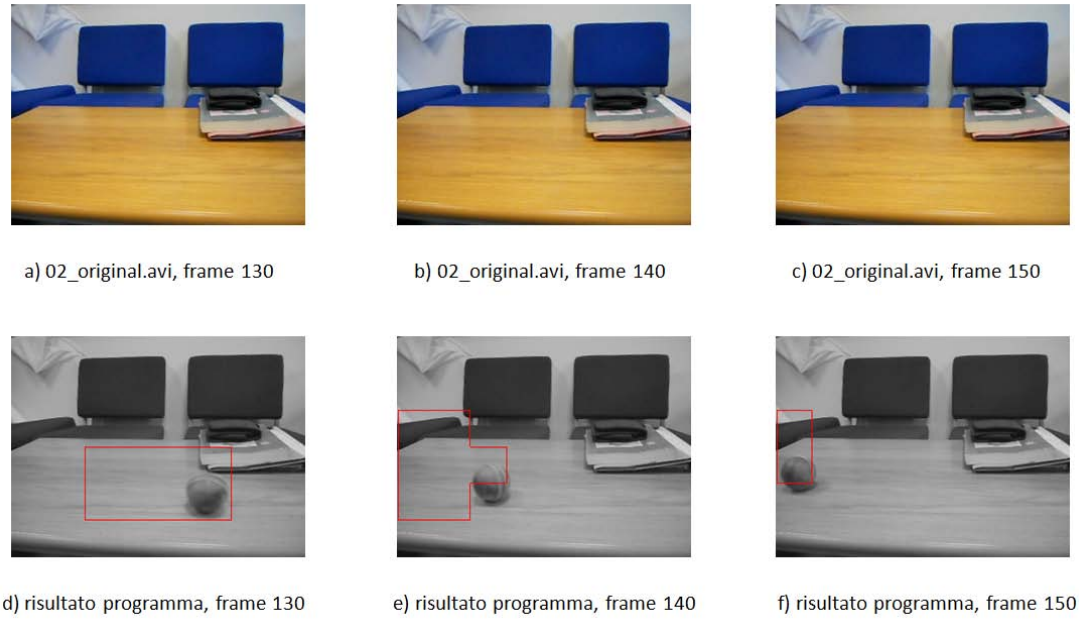
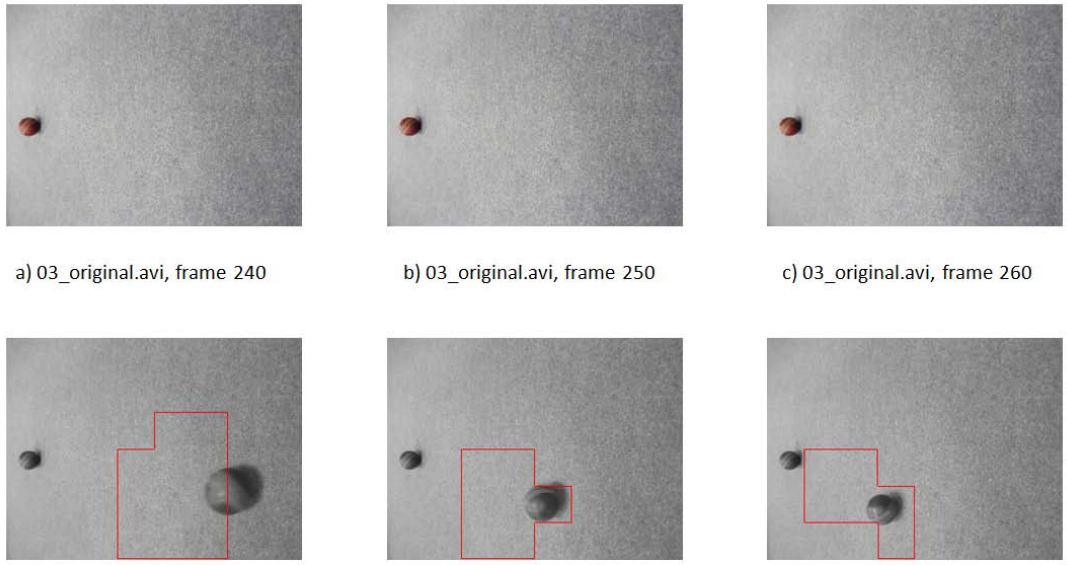


Figura C.1: 02_original.avi e 02_forged.avi



a) 03_original.avi, frame 240 b) 03_original.avi, frame 250 c) 03_original.avi, frame 260

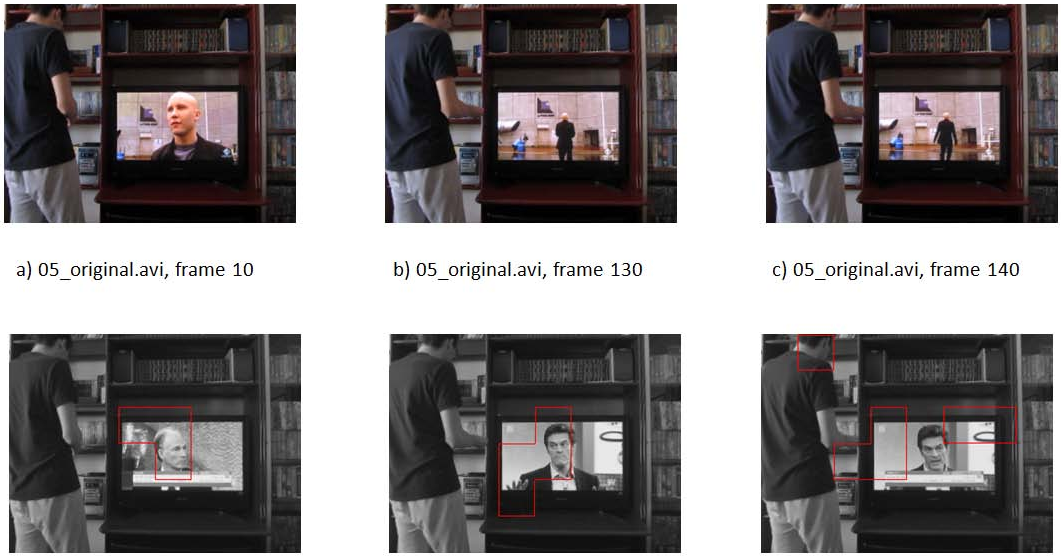
d) risultato programma, frame 240 e) risultato programma, frame 250 f) risultato programma, frame 260

Figura C.2: 03_original.avi e 03_forged.avi

a) 04_original.avi, frame 70 b) 04_original.avi, frame 80 c) 04_original.avi, frame 90

d) risultato programma, frame 70 e) risultato programma, frame 80 f) risultato programma, frame 90

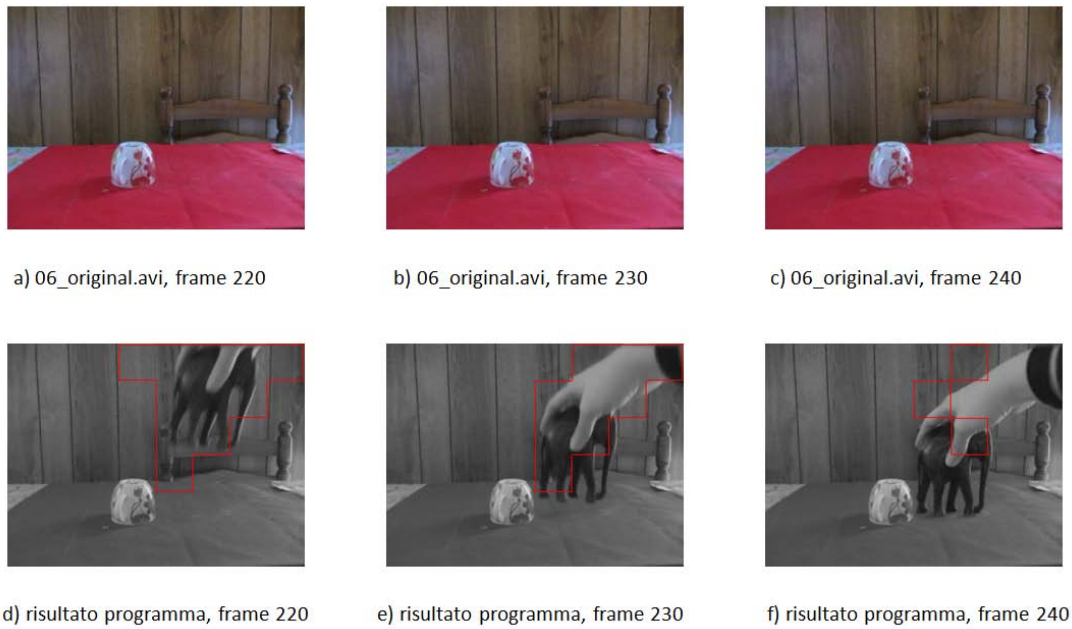
Figura C.3: 04_original.avi e 04_forged.avi



a) 05_original.avi, frame 10 b) 05_original.avi, frame 130 c) 05_original.avi, frame 140

d) risultato programma, frame 10 e) risultato programma, frame 130 f) risultato programma, frame 140

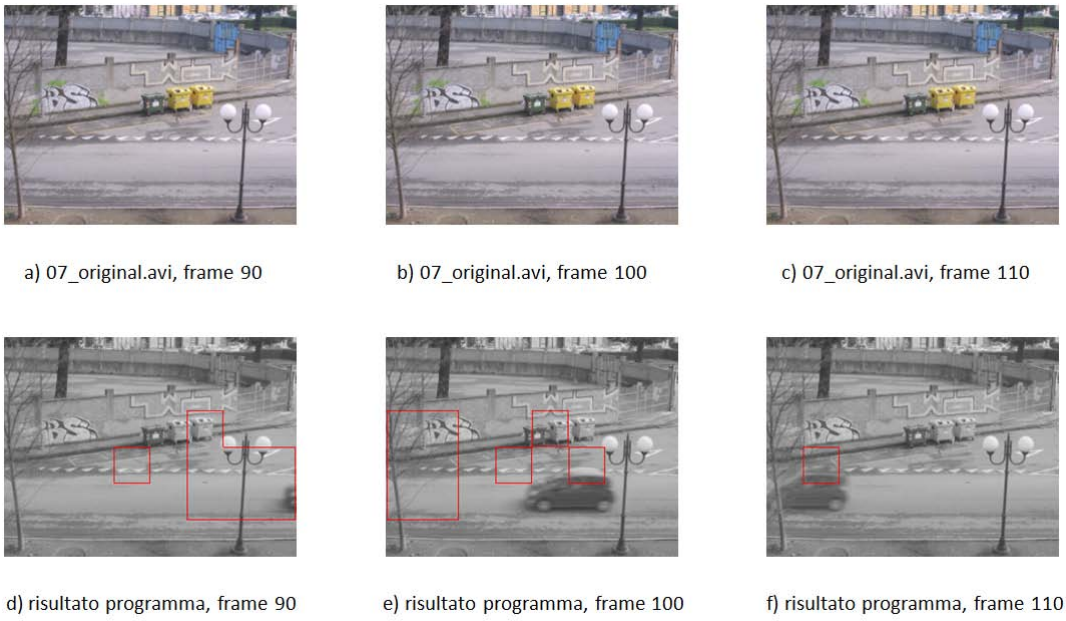
Figura C.4: 05_original.avi e 05_forged.avi



a) 06_original.avi, frame 220 b) 06_original.avi, frame 230 c) 06_original.avi, frame 240

d) risultato programma, frame 220 e) risultato programma, frame 230 f) risultato programma, frame 240

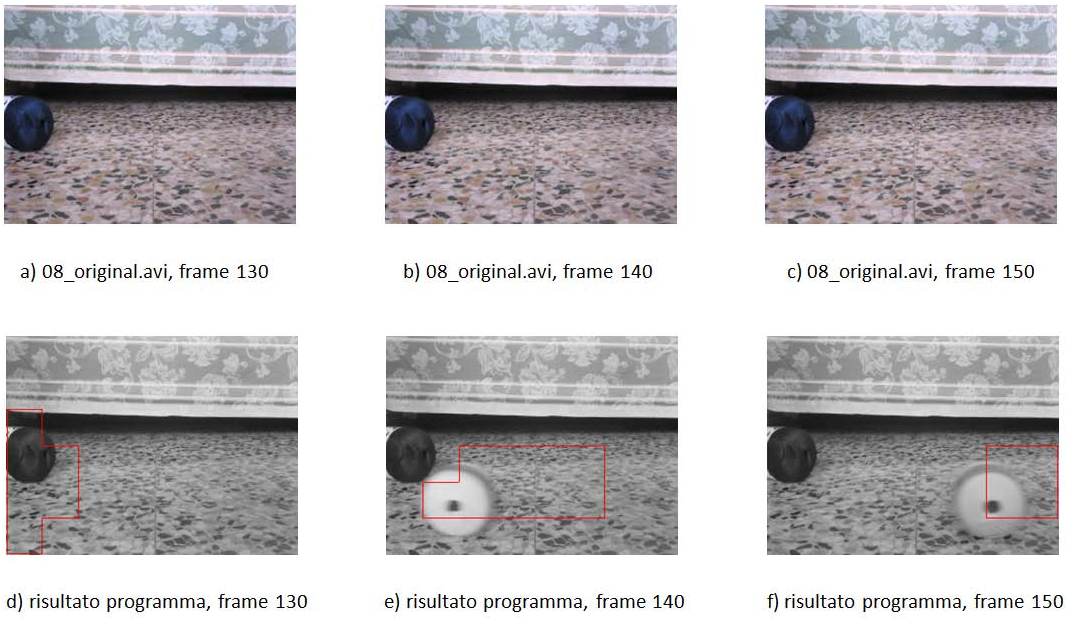
Figura C.5: 06_original.avi e 06_forged.avi



a) 07_original.avi, frame 90 b) 07_original.avi, frame 100 c) 07_original.avi, frame 110

d) risultato programma, frame 90 e) risultato programma, frame 100 f) risultato programma, frame 110

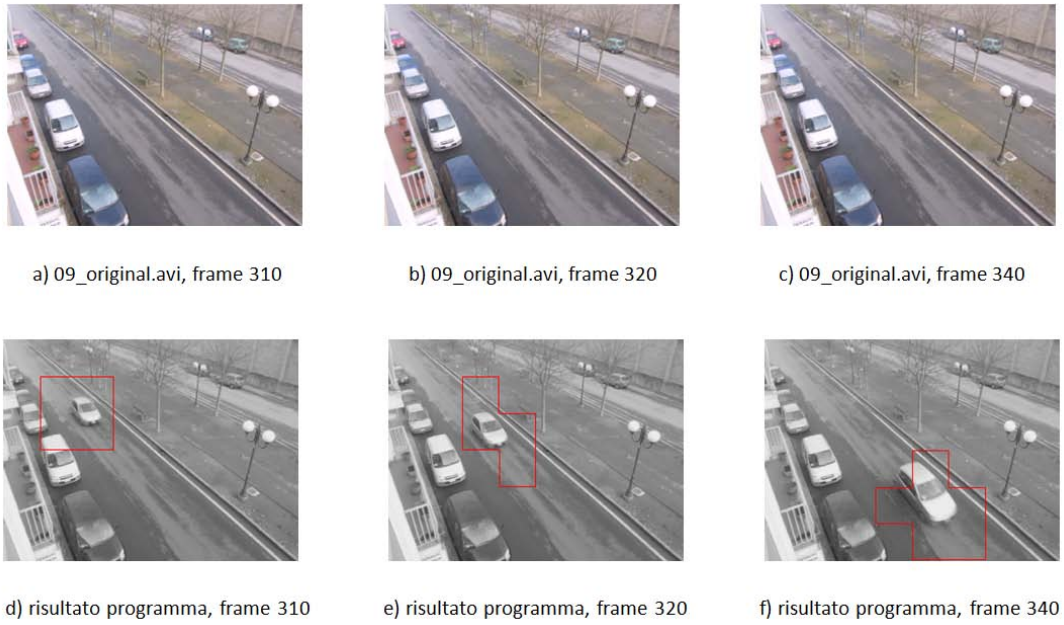
Figura C.6: 07_original.avi e 07_forged.avi



a) 08_original.avi, frame 130 b) 08_original.avi, frame 140 c) 08_original.avi, frame 150

d) risultato programma, frame 130 e) risultato programma, frame 140 f) risultato programma, frame 150

Figura C.7: 08_original.avi e 08_forged.avi



a) 09_original.avi, frame 310 b) 09_original.avi, frame 320 c) 09_original.avi, frame 340

d) risultato programma, frame 310 e) risultato programma, frame 320 f) risultato programma, frame 340

Figura C.8: 09_original.avi e 09_forged.avi

a) 10_original.avi, frame 110 b) 10_original.avi, frame 120 c) 10_original.avi, frame 130

d) risultato programma, frame 110 e) risultato programma, frame 120 f) risultato programma, frame 130

Figura C.9: 10_original.avi e 10_forged.avi

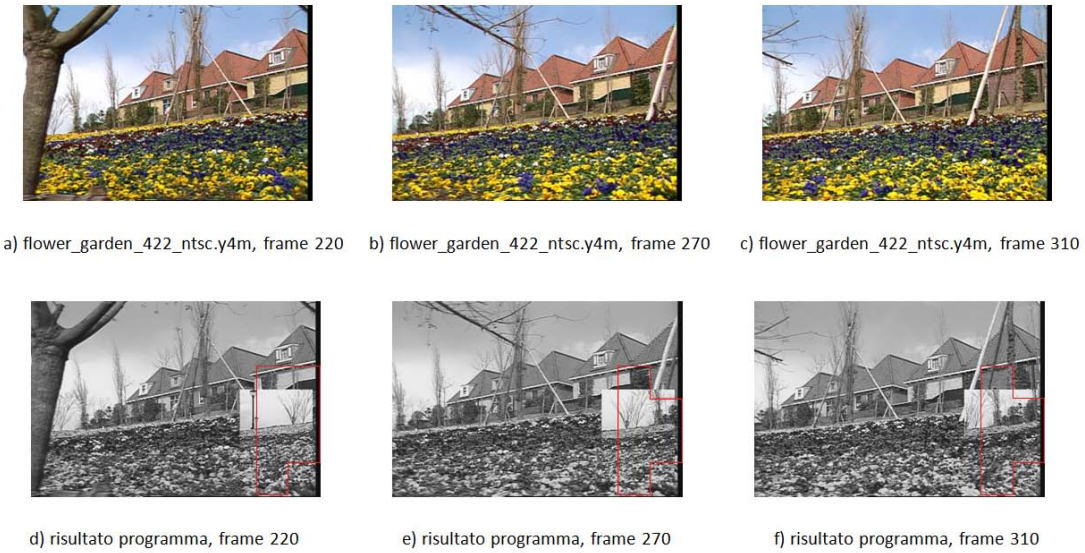


Figura C.10: flower_garden_422_ntsc.y4m e forged_gardenMove.264

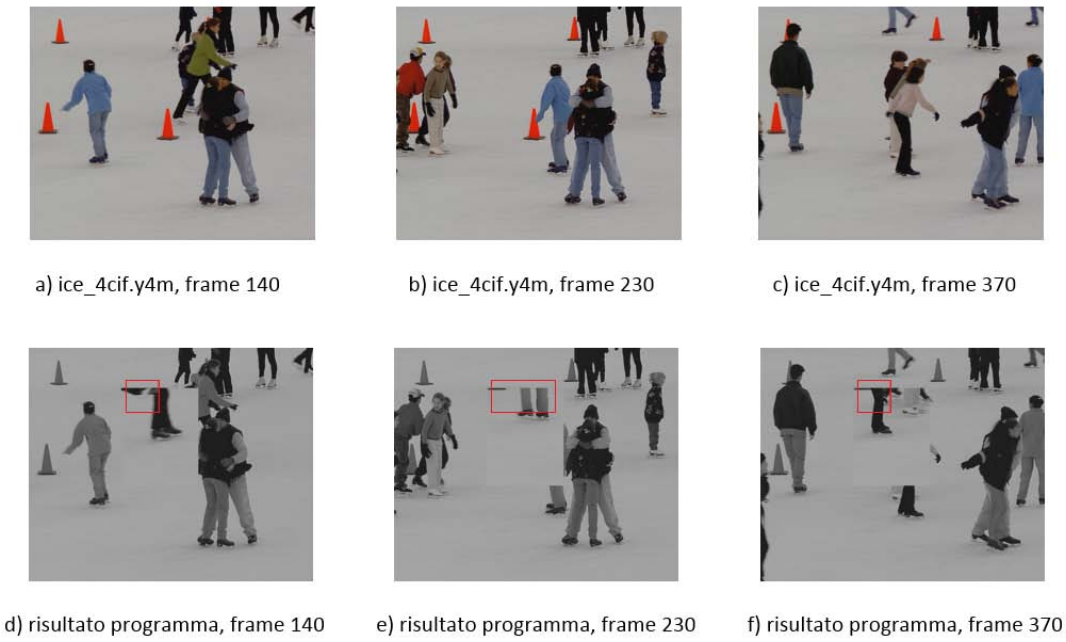


Figura C.11: ice_4cif.y4m e forged_iceMove.264

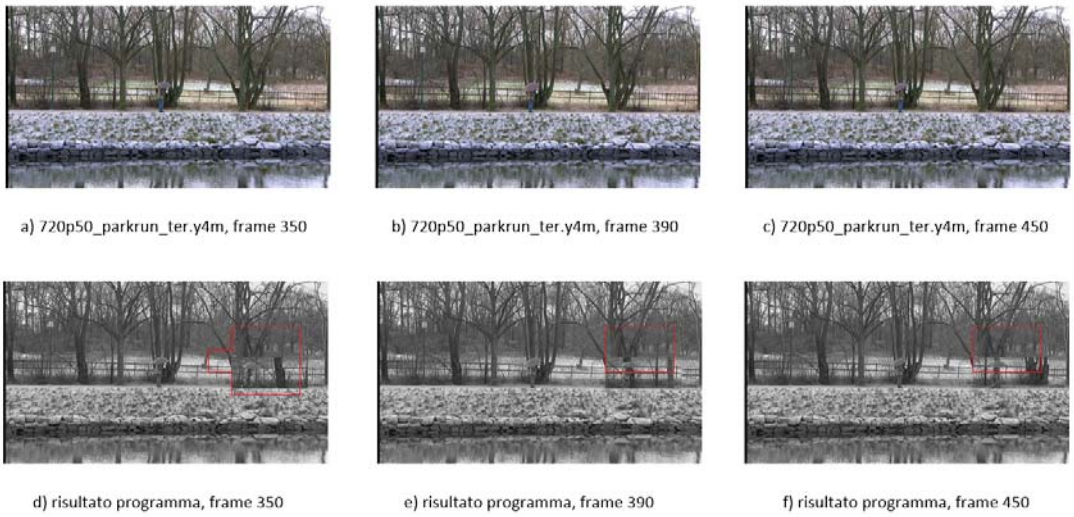


Figura C.12: 720p50_parkrun_ter.y4m e forged_parkrunMove.264

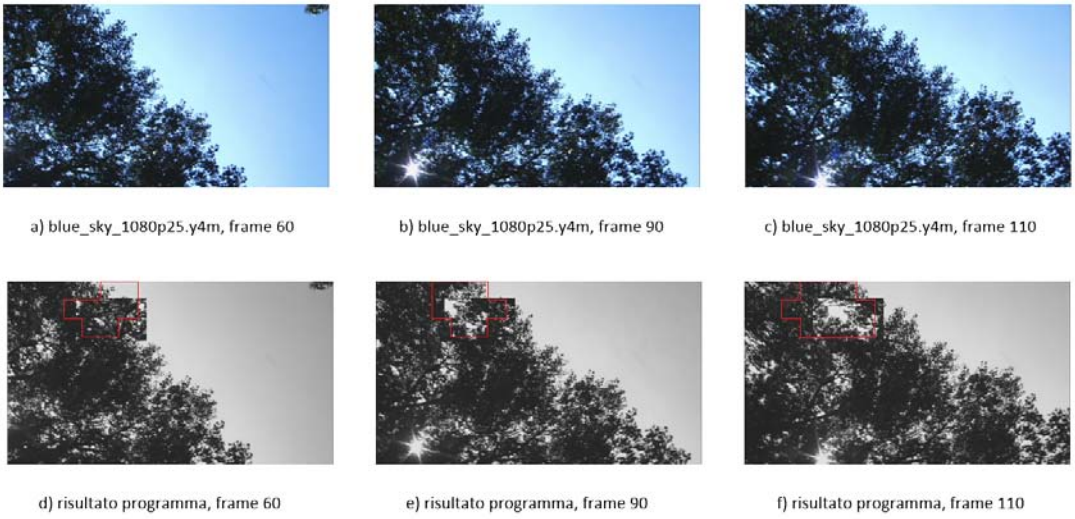


Figura C.13: blue_sky_1080p25.y4m e forged_blue_skyMove.264

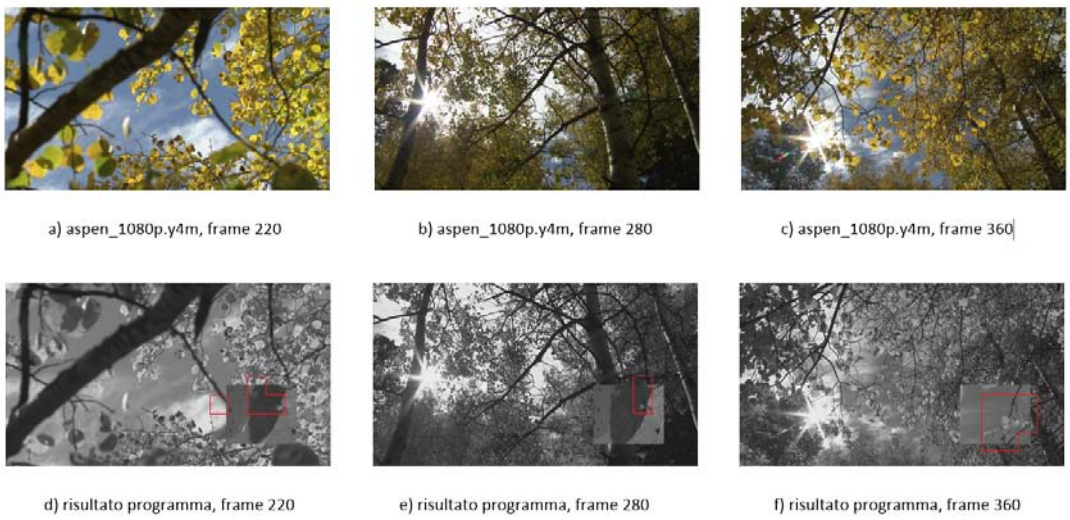


Figura C.14: aspen_1080p.y4m e forged_aspenMove.264

Bibliografia

[1] Paolo Bestagini, Simone Milani, Marco Tangliasacchi, Stefano Tubaro, "Local Tampering detection in video sequences", *Proc.Multimedia Signal Processing (MMSP)*, IEEE 15th International Workshop, 2013, pp.488-493.

[2] www.huffingtonpost.com/entry/fake-police-cam-footage_us_55c23782e4b0f7f0bebb2b5c .

[3] www.digital-forensics.it/computer-forensics .

[4] Ribnick E., Atev S., Masoud O., Papanikolopoulos N., Voyles R., "Real-Time Detection of Camera Tampering," in *Video and Signal Based Surveillance*, 2006. AVSS '06. IEEE International Conference on, vol., no., pp.10-10, Nov. 2006.

[5] Sowmya K.N., H.R. Chennamma, "A survey on video forgery detection", *International Journal of Computer Engineering and Applications*, Volume IX, Issue II, February 2015.

[6] http://searchsecurity.techtarget.com/definition/digital-signature .

[7] Peng Yin and Hong H. Yu, "Classification of video tampering methods and countermeasures using digital watermarking", *Proc. SPIE 4518*, Multimedia Systems and Applications IV, 239 , November 12, 2001.

[8] Dawen Xu, Rangding Wang, Jicheng Wang, "A novel watermarking scheme for H.264/AVC video authentication", *Signal Processing: Image Communication*, Volume 26, Issue 6, July 2011, Pages 267-279.

[9] Ching-Yung Lin, Shih-Fu Chang, "Generating Robust Digital Signature for Image/Video Authentication", in *Multimedia and Security Workshop at ACM Multimedia*, 1998.

[10] S. Milani, M. Fontani, P. Bestagini, M. Barni, A. Piva, M. Tagliasacchi and S. Tubaro, "An overview on video forensics", *APSIPA Transactions on Signal and Information Processing*, Volume 1, e2, 2012.

[11] Mondaini N., Caldelli R., Piva A., Barni M., Cappellini V., "Detection of malevolent changes in digital video for forensic applications", in *Proc. of SPIE, Security, Steganography, and Watermarking of Multimedia Contents IX*, E. J. D. III and P. W. Wong, eds., vol. 6505, no. 1, SPIE, 2007, 65050T.

[12] Wang, W., Farid, H., "Exposing digital forgeries in video by detecting double MPEG compression", in *MM&Sec*, S. Voloshynovskiy, J. Dittmann, and J. J. Fridrich, eds., ACM, 2006, 37?47.

[13] Tanfeng Sun, Wan Wang, Xinghao Jiang, "Exposing video forgeries by detecting MPEG double compression," in *Acoustics, Speech and Signal Processing (ICASSP)*, 2012 IEEE International Conference on , vol., no., pp.1389-1392, 25-30 March 2012.

[14] Milani S., Bestagini P., Tagliasacchi M., Tubaro S., "Multiple compression detection for video sequences," in *Multimedia Signal Processing (MMSP)*, 2012 IEEE 14th International Workshop on , vol., no., pp.112-117, 17-19 Sept. 2012.

[15] Subramanyam A.V., Emmanuel S., "Video forgery detection using HOG features and compression properties," in *Multimedia Signal Processing (MMSP)*, 2012 IEEE 14th International Workshop on, vol., no., pp.89-94, 17-19 Sept. 2012.

[16] Chen Richao, Yang Gaobo, Zhu Ningbo, "Detection of object-based manipulation by the statistical features of object contour", *Forensic Science International*, Volume 236, March 2014, Pages 164-169.

[17] Qiong Dong, Gaobo Yang, Ningbo Zhu, "A MCEA based passive forensics scheme for detecting frame based video tampering", *Elsevier*, 2012, vol.9, pp.151-159.

[18] Chih-Chung Hsu, Tzu-Yi Hung, Lin, Chia-Wen, Chiou-Ting Hsu, "Video forgery detection using correlation of noise residue", in *Multimedia Signal*

Processing, 2008 IEEE 10th Workshop on , vol., no., pp.170-174, 8-10 Oct. 2008.

[19] Weihong Wang, Hany Farid, "Exposing digital forgeries in video by detecting duplication", in *Proceedings of the 9th workshop on Multimedia & security (MM&Sec '07)*, ACM, New York, NY, USA, 35-42.

[20] http://sulfa.cs.surrey.ac.uk/forged.php .

[21] https://media.xiph.org/video/derf/ .