



**Source Device Attribution For Digital
Videos**

Emmanuel KIEGAING KOUOKAM



T.C.
ULUDAĞ UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

SOURCE DEVICE ATTRIBUTION FOR DIGITAL VIDEOS

Emmanuel KIEGAING KOUOKAM

Assoc. Prof. Dr. Ahmet Emir DİRİK
(Supervisor)

MASTER OF SCIENCE THESIS
DEPARTMENT OF ELECTRONIC ENGINEERING

BURSA-2018

TEZ ONAYI

Emmanuel Kiegaing Kouokam tarafından hazırlanan "SAYISAL VİDEOLARDA KAYNAK CİHAZ TANIMA" adlı tez çalışması aşağıdaki jüri tarafından oy birliği/oy çokluğu ile Uludağ Üniversitesi Fen Bilimleri Enstitüsü Elektronik Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Danışman : Doç. Dr. Ahmet Emir Dirik
Uludağ Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü



Başkan : Doç. Dr. Ahmet Emir Dirik
Uludağ Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü



Üye : Doç. Dr. Fatih Çavdur
Uludağ Üniversitesi
Mühendislik Fakültesi
Endüstri Mühendisliği Bölümü



Üye : Dr. Öğr. Üyesi. Murat Türe
Bursa Teknik Üniversitesi
Mühendislik ve Doğa Bilimleri Fakültesi
Mekatronik Mühendisliği Bölümü



Yukarıdaki sonucu onaylarım



Prof. Dr. Ali BAYRAM


Enstitü Müdürü

29.8.2016 (Tarih)

U.Ü. Fen Bilimleri Enstitüsü, tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmada;

- tez içindeki bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- görsel, işitsel ve yazılı tüm bilgi ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu,
- başkalarının eserlerinden yararlanılması durumunda ilgili eserlere bilimsel normlara uygun olarak atıfta bulunduğumu,
- atıfta bulunduğum eserlerin tümünü kaynak olarak gösterdiğimi,
- kullanılan verilerde herhangi bir tahrifat yapmadığımı,
- ve bu tezin herhangi bir bölümünü bu üniversite veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.


Emmanuel Kiegaing
07/08/2018

ÖZET

Yüksek Lisans Tezi

SAYISAL VİDEOLARDA KAYNAK CİHAZ TANIMA

Emmanuel Kiegaing Kouokam

Uludağ Üniversitesi

Fen Bilimleri Enstitüsü

Elektronik Mühendisliği Anabilim Dalı

Danışman: Doç. Dr. Ahmet Emir DİRİK

Kaynak cihaz tespiti, dijital resim veya videolardan çeşitli adli bilişim yöntemleri kullanılarak cihaz tanımayı amaçlayan sayısal adli bilişim içerisindeki ana konulardan birisidir. Adli bilişimde genellikle fotoğraf veya videonun çekilmesi esnasında cihazın donanımsal olarak resimler üzerinde bıraktığı izler kullanılarak tespit işlemleri yapılmaktadır. Bu kalıntı izleri içerisinde sensörlerden kaynaklı kalan PRNU izleri her cihazda kendine özgü olmasından ötürü kaynak cihaz tanımada kullanılan en önemli izlerdir. Bu çalışmada, video sıkıştırmanın, çerçeveler üzerindeki PRNU gürültüsü üzerindeki etkisi irdelenmiş ve oldukça yoğun sıkıştırılmış H.264/AVC videolarında kaynak cihaz tanıma için yeni bir yöntem önerilmiştir. Önerilen yöntem stabilize edilmemiş videolar için oldukça geniş bir veri setinde test edilmiş ve yoğun sıkıştırılmış videolarda bile yüksek yüzdede başarı elde edilmiştir. Ayrıca stabilize edilmiş videolarda kaynak cihaz tanıma yapılabilmesi için yeni bir metot önerilmiştir. Bu metot yoğun sıkıştırılmış ve stabilize edilmiş küçük bir veri setinde test edilmiştir. Elde edilen sonuçlar, önerilen yöntemin oldukça başarılı ve etkili olduğunu göstermektedir.

Anahtar Kelimeler: Sayısal adli bilişim, kaynak cihaz özelliği, PRNU, video sıkıştırma, dijital video stabilizasyon.

2018, ix + 89 sayfa

ABSTRACT

Master of Science Thesis

SOURCE DEVICE ATTRIBUTION FOR DIGITAL VIDEOS

Emmanuel Kiegaing Kouokam

Uludağ University

Graduate School of Natural and Applied Sciences

Department of Electronic Engineering

Supervisor: Assoc. Prof. Dr. Ahmet Emir Dirik

Source device attribution is one of the main tasks of multimedia forensics which aims to identify the device from which a digital image or video originates using blind forensic techniques. Forensics generally rely on unique artifacts created on acquired images or videos during the acquisition process. Photo-Response Non-Uniformity (PRNU) is one of the most important sensor artifacts used in source device attribution due to its unique, random, and robust nature. In this research, we study the effect of video compression on the PRNU noise in video frames and propose new techniques to perform an accurate source device attribution of highly compressed H.264/AVC videos. The proposed scheme for non-stabilized videos was tested on a large set of videos and achieved a high accuracy even on highly compressed videos. We also propose a new scheme for source device attribution of digitally-stabilized video; this scheme was tested on a small set of digitally-stabilized and highly-compressed videos; the results obtained show the effectiveness of the proposed scheme.

Key Words: Video forensics, source device attribution, Photo-Response Non-Uniformity, video compression, digital video stabilization.

2018, ix + 89 pages

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to all those who have contributed to the realization of this research work: my supervisor Assoc. Prof. Dr. Ahmet Emir Dirik for accepting me in his research team, for his guidance, and for providing all the facilities that were necessary to conduct this research in good conditions; Saffet Vatansever for providing me with his Matlab source code for image-based to frame-based fingerprint matching; Dr. Ahmet Karaküçük for his advice and help; all the personnel of Uludağ University; the Turkish government for giving me the opportunity to study at Uludağ university; Arnaud Cedric Kamkoum and Ibrahim Mayanja for proofreading this text; my family and friends for their assistance and moral support.



Emmanuel Kiegajing

07/01/2018

TABLE OF CONTENT

| | Page |
|---|-------------|
| ÖZET | i |
| ABSTRACT | ii |
| ACKNOWLEDGEMENTS | iii |
| ABBREVIATIONS AND SYMBOLS | vi |
| LIST OF FIGURES | vi |
| LIST OF TABLES | viii |
| 1 INTRODUCTION | 1 |
| 2 THEORETICAL FOUNDATIONS | 7 |
| 2.1 PRNU of digital camera sensors and its use for image source attribution | 7 |
| 2.1.1 The image acquisition process and its artifacts | 7 |
| 2.1.2 Camera sensor output model | 11 |
| 2.1.3 Camera sensor fingerprint estimation | 12 |
| 2.1.4 PRNU-Based image source identification | 14 |
| 2.2 The H.264/AVC video compression standard basics | 16 |
| 2.2.1 Block processing | 17 |
| 2.2.2 Prediction in H.264 | 19 |
| 2.2.3 Transform and Quantization in H.264 | 22 |
| 2.2.4 H.264/AVC profiles | 26 |
| 2.3 Effects of video compression on the PRNU noise in video frames | 26 |
| 2.3.1 Open source softwares for H.264/AVC video encoding/decoding | 29 |
| 2.4 Overview of digital video stabilization | 30 |
| 2.4.1 Motion estimation | 31 |
| 2.4.2 Motion smoothing | 33 |
| 2.4.3 Motion compensation | 33 |

| | | |
|-------|---|----|
| 3 | MATERIAL AND METHOD | 35 |
| 3.1 | Source device attribution for non-stabilized videos | 35 |
| 3.1.1 | Frame-based approach | 35 |
| 3.1.2 | Block-based approach | 36 |
| 3.2 | Source device attribution for stabilized videos | 39 |
| 3.2.1 | Estimation of sensor crop and scale parameters that match an image-based to a frame-based fingerprint | 40 |
| 3.2.2 | Proposed scheme for stabilized video frames noise registration | 43 |
| 3.2.3 | Warped frames noise aggregation | 53 |
| 4 | RESULTS | 56 |
| 4.1 | The VISION dataset | 56 |
| 4.1.1 | VISION dataset overview | 56 |
| 4.1.2 | Properties of videos used in our experiments | 57 |
| 4.2 | Source device attribution for non-stabilized videos | 58 |
| 4.2.1 | Source device attribution for native videos | 59 |
| 4.2.2 | Source device attribution for YouTube videos with fingerprints estimated from the native videos | 61 |
| 4.2.3 | Source device attribution for YouTube videos with reference fingerprints esti- mated from YouTube videos | 62 |
| 4.3 | Source device attribution for digitally stabilized videos | 74 |
| 4.3.1 | Source device identification for in-camera-stabilized videos | 75 |
| 4.3.2 | Source device identification for videos stabilized with <i>FFmpeg</i> | 78 |
| 4.3.3 | Source device identification for videos stabilized with YouTube stabilizer | 80 |
| 5 | DISCUSSION AND CONCLUSION | 83 |
| | REFERENCES | 85 |
| | CURRICULUM VITAE | 91 |

ABBREVIATIONS AND SYMBOLS

| Symbol | Meaning |
|----------------------------------|--|
| I | Intensity of an image |
| K | PRNU matrix |
| $\hat{\mathbf{K}}$ | Fingerprint matrix (estimate of K) |
| M_k | Mask of frame k |
| \mathbf{Wb}_{cur} | PRNU noise in a block which is to be encoded |
| \mathbf{Wb}_δ | PRNU noise in the block' prediction residual |
| \mathbf{W}_k | PRNU noise estimated from frame/image k |
| $\widetilde{\mathbf{Wb}}_\delta$ | PRNU noise in a block after transform and quantization |
| $\widetilde{\mathbf{Wb}}_{cur}$ | PRNU noise in the decoded block |
| $\widetilde{\mathbf{Wb}}_{ref}$ | PRNU noise in the reference block |

| Abbreviation | Meaning |
|--------------|---|
| ADC | Analog to Digital Converter |
| AUC | Area Under Curve |
| AVC | Advanced Video Compression |
| BB | Block-Based |
| CCD | Charged Coupled Device |
| CFA | Color Filter Array |
| CMOS | Complementary Metal Oxide Semiconductor |
| CPU | Central Processing Unit |
| DCT | Discrete Cosine Transform |
| DCT-AC | DCT-Alternative Component |
| DCT-DC | DCT-Direct Component |
| FB | Frame-Based |
| FFT | Fast Fourier Transform |
| FP | False Positive |
| GOP | Group Of Pictures |
| GPU | Graphics Processing Unit |
| MPI | Message Passing Interface |
| NCC | Normalized Cross-Correlation |
| PCE | Peak-to-Correlation Energy |
| PRNU | Photo-Response Non-Uniformity |
| ROC | Receiver Operating Characteristic |
| TP | True Positive |
| 720p | 1280 × 720 progressive video |
| 1080p | 1920 × 1080 progressive video |

LIST OF FIGURES

| | Page |
|--|-------------|
| Figure 2.1 Image acquisition pipeline | 7 |
| Figure 2.2 Components of the imaging sensor pattern noise | 9 |
| Figure 2.3 Influence of image content on the estimated noise residual (pictures are from the VISION dataset) | 13 |
| Figure 2.4 Reference and linear patterns for an exemplar of Samsung Galaxy S3 Mini smartphone camera | 14 |
| Figure 2.5 Principle of PRNU-based source device attribution for digital images | 15 |
| Figure 2.6 Block diagram of an H.264/AVC encoder/decoder (Marpe et al. 2006) | 17 |
| Figure 2.7 4:2:0 color sub-sampling scheme (Richardson and E. 2010) | 18 |
| Figure 2.8 Example of partition choices for a P frame (Richardson and E. 2010) | 18 |
| Figure 2.9 Principle of prediction residue computation in inter-frame prediction (Richardson and E. 2010) | 19 |
| Figure 2.10 Principle of intra-frame and inter-frame predictions (Richardson and E. 2010) | 20 |
| Figure 2.11 Intra-prediction modes for 16×16 and 4×4 sized blocks (Richardson and E. 2010) | 20 |
| Figure 2.12 Macroblock partitioning in inter-frame prediction (Richardson and E. 2010) | 21 |
| Figure 2.13 Zigzag addressing : DCT coefficients appear in increasing frequencies (Richardson and E. 2010) | 23 |
| Figure 2.14 Macroblock Luma direct and inverse transform and quantization with a 4 × 4 DCT transform core (Richardson and E. 2010) | 24 |
| Figure 2.15 Macroblock Chroma direct and inverse transform and quantization with a 4 × 4 DCT transform core (Richardson and E. 2010) | 25 |
| Figure 2.16 Macroblock Luma direct and inverse transform and quantization with a 8 × 8 DCT transform core (Richardson and E. 2010) | 25 |
| Figure 2.17 H.264/AVC profiles (Marpe et al. 2006) | 26 |
| Figure 2.18 Operations applied on a block's PRNU noise during the block's encoding/decoding. | 27 |
| Figure 2.19 Video compression effect on the PRNU noise (the contrast has been adjusted to improve visibility) | 29 |
| Figure 2.20 Video stabilization on a sample input sequence (Hlmg et al. 2014) | 31 |
| Figure 3.1 PRNU fingerprints estimated from an input video | 36 |
| Figure 3.2 H.264-video Macroblocks parsing process flow | 37 |
| Figure 3.3 Example of a highly compressed I frame and its associated frame mask | 38 |
| Figure 3.4 Proposed scheme for source device identification of stabilized videos | 39 |
| Figure 3.5 Sensor resolution downscaling process in video acquisition | 40 |
| Figure 3.6 Proposed scheme for row-scaling and column-cropping parameters estimation | 41 |
| Figure 3.7 Row-scaling and column-cropping based parameters estimation : peaks represent searched parameters | 43 |

| | |
|---|----|
| Figure 3.8 Affine transform of a rectangle with its control points | 44 |
| Figure 3.9 Affine transform parameters estimation scheme | 47 |
| Figure 3.10 Computed maximum cross-correlation values at each step | 48 |
| Figure 3.11 Peak in the NCC between a reference fingerprint and a registered frame noise | 53 |
| Figure 4.1 Architecture of the VISION dataset (Dasara et al. 2017) | 57 |
| Figure 4.2 PCE distributions of 720p native videos matched to native videos | 62 |
| Figure 4.3 PCE distributions of 1080p native videos matched to native videos | 63 |
| Figure 4.4 PCE distributions of 720p native videos matched to YouTube videos | 65 |
| Figure 4.5 PCE distributions of 1080p native videos matched to YouTube videos | 66 |
| Figure 4.6 ROC curves of 720p native videos matched to YouTube videos | 67 |
| Figure 4.7 ROC curves of 1080p native videos matched to YouTube videos | 68 |
| Figure 4.8 PCE distributions of 720p YouTube videos matched to YouTube videos | 70 |
| Figure 4.9 PCE distributions of 1080p YouTube videos matched to YouTube videos | 71 |
| Figure 4.10 ROC curves of 720p YouTube videos matched to YouTube videos | 72 |
| Figure 4.11 ROC curves of 1080p native videos matched to YouTube videos | 73 |
| Figure 4.12 PCE values obtained when matching an image-based fingerprint to the PRNU noise extracted from I frames of a flat-content stabilized video (iPhone6) | 76 |
| Figure 4.13 PCE values of non-registered and registered PRNU frames noise | 76 |
| Figure 4.14 NCC corresponding to wrongly and correctly estimated affine transform | 77 |
| Figure 4.15 Cross-correlation before and after frame noise registration (iPhone6 test video) | 78 |
| Figure 4.16 PCE of non-registered and registered frames noise (video stabilized with FFmpeg) | 79 |
| Figure 4.17 Cross-correlation before and after frame noise registration (FFmpeg-deshake test video) | 79 |
| Figure 4.18 PCE of non-registered and registered frames noise (video stabilized with YouTubeStabilizer) | 81 |
| Figure 4.19 Cross-correlation before and after frame noise registration (YouTube-stabilized test video) | 81 |

LIST OF TABLES

| | Page |
|--|-------------|
| Table 2.1 Information included in the JM log file according to the log level | 30 |
| Table 2.2 Elementary operations involved in 2D-motion modeling | 32 |
| Table 2.3 Properties of 2D transformations used for motion modeling | 33 |
| Table 3.1 Combination of frame types for fingerprint estimation in the frame-based approach. | 35 |
| Table 4.1 The set of native videos used in our experiments | 57 |
| Table 4.2 Properties of native videos ([minimum average maximum]) | 58 |
| Table 4.3 Properties of YouTube videos ([minimum average maximum]) | 59 |
| Table 4.4 Number of correlation for matching and non-matching videos for each device | 60 |
| Table 4.5 AUC values of 720p native videos matched to native videos | 60 |
| Table 4.6 AUC values of 1080p native videos matched to native videos | 61 |
| Table 4.7 AUC values of 720p native videos matched to YouTube videos | 61 |
| Table 4.8 AUC values of 1080p native videos matched to YouTube videos | 64 |
| Table 4.9 AUC for each device when matching 720p native videos to YouTube videos | 64 |
| Table 4.10 AUC values for each device when matching 1080p native videos to YouTube videos | 64 |
| Table 4.11 AUC values of 720p YouTube videos matched to YouTube videos | 69 |
| Table 4.12 AUC values of 1080p YouTube videos matched to YouTube videos | 69 |
| Table 4.13 AUC values for each device when linking 720p YouTube videos | 69 |
| Table 4.14 AUC values for each device when linking 1080p YouTube videos | 74 |
| Table 4.15 Properties of computers in the cluster used for frames noise registration | 75 |
| Table 4.16 Processing time for the registration of the PRNU noise in one frame | 75 |
| Table 4.17 iPhone6 test video results | 77 |
| Table 4.18 Properties of fingerprint and test videos used in the FFmpeg experiment | 78 |
| Table 4.19 FFmpeg-deshake test video results | 79 |
| Table 4.20 Properties of fingerprint and test videos used in the Youtube-Stabilizer experiment | 80 |
| Table 4.21 YouTube-stabilized test video results | 80 |

1. INTRODUCTION

The use of digital media (digital images and videos) has increased steadily in the last decade to a point where it has become a major means for sharing information. The booming of the smartphone and tablet markets has made it possible for everyone to have all the necessary hardware and software to capture, reproduce, and share images and videos. Nowadays digital cameras can be found everywhere, in standard surveillance cameras, small sized hidden (or spy) cameras, as well as video recorders for cars and helmets. According to the Internet company Alexa (2018), YouTube and Facebook are currently the second and third most visited websites on the Internet, respectively. Iuliani et al. (2017) also reported that images and videos acquired with smartphones represent 85% of images and videos shared on the Internet.

Digital media are not only used for education, entertainment or information communication; they also constitute an effective means to perpetrate illegal acts such as movie piracy, terrorist propaganda, child pornography. The BBC (2016) reports a recent case where punishable acts were massively committed using digital cameras. The reported event took place in South Korea, where the practice of hiding spy cameras in women's toilets and later sharing the acquired videos on the Internet was popular. In some countries, it is possible to use an image or a video as evidence during a trial in a court of justice. All these facts raise the importance and necessity of finding methods to accurately identify the device from which a given image or video originates.

Identifying the origin of digital media can be the source of important information during an investigation. For instance, it can mean that the owner of the camera has witnessed the scene that was captured, that he has been at the place where the footage was taken. Device linking, which consists of determining whether two images or videos originate from the same device or not, can be used to trace a suspect (for instance a pedophile) sharing his offensive content through different social media accounts/platforms.

There are two approaches for digital media source device attribution: an active approach, called multimedia security; and a passive approach, called multimedia forensics. The active approach is based on techniques such as watermarking or steganography, which are invisible and fragile patterns added to acquired images or videos during the acquisition process and which can later be used to check the integrity and source of the media. Contrary to multimedia security, multimedia forensics does not use any active means and relies solely on artifacts in the acquired media created by components or algorithms used in the acquisition device. It is also important to mention that when acquiring an image or video, digital cameras also add to the file's headers data related to the acquired media (metadata), such as the acquisition device brand, time, location, and type of compression. However, these data are not reliable since they can easily be modified with simple software tools like Exiftool (Harvey 2018).

This research is in the scope of multimedia forensics and uses the camera sensor's Photo-Response Non-Uniformity (PRNU) (which adds a noise pattern to all the frames of videos acquired with a given device) to perform video source device attribution. Two quantities are necessary to perform PRNU-based source device attribution of digital images or videos: the camera fingerprint and the query image or video noise. By camera fingerprint, we mean a strong estimation of the PRNU noise pattern, which is generally obtained using a set of low textured images or video frames. The query image or video noise refers to the PRNU noise pattern estimated from a single image or a set of video frames from a random video. The challenge in PRNU-based source device attribution is about finding methods which provide a good estimate of the camera fingerprint and the image/video noise.

The idea of using CCD (Charge-Coupled Device) sensor imperfections to perform video source identification first originated from K. Kurosawa in late 1999. Kurosawa et al. (1999) showed that dark currents in a CCD chip form a fixed noise pattern which is added to the videotape taken with the camcorder. Thus, this fixed noise pattern can be used as a "fingerprint" to identify the source of a given videotape. Six years later, Lukas et al. (2005) investigated the different types of noise in imaging sensors, built strong mathematical foundations for Photo-Response Non-Uniformity noise estimation from digital images and, showed how

it could effectively be used to perform digital image source attribution and copy-move forgery detection.

Chen et al. (2007) investigated the identification of video-source camcorders and showed that PRNU can efficiently be used to identify the source camcorder of a video (even a low-resolution video) using individual frames to estimate the sensor's fingerprint, given that enough frames are available (a ten-minute video clip was enough to identify the source of low-resolution videos). Hyun et al. (2012) improved the results in (Chen et al. 2007) by applying a MACE (Minimum Average Correlation Energy) filter to the estimated reference PRNU fingerprint before correlating it with a query video's Sensor Pattern Noise (SPN). Through this, an improvement of up to 10% of the correct attribution rate was achieved.

Houten and Geradts (2009) investigated the usage of the PRNU for source identification of YouTube videos. A set of webcams and codecs were used to record and encode videos which were later uploaded to and downloaded from YouTube. PRNU noise was then estimated from the downloaded videos and classified using normalized cross-correlation. Even though this work gave good results, its findings are now out of date since devices and codecs used by YouTube back in 2009 have significantly evolved.

Villalba et al. (2016) proposed a video source identification scheme based on the usage of PRNU and Support Vector Machine (SVM). A set of 5 smartphones from 5 different brands were used to acquire videos used in training and testing steps. A total of 81 features which are the SPN wavelet components' moments were used to feed an SVM-based classifier. Only native videos (videos taken from the acquisition devices without any post-processing) which have been center-cropped to various resolutions were used. It was reported that the proposed classification scheme has an accuracy of about 87.4 to 90 % depending on video resolution.

Because of the increasing use of digital video stabilization in handheld devices, it became necessary to propose new schemes capable of performing source device attribution for digitally stabilized videos. Performing video source attribution for stabilized video is a challeng-

ing task since video stabilization applies geometric transformations to video frames. Transformations applied to video frames can be rigid, similarity, affine, or perspective transforms, which all lead to the misalignment of the video frames with each other and with the sensor array. Taspinar et al. (2016) proposed a registration scheme for the PRNU noise pattern estimated from digitally stabilized video frames. They considered the first I frame in the video as the reference frame and estimated, using a brute search force, rigid transformations (rotation and translation) registering the PRNU noise from a maximum of 50 I frames to the PRNU noise of the reference frame. The estimation of the rigid transforms was based on the computation of the cross-correlation between the PRNU noise from the reference frame and the rotated frame PRNU noise. Rotated frames giving a maximum cross-correlation value greater than a given threshold were selected. The final PRNU video noise (or camera fingerprint) was estimated as the sum of PRNU noise from all the selected frames together with the noise from the reference frame. They reported a TP rate of 0.83 (for an FP rate of 0.0) on a set of 100 videos when the camera fingerprint was estimated from non-stabilized videos. The TP dropped to 0.65 when videos used for camera fingerprint estimation and query noise were both stabilized videos.

More recently, Iuliani et al. (2017) proposed a "hybrid" approach to video source attribution. Considering the idea that was suggested in (Taspinar et al. 2016), which consist of using images instead of video frames to estimate camera reference fingerprints. For a large set of smartphone and tablet cameras, they established transfer functions between a fingerprint estimated from still images and the one estimated from frames of a non-stabilized video taken with the same camera. These transfer functions consisted of crop and scale parameters that best matched the two fingerprints. Each transfer function was applied to the fingerprint estimated from images and then correlated with the noise pattern estimated from frames of a non-stabilized query video. For stabilized query videos, a set of frames from the video are registered to the fingerprint estimated from still images. The frame registration is done by estimating crop, scale, and rotation parameters giving a maximum cross-correlation value above a given threshold. The PRNU noise pattern of the registered frames is then averaged to

form the video's pattern noise. This approach solved the problem of estimating the reference fingerprint of cameras featuring digital video stabilization (like iPhones and some Android smartphones). They also proposed to link a Facebook and a YouTube account (using images shared on Facebook to estimate the reference PRNU fingerprint and a query YouTube video's frames to estimate the video's PRNU noise). This source identification scheme gave accurate results on native unstabilized and stabilized videos, but its performances on YouTube videos source attribution were very poor (a TP rate of 0.5 at its best). Nevertheless, this method is not usable to perform video device linking. Moreover, in the case of Facebook-shared images, estimating a camera fingerprint using images from unknown sources is not realistic, since it is assumed that they are all coming from the same device, which may not always be the case. It is, thus, necessary to find methods to estimate reliable PRNU fingerprint and video noise using videos frames even when the videos under investigation have been highly compressed and stabilized.

In this research, we investigate novel approaches to source device attribution for highly compressed and digitally stabilized H.264/AVC videos. We propose and test two methods to estimate camera fingerprint and video noise from non-stabilized video frames: a frame-based and, a block-based approach. For digitally stabilized (and highly compressed) videos, we propose a novel scheme to estimate crop and scale parameters matching a PRNU fingerprint estimated from still images to a PRNU noise estimated from frames of a digitally stabilized video. Finally, we propose an effective scheme to register the PRNU noise estimated from frames of a digitally stabilized video to a camera fingerprint.

The major contribution of this research in the field of source device attribution for highly-compressed and non-stabilized videos is the block-based approach which gives, at the best of our knowledge, results which are by far superior to previously published research work. Concerning stabilized and highly-compressed videos, the proof of concept of the proposed method shows the effectiveness of the method which, nevertheless, requires huge computation resources. But, this is not a big deal in the cloud-computing era.

This thesis follows the following organization: Chapter 2 presents theoretical foundations of concepts used throughout the thesis. In Chapter 3, we present our approach to source device attribution of highly compressed and stabilized videos. Experimental results are given in Chapter 4. Chapter 5 concludes our dissertation and gives future perspectives.



2. THEORETICAL FOUNDATIONS

In this chapter, we present theoretical concepts used throughout the thesis. These concepts are mainly related to three topics: The Photo-Response Non-Uniformity Response (PRNU) of digital camera sensors and its use in image/video source device attribution, the H.264/AVC video compression standard, and digital video stabilization.

2.1 PRNU of digital camera sensors and its use for image source attribution

This section introduces the PRNU of digital camera sensors and shows how it is used to perform source attribution for digital media. Before that, we describe the image acquisition process in modern digital cameras, and give for each step of the acquisition the artifacts which can be used for forensics tasks.

2.1.1 The image acquisition process and its artifacts

Figure 2.1 gives a simplified view of the image acquisition process as it is performed in modern digital cameras. The image/video acquisition process starts with a lens which focuses

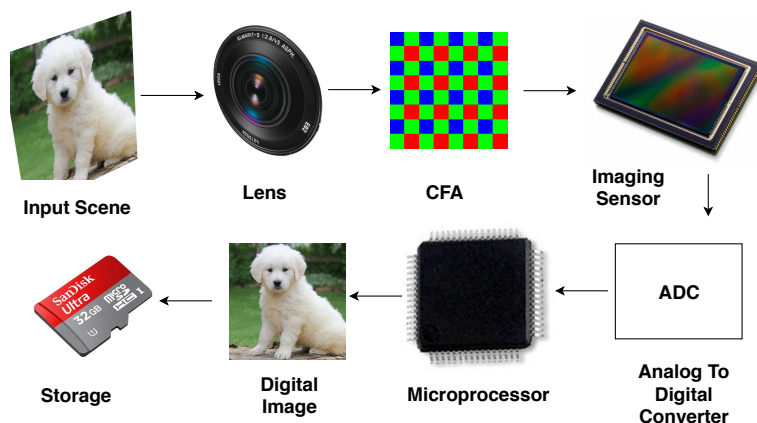


Figure 2.1. Image acquisition pipeline

a scene that lands on a monochromatic image sensor passing through a Color Filter Array

(CFA). The CFA is a matrix of small squared optical filters (corresponding to the sensor's pixels) that filter one of the elementary colors (red, green or blue), and organized according to a given pattern (generally the BAYER patterns). Thanks to the CFA, it is possible to acquire colored images using monochromatic image sensors. The colored image is obtained by interpolating the available color information for each pixel. The imaging sensor array converts the light that falls on each of its cells (pixel) to a voltage. An Analog to Digital Converter (ADC) then converts the voltage into digital values. There are mainly two types of imaging sensor technologies: the CCD (Charge-Coupled Device) and CMOS (Complementary Metal Oxide Semiconductor), which are all made of silicon. The result of the digitalization of the acquired image is a mosaiced image which will be processed by a microprocessor or an ASIC. The processor in the camera performs various operations notably the demosaicing. After demosaicing, the microprocessor applies some image enhancement operations such as white balance, gamma correction on the demosaiced image. The last step is the compression of the acquired image or video frame before its storage into the embedded storage medium.

Components and algorithms used in image acquisition are far from being perfect. Each of them creates artifacts on acquired images/videos, which, even though they do not visibly influence the subjective quality of images/videos, may be used for digital image/video forensics task. The most significant artifacts created by the camera's components or processing algorithms are;

- **The lens distortion and chromatic aberrations**

The most significant imperfections of the camera lens are the radial distortion and chromatic aberrations. The camera lens radial distortion causes straight lines in the scene to become curved lines on the camera sensor. This imperfection of the lens is used in (Choi et al. 2006) to perform image source identification. Chromatic aberrations are due to the fact that the response of the lens depends on the wavelength of the incident light. Because of that, different colors fail to fall at the same position on the camera sensor. The lens chromatic aberrations are used in (Van et al. 2007) as cues to perform

source device (cell phone) attribution for digital images.

- **CFA demosaicing artifacts**

The demosaicing operation reconstitutes the pixel's missing color components by performing an interpolation on the available color information. Depending on the interpolation method used, demosaicing will introduce a specific correlation between color values of a given pixel and those of its neighbors. It is, thus, possible to identify a specific demosaicing algorithm by studying the dependencies of colors between neighboring pixels. The demosaicing artifacts are used in (Bayram et al. 2005) to discriminate camera models.

- **Imaging sensor imperfections**

Imaging sensors are electronic components made of silicon and, therefore possess imperfections due to the inhomogeneity of the silicon and imperfections in manufacture processes. The imaging sensor imperfections create a pattern noise which is present in all images and videos frames taken with the camera. The pattern noise is specific to each sensor (even sensors coming from the same wafer have different pattern noise) and is composed of two main components: The Fixed Pattern Noise (FPN) and the Photo Response Non-Uniformity (PRNU) as displayed in Figure 2.2 (Lukas et al. 2006). The

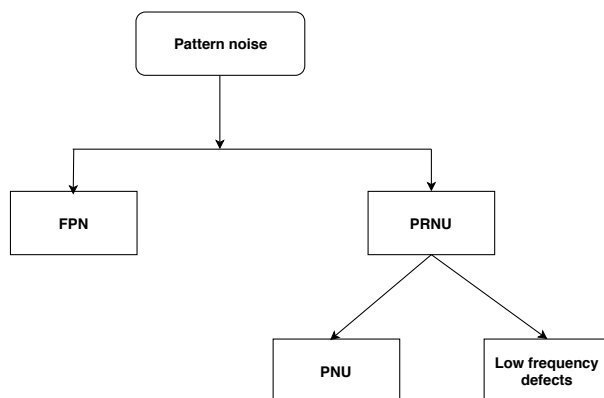


Figure 2.2. Components of the imaging sensor pattern noise

FPN is due to dark currents and represents the pixel-to-pixel differences when the sensor is in complete darkness. High-end consumer cameras automatically remove the FPN by subtracting a dark frame from the acquired image. The PRNU which represents the dominant component of the pattern noise in natural images is the sum of two quantities: PNU and the low-frequency defects. The PNU is due to silicon inhomogeneity and manufacture process imperfections which cause each cell (pixel) of the sensor array to have a different response characteristic (its capacity to convert incident light to current). The low-frequency defects are not intrinsic to the sensor and originate from light refraction on dust particles and zoom settings. The PRNU estimation process performs high-pass filtering and, thus, naturally removes the low-frequency defects. Therefore, in the remaining part of the text, we will refer to PNU as PRNU.

PRNU has the following properties that make it a reliable quantity for multimedia forensics tasks Fridrich (2009) :

1. **Dimensionality.** The fingerprint is stochastic in nature and has a large information content, this fact makes it unique to each sensor.
 2. **Universality.** All imaging sensors (CCD or CMOS) exhibit PRNU.
 3. **Generality.** The PRNU noise pattern is present in every picture, independent of the camera optics, settings or scene content (exception made of completely dark images).
 4. **Stability.** The PRNU pattern noise is stable over time and under a wide range of environmental conditions (temperature, humidity).
 5. **Robustness.** The PRNU pattern noise survives various operations such as filtering, gamma correction and many other typical processing.
- **Post processing and compression algorithm artifacts**

Before its final storage, the camera compresses the image or video frame in order to reduce the space it will take on the storage media. The compression is done using specific algorithms leaving some cues in the compressed image or video. Nowadays, most digital cameras use the JPEG algorithm for image compression and the H.264/AVC for video compression. JPEG image compression divides the image to be compressed in 8×8 sized blocks, quantizes and entropy-codes the DCT transform of each block. The quantization matrix used to quantize DCT coefficients is a design choice that depends on the camera manufacturer. Farid (2006) uses artifacts generated by JPEG compression to perform source device model identification for images. Video compression standards like H.264/AVC (that we will describe in a subsequent section) imply a succession of complex operations which leave cues in the form of artifacts in the compressed video. Su et al. (2009) uses artifacts due to motion estimation algorithms to identify the brand of the camera from which a given video originates; they also use the same approach to identify the software which has been used to compress a given video.

2.1.2 Camera sensor output model

As we have mentioned in the previous section, the PRNU pattern noise is due to differences in the responses of sensor cells. These differences can be quantified using a matrix, \mathbf{K} , of the same size with the sensor. In order to establish a method to estimate \mathbf{K} , we need to derive a mathematical expression of the camera sensor output. Let us consider \mathbf{Y} , a matrix of the same size with the image sensor that represents the light that falls on the image sensor. The digitalized output of the sensor is \mathbf{I} given by (2.1), in which all operations are element-wise (this will be the case for all matrix operations in this text).

$$\mathbf{I} = g^\gamma \cdot [(\mathbf{1} + \mathbf{K})\mathbf{Y} + \mathbf{\Omega}]^\gamma + \mathbf{Q}. \quad (2.1)$$

In (2.1), g is a gain factor and γ is the gamma correction factor. The PRNU matrix \mathbf{K} is a zero-mean noise like-signal. $\mathbf{\Omega}$ is the sum of noises due to dark current, shot noise and read-

out noise. \mathbf{Q} is the noise due to quantization and JPEG compression. By factorizing (2.1) by \mathbf{Y} and taking the two first terms of its Taylor expansion at $\mathbf{Y} = \mathbf{0}$, we obtain the following:

$$\mathbf{I} = g^\gamma \cdot [(\mathbf{1} + \mathbf{K})\mathbf{Y} + \Omega]^\gamma + \mathbf{Q} = (g\mathbf{Y})^\gamma \cdot (1 + \gamma\mathbf{K} + \gamma\Omega/\mathbf{Y}) + \mathbf{Q} = \mathbf{I}^{(0)} + \mathbf{I}^{(0)}\mathbf{K} + \Theta. \quad (2.2)$$

In (2.2), $\mathbf{I}^{(0)} = (g\mathbf{Y})^\gamma$ is the sensor ideal output, which is the output it would have if the sensor was perfect. $\gamma\mathbf{I}^{(0)}\mathbf{K}$, that we will simply note $\mathbf{I}^{(0)}\mathbf{K}$ (the factor γ has been included in the fingerprint matrix \mathbf{K}) is the noise due to PRNU. $\Theta = \gamma\mathbf{I}^{(0)}\Omega/\mathbf{Y} + \mathbf{Q}$ is the modeling noise.

2.1.3 Camera sensor fingerprint estimation

Based on the camera sensor output model given in (2.2), we derive a mathematical method to estimate the camera fingerprint. What we mean by camera fingerprint is just a good estimate of matrix \mathbf{K} . A straightforward method for calculating $\hat{\mathbf{K}}$, an estimate of \mathbf{K} , can be found in (Fridrich 2009). For a given image, by subtracting the denoised image $\hat{\mathbf{I}}^{(0)} = F(\mathbf{I})$ from both sides of (2.2), we obtain the image noise residual given in (2.3).

$$\mathbf{W} = \mathbf{I} - \hat{\mathbf{I}}^{(0)} = \mathbf{I}\mathbf{K} + \mathbf{I}^{(0)} - \hat{\mathbf{I}}^{(0)} + (\mathbf{I}^{(0)} - \mathbf{I})\mathbf{K} + \Theta = \mathbf{I}\mathbf{K} + \Phi \quad (2.3)$$

In (2.3), the noiseless image $\hat{\mathbf{I}}^{(0)} = F(\mathbf{I})$ is obtained with a wavelet-based denoising filter, as described in (Mihcak et al. 1999). $\Phi = \mathbf{I}^{(0)} - \hat{\mathbf{I}}^{(0)} + (\mathbf{I}^{(0)} - \mathbf{I})\mathbf{K} + \Theta$ is the sum of Θ and additional terms introduced by the denoising filter.

Given a set of d images taken with the same camera, a maximum-likelihood-based estimation of \mathbf{K} can be performed using (2.4) (Fridrich 2009).

$$\hat{\mathbf{K}} = \frac{\sum_{k=1}^d \mathbf{W}_k \mathbf{I}_k}{\sum_{k=1}^d (\mathbf{I}_k)^2} \quad (2.4)$$

The computation of the fingerprint estimate $\hat{\mathbf{K}}$ is based on the noise residual \mathbf{W}_k which does not contain only the noise due to PRNU (and other sources of noise) but also eventually the high frequency content of the sample image. This is illustrated by Figure 2.3, where we can see in sub figure (d) how the high frequency content (corresponding to the contours in the image) dominates over the PRNU noise. It is, thus, advised to use flat content images for fingerprint estimation. (Fridrich 2009) states that from 10 to 25 flat content images are enough to obtain a good estimate of a camera fingerprint, and that an estimation of the same accuracy using natural images requires twice the number of images.

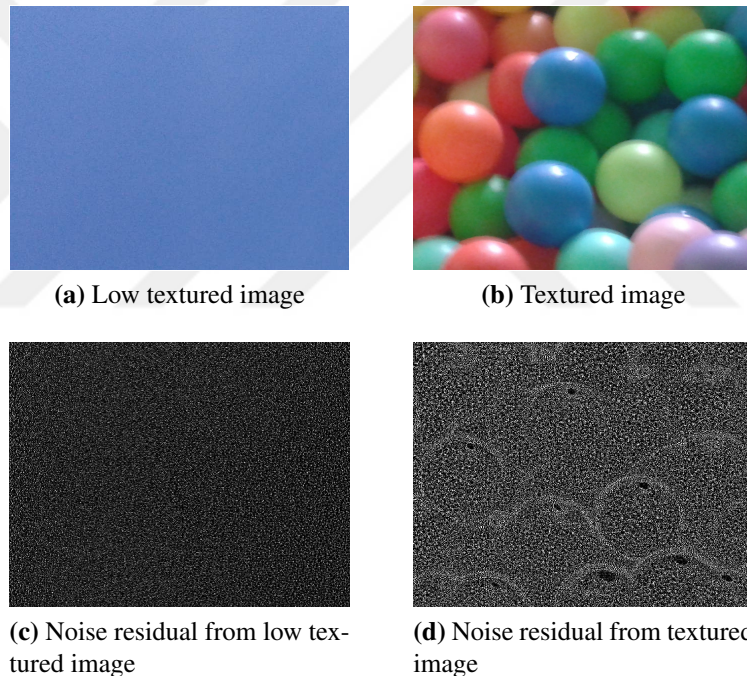


Figure 2.3. Influence of image content on the estimated noise residual (pictures are from the VISION dataset)

The result of the fingerprint estimate given by (2.4) does not contain only the PRNU pattern noise unique to each camera sensor, but also artifacts introduced by processing algorithms used in image acquisition process such as CFA demosaicing and JPEG compression. These artifacts are the same for cameras of the same brand/model and together form what is called the linear pattern, meanwhile the noise pattern due to the PRNU only is called the reference

pattern. Figure 2.4 gives a crop of the reference pattern and the linear pattern estimates for an exemplar of Samsung Galaxy S3 Mini smartphone camera. It is necessary to remove the

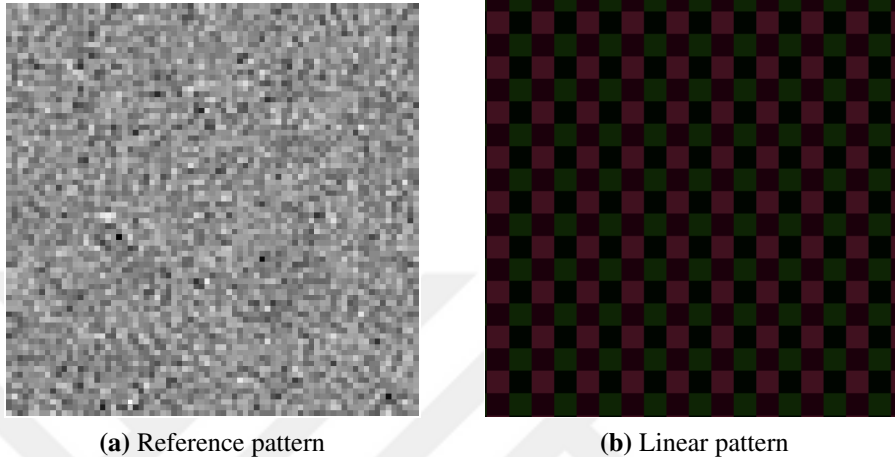


Figure 2.4. Reference and linear patterns for an exemplar of Samsung Galaxy S3 Mini smartphone camera

linear pattern from the fingerprint in order to be able to discriminate cameras from the same model. Filler et al. (2008) used the linear pattern to identify the brand and model of the camera from which an image originates. Removing the linear pattern from the fingerprint is a straightforward task. Since it appears periodically as the averages of rows and columns of $\hat{\mathbf{K}}$, it is enough to subtract these averages from rows and columns of $\hat{\mathbf{K}}$. For the purpose of language simplicity, in the remaining part of this text, we will refer to the reference pattern simply as the fingerprint.

2.1.4 PRNU-Based image source identification

PRNU-based source attribution for digital images has been deeply studied in (Lukas et al. 2005), (Lukas et al. 2006), (Goljan et al. 2007), (Goljan and Fridrich 2008), (Goljan et al. 2009), (Fridrich 2009). In all these research works, the source attribution is done according to the pipeline given in Figure 2.5

The source attribution of a given image with an estimated PRNU noise \mathbf{N} and a camera that

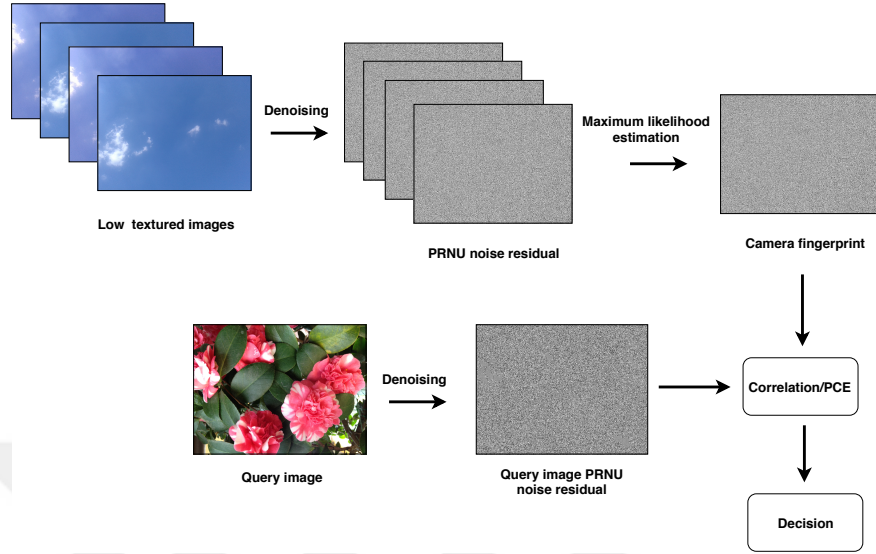


Figure 2.5. Principle of PRNU-based source device attribution for digital images

has a fingerprint \mathbf{F} is formulated as a two-channel hypothesis testing as given in (2.5) (Goljan and Fridrich 2008).

$$\begin{aligned} H_0 : \mathbf{F} &\neq \mathbf{N} \\ H_1 : \mathbf{F} &= \mathbf{N} \end{aligned} \quad (2.5)$$

The classification is based on the computation of the peak (ρ_{peak}) in the normalized cross correlation matrix $\rho(r, c)$ between \mathbf{F} and \mathbf{N} defined as;

$$\rho(r, c) = \frac{\sum_{i=1}^m \sum_{j=1}^n (\mathbf{F}(i, j) - \bar{\mathbf{F}})(\mathbf{N}(i+r, j+c) - \bar{\mathbf{N}})}{\|\mathbf{F} - \bar{\mathbf{F}}\| \|\mathbf{N} - \bar{\mathbf{N}}\|} \quad (2.6)$$

In (2.6), $\bar{\mathbf{F}}$ and $\bar{\mathbf{N}}$ represent the average of matrices \mathbf{F} and \mathbf{N} respectively. Operator $\|X\|$ is the Euclidean norm, r and c are shift parameters ranging from 1 to m and 1 to n , respectively.

The Peak to Correlation Energy (PCE) is suggested in (Fridrich 2009) as a robust way to measure the sharpness of the highest peak in the normalized cross correlation matrix. For a given maximum peak value ρ_{peak} in the normalized cross correlation matrix, its PCE is defined as follows:

$$\text{PCE}(\rho_{peak}) = \frac{\rho_{peak}^2}{\frac{1}{mn-|N|} \sum_{r,c \notin N} \rho(r, c)^2} \quad (2.7)$$

In (2.7), N is a small region surrounding ρ_{peak} and $|N|$ its cardinal. An aggregation threshold τ is defined, and, the null hypothesis (H_0) is rejected if the $PCE(\rho_{peak})$ value is greater than τ and H_1 is rejected elsewhere.

For color images (which is generally the case), three fingerprints $\hat{\mathbf{K}}_R$, $\hat{\mathbf{K}}_G$, $\hat{\mathbf{K}}_B$, that correspond to the three color channels (red, green, and blue) are estimated and combined using the RGB to gray scale conversion equation given by (2.8).

$$\hat{\mathbf{K}} = 0.29\hat{\mathbf{K}}_R + 0.58\hat{\mathbf{K}}_G + 0.11\hat{\mathbf{K}}_B. \quad (2.8)$$

It is on the above notes that we end this section about the PRNU of digital camera sensors and its usage for image source attribution. The next section will present the basis of the H.264/AVC compression standard.

2.2 The H.264/AVC video compression standard basics

In this section, we present key aspects of the H.264/AVC (Advanced Video Compression) video compression standard with the aim of determining how operations involved in video compression affect the PRNU residual noise in video frames. The H.264/AVC video compression standard is a wide subject which cannot be treated in its fullness here, we will only focus on its aspects which are relevant to source attribution of compressed videos. The reader can refer to (Richardson and E. 2010) for a complete and comprehensive guide about the standard and to (JVT 2016) for the all technical details.

The H.264/AVC video compression standard is the world's leading standard for video compression. Nowadays, it is used by almost all smartphones and video-sharing platforms (or social media) like YouTube and Facebook. The H.264/AVC standard is managed by the JVT (Joint Video Team), its first version was released in 2003 and is expected to be replaced by the H.265/HEVC (High-Efficiency Video Coding) standard in the next decade.

Figure 2.6 gives the block diagram of an H.264 encoder/decoder. The figure presents the main principles on which modern video compression standards rely. These principles are; block processing, prediction, transform, quantization, and entropy coding. All these operations (except entropy coding) are performed separately on luminance (Y) and chroma (Cr, Cb) components of video frames. As it is the case for lossy image compression algorithms like JPEG, H.264 takes advantage of the Human Vision System's (HVS) low sensitivity to color, and subsamples the color information. Various chroma sub-sampling schemes are possible : 4:4:4 (no sub-sampling), 4:2:2 and 4:2:0. Figure 2.7 shows the 4:2:0 sub-sampling scheme which is the sub-sampling scheme used in low or middle quality videos. For 4 pixels in vertical and horizontal axes, 2 Cr and 2 Cb pixels are used.

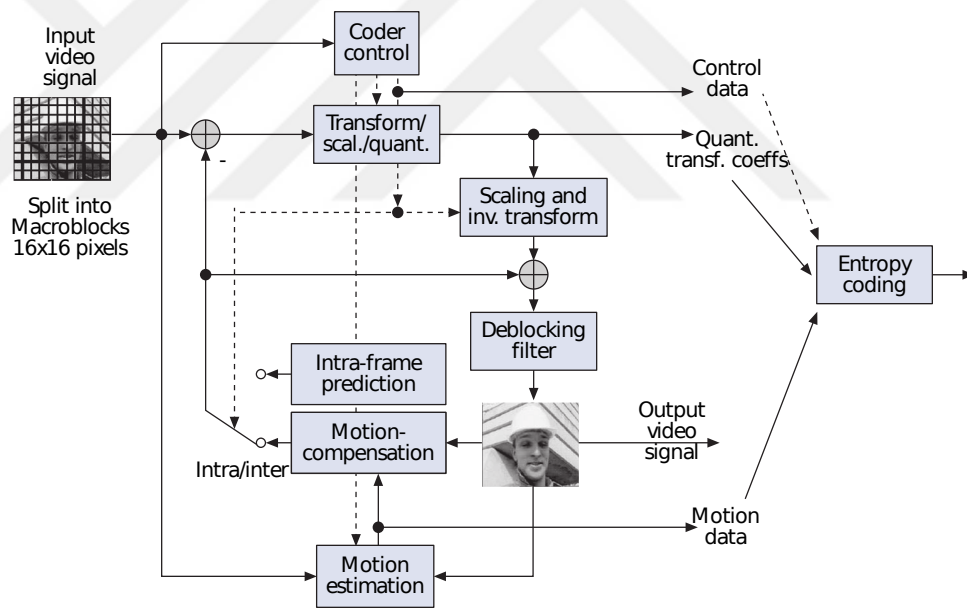


Figure 2.6. Block diagram of an H.264/AVC encoder/decoder (Marpe et al. 2006)

2.2.1 Block processing

An H.264 encoder divides input frames into one or more slices containing Macroblocks (MB) of size 16×16 . According to the type of prediction used, Macroblocks are divided in blocks or sub-Macroblocks of various size as we will show later. Figure 2.8 presents an example

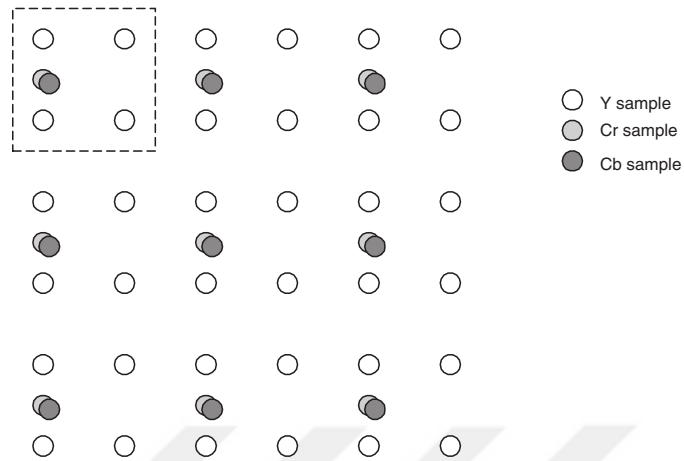


Figure 2.7. 4:2:0 color sub-sampling scheme (Richardson and E. 2010)

of block partitionment in a P frame, low textured blocks are partitioned in bigger blocks meanwhile smaller blocks are used for textured regions of the frame.



Figure 2.8. Example of partition choices for a P frame (Richardson and E. 2010)

2.2.2 Prediction in H.264

Video compression algorithms use prediction to take advantage of high redundancies present in videos. It is a process through which a set of prediction values (previously encoded and decoded blocks), also called reference(s), is used to predict the values of the current block. A prediction residue (the difference between the current values and the reference values) is then formed, and since it often has small values, it requires fewer bits to be represented. Figure 2.9 presents the principle of prediction residue computation as performed in inter-frame prediction. Frame 1 is used as reference to predict Frame 2 and Frame 2 residue is the difference between . The Frame residue has small values except in regions where there is a big difference between the two frames. There are two types of prediction in modern video

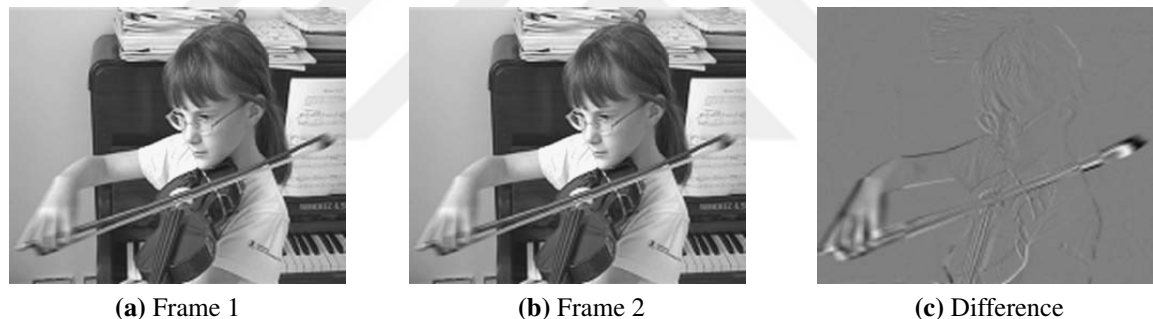


Figure 2.9. Principle of prediction residue computation in inter-frame prediction (Richardson and E. 2010)

compression standards illustrated in Figure 2.10 : intra-frame prediction (or just intra prediction), which exploits spatial redundancy; and inter-frame prediction (or just inter prediction), which exploits temporal redundancy. In intra-frame prediction, reference values and values to be predicted are in the same frame. Previously encoded neighboring samples are used as references. In H.264, intra-prediction is performed on blocks of size 16×16 , 4×4 or 8×8 (Only in high profile encoder). Figure 2.11 presents intra-frame prediction modes used with 16×16 and 4×4 sized blocks. In Figure 2.11, Arrows, when present, show which and how reference samples are used to predict samples in the block which is to be predicted. The DC mode (mode 2) is a particular mode in the sense that it uses a single prediction value

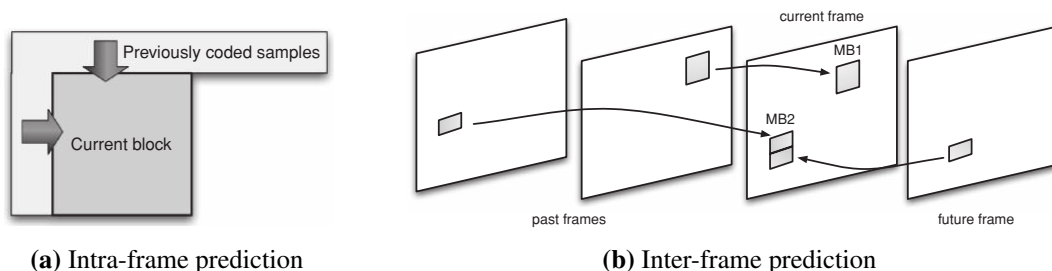
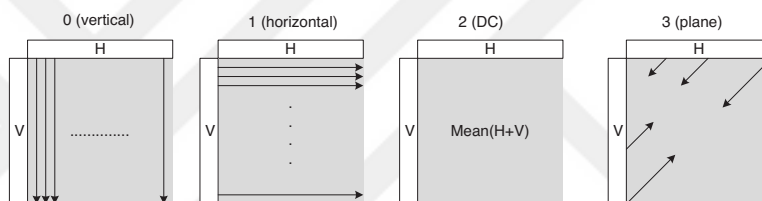
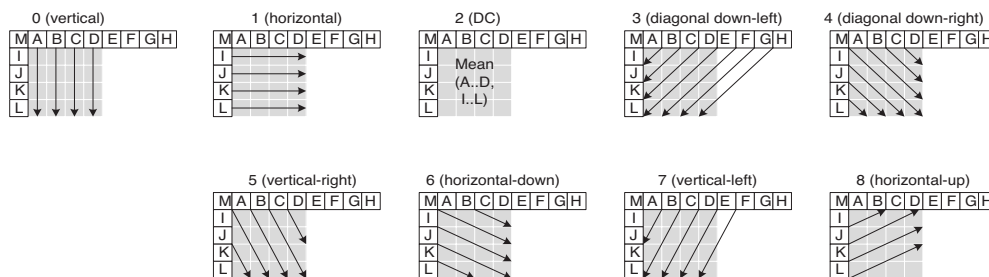


Figure 2.10. Principle of intra-frame and inter-frame predictions (Richardson and E. 2010)

which is the average of all reference samples. In H.264 intra-frame predicted Macroblocks are called I Macroblocks. In inter-frame prediction, reference values are located in past and



(a) 16×16 intra-prediction modes



(b) 4×4 intra-frame prediction modes

Figure 2.11. Intra-prediction modes for 16×16 and 4×4 sized blocks (Richardson and E. 2010)

future frames (see Figure 2.10 (b)). To optimize the efficiency of operations involved in inter-frame prediction (motion estimation and motion compensation), inter-frame prediction uses a different scheme for block partitioning. Figure 2.12 presents the block partitioning scheme used in inter-frame prediction. Inter-frame prediction is based on motion estimation and motion compensation. Motion estimation, which is applied on Macroblock partitions (or sub-Macroblock partition) consists of estimating the motion of a block in the current frame

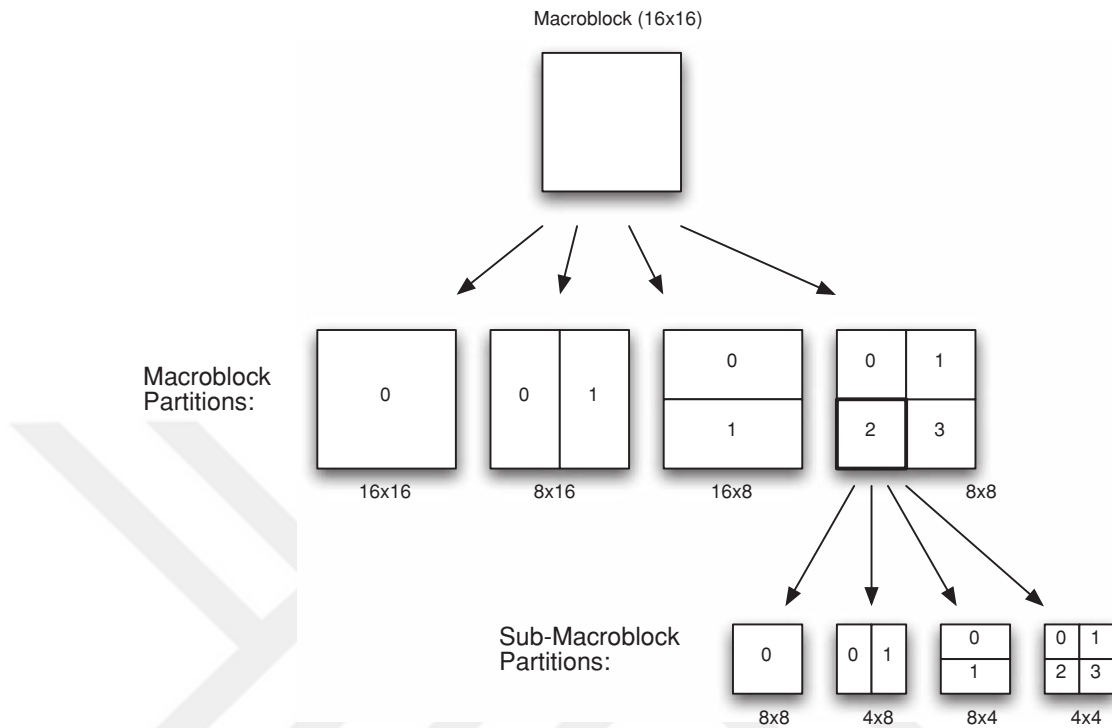


Figure 2.12. Macroblock partitioning in inter-frame prediction (Richardson and E. 2010)

from past and future frames. In other words, motion estimation estimates the position of the predicted block in past or future frames. The result of motion estimation is the predicted block's motion vector used to perform motion compensation. Motion compensation applies the estimated motion vector to the predicted block and subtracts the reference block from the motion compensated block to obtain the block's prediction residue. There are two types of inter-frame predicted Macroblocks which are P and B Macroblocks. P Macroblocks are predicted only from Macroblocks in past frames while B Macroblocks are predicted from Macroblocks in past and future frames.

An H.264-encoded video may contain three types of frames: I, P, and B frames. I frames are frames that contain only I predicted Macroblocks, P frames contain I and P Macroblocks, and B frames contain I, P and B Macroblocks. Encoded frames are gathered in structures called GOP (Group Of Pictures). GOPs always start with I frames. It is important here to highlight the fact that prediction does not result in any loss of information since the prediction residue

can be used to correct prediction errors.

2.2.3 Transform and Quantization in H.264

Transform aims to reduce the statistical correlation between prediction residue values such that most of the information these values contain can be concentrated in a small set of samples. Quantization reduces the precision of transformed samples in order to reduce the number of bits necessary to represent them.

In H.264/AVC, a 2-dimensional Discrete Cosine Transform (2D-DCT) and its inverse (2D-IDCT) are used to perform the transform operation. The 2D-DCT $C_{x,y}$ for a block $I_{i,j}$ of size $N \times N$ is defined by (2.9).

$$C_{x,y} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} I_{i,j} \cos\left(\frac{(2x+1)i\pi}{2N}\right) \cos\left(\frac{(2y+1)j\pi}{2N}\right) \quad (2.9)$$

In a matrix form, the 2D-DCT can be written as given by (2.10) where \mathbf{A} is an orthogonal matrix defined by (2.12) for a 4×4 sized 2D-DCT transform.

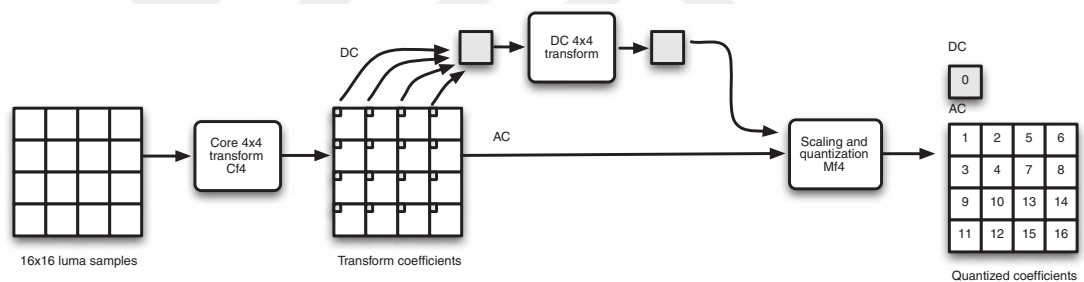
$$\mathbf{C} = \mathbf{A} \cdot \mathbf{I} \cdot \mathbf{A}^T \quad (2.10)$$

$$\mathbf{A} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \sqrt{\frac{1}{2}} \cos\left(\frac{\pi}{8}\right) & \sqrt{\frac{1}{2}} \cos\left(\frac{3\pi}{8}\right) & -\sqrt{\frac{1}{2}} \cos\left(\frac{3\pi}{8}\right) & -\sqrt{\frac{1}{2}} \cos\left(\frac{\pi}{8}\right) \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \sqrt{\frac{1}{2}} \cos\left(\frac{3\pi}{8}\right) & -\sqrt{\frac{1}{2}} \cos\left(\frac{\pi}{8}\right) & \sqrt{\frac{1}{2}} \cos\left(\frac{\pi}{8}\right) & -\sqrt{\frac{1}{2}} \cos\left(\frac{3\pi}{8}\right) \end{bmatrix} \quad (2.11)$$

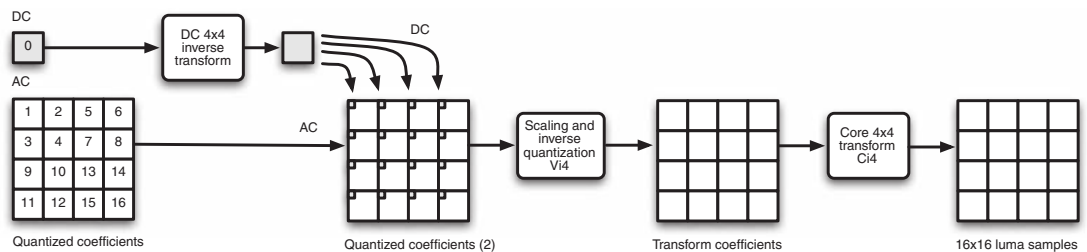
$$\mathbf{A} = \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.653 & 0.271 & -0.271 & -0.653 \\ 0.5 & -0.5 & -0.5 & 0.5 \\ 0.271 & -0.653 & 0.653 & -0.271 \end{bmatrix} \quad (2.12)$$

Contrary to JPEG compression where the transform matrix coefficients are quantized with quantization matrices that perform fine quantization on low-frequency components and coarse quantization on high-frequency components (in order to take advantage of the Human Visual System's low sensitivity to high frequencies), H.264 quantizes all the transform coefficients using the same quantization step (except for High profile encoders which use the same quantization scheme used in JPEG compression).

Figures 2.14, 2.15, and 2.16 present how H.264 performs direct and inverse transform and quantization of the Macroblock residue's Luma and Chroma components using 4×4 and 8×8 DCT transform cores. 8×8 cores are used only for the Luma component in high profile encoders. It is important to highlight that among all the operations involved in



(a) Direct transform and quantization of a Macroblock Luma component

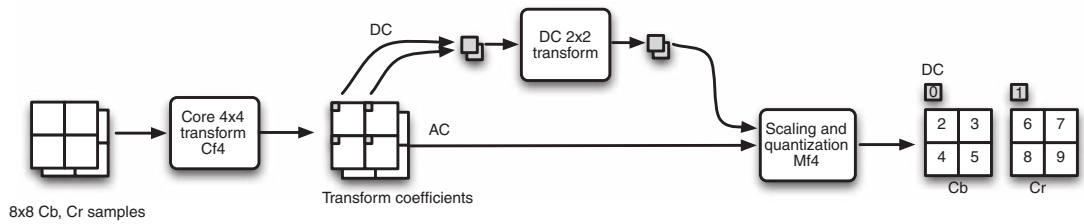


(b) Inverse transform and quantization of a Macroblock Luma component

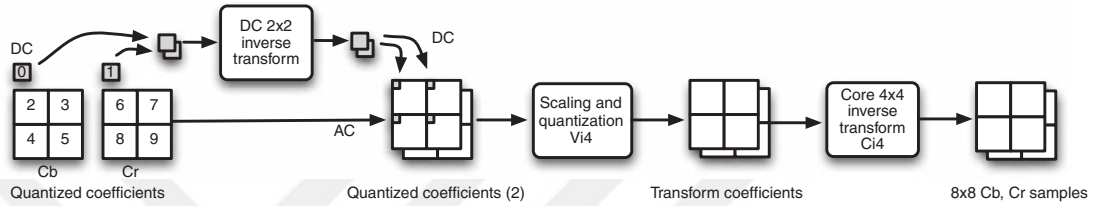
Figure 2.14. Macroblock Luma direct and inverse transform and quantization with a 4×4 DCT transform core (Richardson and E. 2010)

video compression, quantization is the only **irreversible** one, and, therefore leads to a loss of information.

The last step in the video compression process is entropy coding of compression data (con-

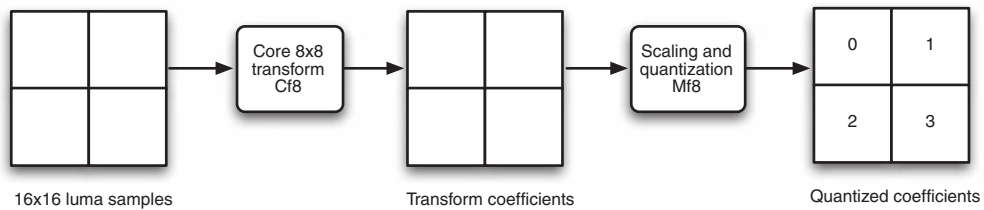


(a) Direct transform and quantization of a Macroblock Chroma component (4:2:0)

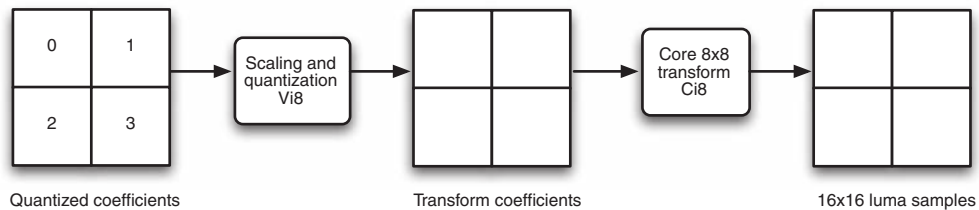


(b) Inverse transform and quantization of a Macroblock Chroma component (4:2:0)

Figure 2.15. Macroblock Chroma direct and inverse transform and quantization with a 4×4 DCT transform core (Richardson and E. 2010)



(a) Direct transform and quantization of a Macroblock Luma component



(b) Inverse transform and quantization of a Macroblock Luma component

Figure 2.16. Macroblock Luma direct and inverse transform and quantization with a 8×8 DCT transform core (Richardson and E. 2010)

ontrol data, quantized transform coefficients, intra-prediction data, motion vectors...). Entropy coding is based on information theory and aims to reduce the number of bits necessary to represent the encoded bitstream. Since entropy coding has no impact on encoded data (it cannot lead to any loss of information), we will not detail it here.

effects of video compression on PRNU noise contained in video frames, we give in Figure 2.18 operations the PRNU noise in a block goes through during encoding and decoding. Let

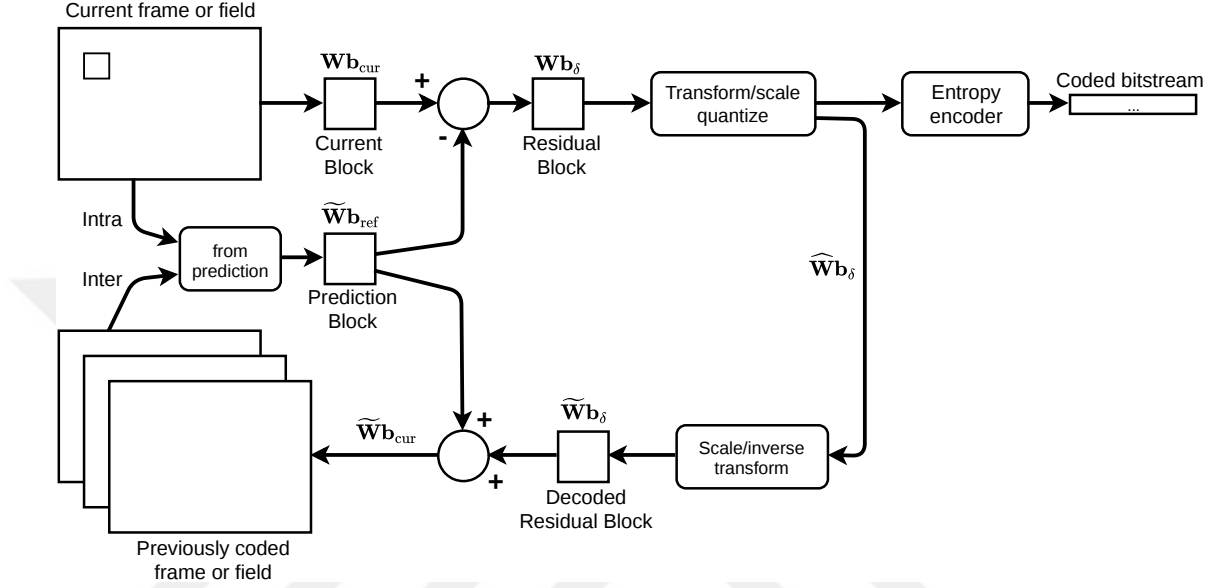


Figure 2.18. Operations applied on a block's PRNU noise during the block's encoding/decoding

us consider a block which is to be encoded. We note \mathbf{Wb}_{cur} as the PRNU noise the block contains, $\tilde{\mathbf{Wb}}_{cur}$ as the PRNU noise that will remain in the block after it has been encoded and decoded, $\tilde{\mathbf{Wb}}_{ref}$ as the PRNU noise contained in the reference block (a previously encoded and decoded block), \mathbf{Wb}_{δ} as the PRNU noise in the current block's prediction residue (see Fig. 2.18). The operations applied to the PRNU noise throughout the block's encoding and decoding processes are given by the following equations. The block's PRNU noise residue is given by

$$\mathbf{Wb}_{\delta} = \mathbf{Wb}_{cur} - \tilde{\mathbf{Wb}}_{ref} \quad (2.15)$$

The output of the DCT transform, scaling, and quantization operations for the residue input can be written as follows

$$\widehat{\mathbf{Wb}}_{\delta} = \text{Quant}[\text{Scale}[\text{DCT}(\mathbf{Wb}_{\delta})]] \quad (2.16)$$

The block reconstruction (decoding) equation is given as follows

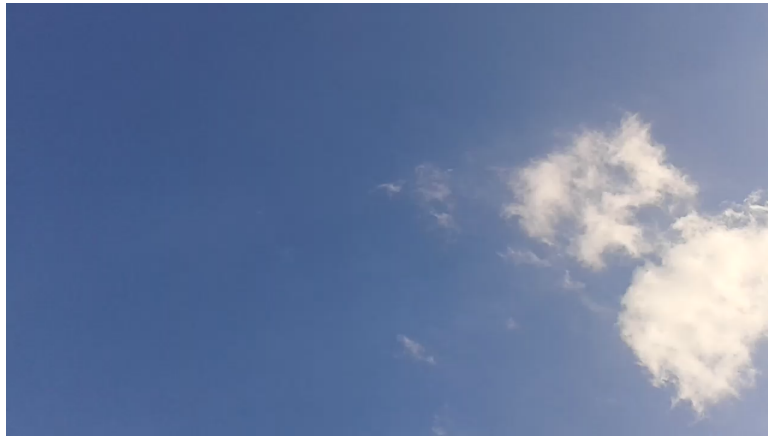
$$\widetilde{\mathbf{Wb}}_{\text{cur}} = \widetilde{\mathbf{Wb}}_{\text{ref}} + \text{DCT}^{-1}[\text{Scale}(\widehat{\mathbf{Wb}}_{\delta})] \quad (2.17)$$

The result of the block reconstruction equation in (2.17) depends on the DCT inverse transform of the block's DCT coefficients matrix and :

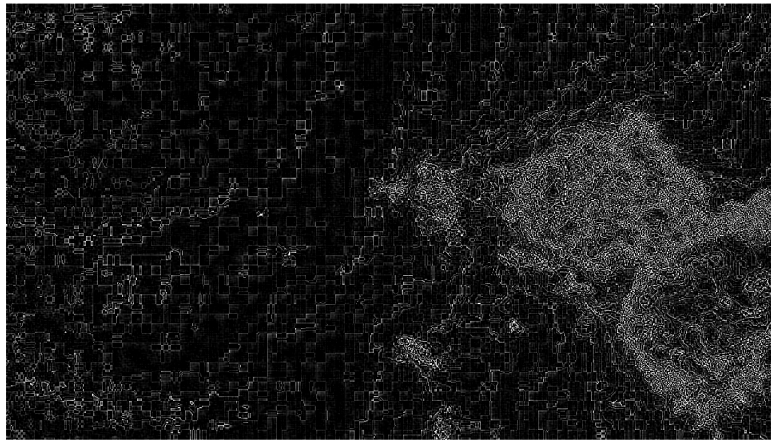
$$\text{If } \widehat{\mathbf{Wb}}_{\delta} = \mathbf{0} \Rightarrow \widetilde{\mathbf{Wb}}_{\text{cur}} = \widetilde{\mathbf{Wb}}_{\text{ref}} \quad (2.18)$$

$$\text{If } \widehat{\mathbf{Wb}}_{\delta} \neq \mathbf{0} \Rightarrow \widetilde{\mathbf{Wb}}_{\text{cur}} \approx \mathbf{Wb}_{\text{cur}} \quad (2.19)$$

Equations (2.18) and (2.19) show that the PRNU noise in the encoded block is not (completely) lost if the DCT-AC coefficients of the encoded block are not all null (because the PRNU noise is by nature a high frequency signal). The strength of the PRNU noise remaining in the decoded block depends on the number of its non-null DCT-AC coefficients and on the scene content, since it has been proven in (Li 2010) (and Section 2.1.3) that high-frequency content scenes affect the quality of the PRNU noise estimation. If all the DCT-AC coefficients (Luma and Chroma) associated to a Macroblock are null, then its PRNU noise is irreversibly lost and replaced by the one in its prediction block(s). To illustrate how compression degrades the PRNU noise in video frames, Figure 2.19 presents an I frame from a highly compressed (YouTube) video and its estimated PRNU noise. We notice that for uniformly textured regions the estimated PRNU noise is also uniform (with some block effects). This is explained by the fact that Intra-frame prediction and DCT transform applied to uniformly textured regions give small-valued DCT-AC coefficients that will fade to zero when they are (highly) quantized. The PRNU noise in predicted blocks is, therefore, equal to the one in its reference blocks. When the prediction value is a weighted sum of prediction samples (like in DC intra-prediction mode shown in Figure 2.11), the PRNU noise of all the pixels in the block is replaced by the weighted sum of prediction samples' PRNU noise values and thus it results a uniform PRNU noise region.



(a) An I frame from a highly compressed video



(b) PRNU noise estimated from the I frame

Figure 2.19. Video compression effect on the PRNU noise (the contrast has been adjusted to improve visibility)

2.3.1 Open source softwares for H.264/AVC video encoding/decoding

We present here some useful Open-source softwares that can be used for H.264 video encoding/decoding and to extract information related to encoded videos. These softwares are: *FFmpeg*, *FFprobe*, and the *jm* H.264/AVC reference encoder/decoder software.

- ***FFmpeg***: FFmpeg (2018) is an open-source library/tool which can be used for video encoding, decoding, transcoding, filtering, and streaming. It supports various video codecs such as H.264/AVC, H.265/HEVC. Thanks to it, it is possible to extract all

frames or a subset of frames (I, P, B frames) from an input video. *FFmpeg* also possesses a tool for video stabilization called *deshake*.

- ***FFprobe*:** *ffprobe* is a tool included in *FFmpeg*. It is meant to be used to extract information from a multimedia stream (in our case, information about a video such as video resolution, video bit rate, encoding profile, and type of each frame) and prints it in human and machine-readable format.
- **The *jm* reference software:** The JM software (JVT 2016) is the official reference H.264/AVC video encoding/decoding software held by the JVT. An interesting point about the JM software is that it has the option to create a trace log file (under the form of an xml file) containing related to encoding/decoding processes while performing video encoding/decoding. Five log levels are available. Table 2.1 gives the information included in the trace log file according to the log level selected.

Table 2.1. Information included in the JM log file according to the log level

| Log level | Information included |
|-----------|-----------------------------|
| 0 | Slice and NAL headers |
| 1 | Macroblock headers |
| 2 | SubMacroblock headers |
| 3 | Motion vectors |
| 4 | Macroblock DCT coefficients |

2.4 Overview of digital video stabilization

Digital video stabilization algorithms are used to remove shakiness from videos taken with hand-held devices to improve viewer experience. Digital cameras can perform video stabilization in real time while acquiring a video. Digital video stabilization can also be done outside the camera using a third-party software. Some high-end cameras possess optical-stabilization systems which use a system of mobile lenses counteracting the camera's vibra-

tions. Optical stabilization has no impact on the alignment of video frames with the camera sensor, for this reason, we will not study it.

Video stabilization is a widely studied subject, many algorithms related to it are available in the literature as in (Chang et al. 2006), (Matsushita et al. 2006), (Grundmann et al. 2011), (Lim et al. 2017). The complexity of operations involved in a given algorithm is one of the most important choice criteria for a given application. Embedded devices generally use light-weight algorithms due to their limited computing capabilities while third-party softwares can use very complex algorithms. Most algorithms available in the literature follow the flow process given in Figure 2.20 (Hlmg et al. 2014). Video stabilization algorithms perform stabilization in three steps: motion estimation, motion smoothing, and motion compensation.

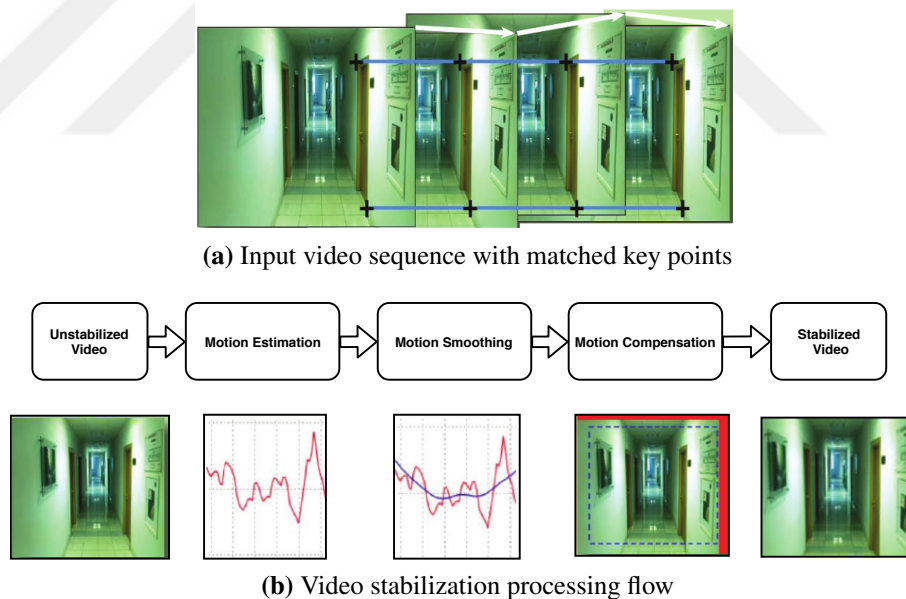


Figure 2.20. Video stabilization on a sample input sequence (Hlmg et al. 2014)

2.4.1 Motion estimation

Motion estimation is a crucial step for any video stabilization algorithm, it consists of determining the global (or dominant) motion from one frame to another. This global motion

is assumed to be due to the camera motion. Common video stabilization algorithms use feature-based (or optical-flow-based) approaches to perform motion estimation. In these approaches, features like Harris corners or SIFT (Scale Invariant Features) are computed and matched to estimate the global motion from one frame to another. Low-cost algorithms use 2D-motion models to model global-frame motion. Table 2.2 presents elementary operations which can be used for 2D-motion modeling (Rafael C. Gonzalez 2009). These elementary operations are combined (by multiplying the elementary transform matrices) to form more complex and accurate models as given by Table 2.3. Low-cost algorithms use affine motion models because of the effectiveness of the approximation and its low cost computation.

Table 2.2. Elementary operations involved in 2D-motion modeling

| Operation | Transform Matrix | Coordinate Equation |
|--------------------|---|---|
| Identity | $\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} w \\ z \\ 1 \end{bmatrix}$ |
| Scaling | $\mathbf{S} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} S_x w \\ S_y z \\ 1 \end{bmatrix}$ |
| Rotation | $\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} w\cos\theta - z\sin\theta \\ w\sin\theta + z\cos\theta \\ 1 \end{bmatrix}$ |
| Shear (horizontal) | $\mathbf{Sh}_x = \begin{bmatrix} 1 & \alpha & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} w + \alpha z \\ z \\ 1 \end{bmatrix}$ |
| Shear (vertical) | $\mathbf{Sh}_y = \begin{bmatrix} 1 & 0 & 0 \\ \beta & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} w \\ \beta w + z \\ 1 \end{bmatrix}$ |
| Translation | $\mathbf{T} = \begin{bmatrix} 1 & 0 & \delta_x \\ 0 & 1 & \delta_y \\ 0 & 0 & 1 \end{bmatrix}$ | $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} w + \delta_x \\ z + \delta_y \\ 1 \end{bmatrix}$ |

Table 2.3. Properties of 2D transformations used for motion modeling

| Transformation | Elementary operations | # Control points | Description |
|-----------------------|-------------------------------------|-------------------------|---|
| Rigid | Rotation, translation | 2 | keeps angles, and straight and parallel lines |
| Similarity | Rotation, scale, translation | 2 | keeps angles, and straight and parallel lines |
| Affine | Rotation, scale, shear, translation | 3 | keeps straight and parallel lines |
| Projective | Rotation, scale, shear, translation | 4 | parallel lines converge toward vanishing points |

2.4.2 Motion smoothing

Motion smoothing aims to remove high-frequency motion vectors estimated in the motion estimation stage. Motion smoothing uses low-pass filters to remove jitters from the camera global motion.

2.4.3 Motion compensation

Motion compensation produces a stabilized video by using smoothed global motion vectors to reconstruct the video frame by frame. When necessary, the motion compensation counteracts high-frequency shakiness by warping shaky frames in order to correct unwanted motion. Warping applied to jittery frames implies scaling, rotation, shear, translation, and crop operations. These operations misalign the PRNU noise of warped frames with the camera sensor and with the other frames. After motion compensation, post-processing operations such as inpainting or blurring can be used to improve the quality on the stabilized video.

In this Chapter, we presented theoretical concepts used throughout this research. These concepts were related to PRNU of digital camera sensors, H.264/AVC video compression and its impact on PRNU in video frames, and digital video stabilization. Digital video is a feature of digital cameras that makes video source device attribution a more complicated task. The next

Chapter will present our proposed methods for video device source attribution. We consider non-stabilized, stabilized and highly compressed videos.



3. MATERIAL AND METHOD

In this chapter, we present our approach to video source-device attribution. Performing source device attribution necessitates two quantities, the reference camera fingerprint and the query video's noise. In this Chapter, we present novel approach to estimate camera fingerprint and query video noise under various circumstances such as when videos are highly compressed and digitally stabilized.

3.1 Source device attribution for non-stabilized videos

In this section, we consider the case where the videos under investigation are non-stabilized videos. We propose to perform source device attribution for non-stabilized videos using only video frames. we propose two approaches, the first one is based on frame processing and the second on block processing.

3.1.1 Frame-based approach

In the frame-based approach, we try to figure out what is the best set of frames which should be used for camera fingerprint and video noise estimation. As we said in Chapter 2, there are mainly three types of frames in a H.264/AVC video: I, P, and B frames. We propose to test experimentally combinations of these types of frame as given in Table 3.1. FFmpeg is

Table 3.1. Combination of frame types for fingerprint estimation in the frame-based approach

| Case | Fingerprint estimated from | Video noise estimated from |
|-------|----------------------------|----------------------------|
| FB_C1 | I frames | I frames |
| FB_C2 | I frames | I + P + B frames |
| FB_C3 | I + P + B frames | I frames |
| FB_C4 | I + P + B frames | I + P + B frames |

used to extract video frames from input videos according to their types and store them in an uncompressed image format (.bmp); FFprobe is used to extract properties of the input video such as the types of frames, the resolution, the video bit rate ... Figure 3.1 illustrates this process; in all cases, the fingerprint/video noise is estimated according to (3.1).

$$\hat{\mathbf{K}} = \frac{\sum_{k=1}^d \mathbf{W}_k \mathbf{I}_k}{\sum_{k=1}^d (\mathbf{I}_k)^2} \quad (3.1)$$

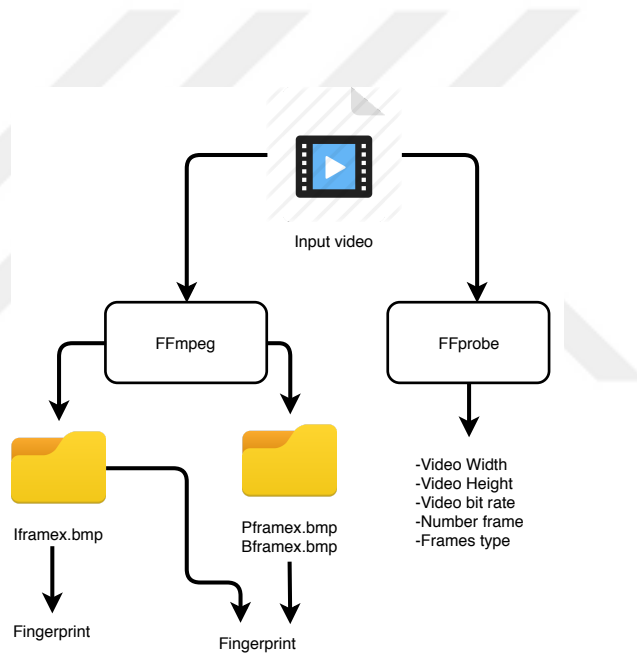


Figure 3.1. PRNU fingerprints estimated from an input video

3.1.2 Block-based approach

The block-based approach relies on the conclusion of Section 2.3 where we established that, the PRNU noise in an encoded block is destroyed (replaced by the one in its prediction block(s)) if the DCT transform matrix of its prediction residue does not have AC component. In the block-based approach, we parse the encoded blocks' DCT coefficients and eliminate those which do not have DCT-AC component. Only blocks that have at least one non-null

DCT-AC coefficient are used for PRNU fingerprint estimation. To parse the bitstream of encoded videos, we used FFmpeg and a modified version of the *jm* 16.1 reference software (JVT 2016). The process is given in Figure 3.2. The input video is first converted to the .264 (H.264 codec) format using FFmpeg; then, the 264 file is decoded using *jm* decoder that we modified to parse DCT coefficients of each blocks after they have been decoded. An xml file is generated during decoding and contains for each Macroblock the following informations: the position of the Macroblock in the frame, the type of prediction used to encode the Macroblock, 4×4 matrix of binary values (0 or 1) giving 4×4 blocks in which there is still PRNU noise. The decoding trace file is then used to build video frame masks.

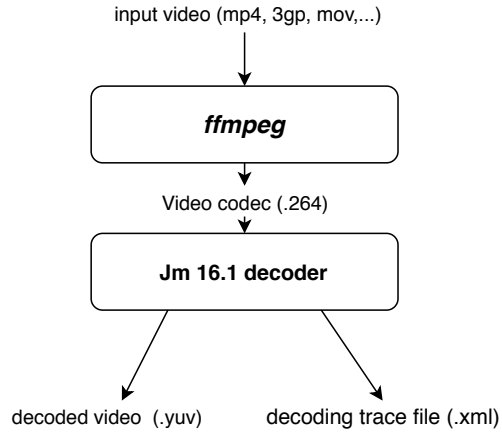


Figure 3.2. H.264-video Macroblocks parsing process flow

A given frame mask M_k associated to a frame k gives pixels in the frame that still have PRNU noise. Elements of M_k are defined in (3.2) where r and c are row and columns indexes respectively.

$$M_k(r, c) = \begin{cases} 0, & \text{If the current block's residue transform does not have DCT-AC component} \\ 1, & \text{Else} \end{cases} \quad (3.2)$$

Figure 3.3 gives an example of a highly compressed I frame and its frame mask. Blocks in white are blocks which shall be used for fingerprint estimation. In the block-based approach, the fingerprint is estimated using frame masks to eliminate blocks in which there is



(a) Example of a highly compressed I frame



(b) Frame mask

Figure 3.3. Example of a highly compressed I frame and its associated frame mask

no remaining PRNU noise.

The formula used to estimate fingerprint in the block-based approach is given by (3.3).

$$\hat{\mathbf{K}}^l = \frac{\sum_{k=1}^d \mathbf{W}_k \mathbf{I}_k \mathbf{M}_k}{\sum_{k=1}^d (\mathbf{I}_k \mathbf{M}_k)^2 + 1} \quad (3.3)$$

3.2 Source device attribution for stabilized videos

Digital video stabilization makes PRNU fingerprint estimation more complicated because it desynchronizes the PRNU noise in video frames toward each order and the camera sensor array. The desynchronization of the PRNU noise in stabilized frames is due to geometrical transforms (scaling, rotation, shear, translation ...) that the stabilization algorithm applies on them to compensate for unwanted motion. It is, therefore, not possible to use frames from a stabilized video to estimate a camera fingerprint or a video noise. Stabilized video frames have to be resynchronized (registered) with each other before using them to estimate a video PRNU pattern noise. The method we propose to perform source device identification of

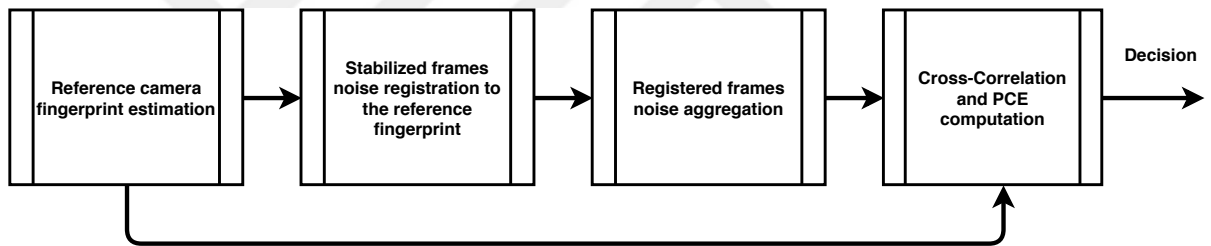


Figure 3.4. Proposed scheme for source device identification of stabilized videos

stabilized videos is given in Figure 3.4, it requires a good estimate of the camera fingerprint to which stabilized frames' noise will be registered; this is not a big deal if the suspect camera does not feature in-camera digital video stabilization. When the suspect camera features digital video stabilization which cannot be disabled by software (as it is the case for iPhones), we have to find alternative methods to obtain a good estimate of the camera fingerprint. We propose as it has been done before in (Iuliani et al. 2017) to use still images to perform a suspect camera's fingerprint estimation; once a reliable estimate of the camera fingerprint is available, it can then be used as reference to register the noise from frames of a stabilized video. In the next two subsections, we first present our method to estimate crop and scale parameters that match a fingerprint estimated from still images with a fingerprint estimated from video frames; then, we present our approach to register stabilized frames noise to a fingerprint.

3.2.1 Estimation of sensor crop and scale parameters that match an image-based to a frame-based fingerprint

Using still images to estimate the fingerprint of a camera featuring digital video stabilization is a good idea since Samet Tapinar et al. demonstrated in (Taspinar et al. 2016) that a reliable estimate of a camera fingerprint could not be obtained using stabilized frames. The problem we have to solve in order to use a fingerprint estimated from images to perform source-device attribution for videos comes from the fact that, even though most cameras use the same sensor for images and videos acquisition, video resolution is in most cases far smaller than image resolution. The reasons why cameras downgrade video resolution are first that storing a video of few minutes (which is made up of thousands of frames) requires much more space compared to a single still image. Secondly, video resolutions are standardized (the 16:9 format is commonly used) which is not the case for still images.

Digital cameras downscale the sensor resolution by performing a central crop and a scaling on the full sensor as illustrated in Figure 3.5 (Iuliani et al. 2017). To estimate the scale and crop

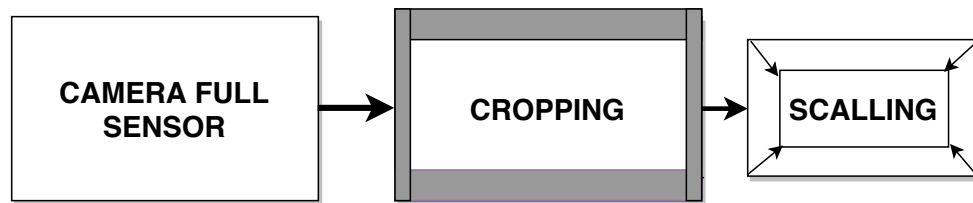


Figure 3.5. Sensor resolution downscaling process in video acquisition

parameters of a given camera sensor, we need a fingerprint estimated from flat-content images and a fingerprint estimated from frames of a flat-content video. When the camera features digital video stabilization, video frames cannot be used to perform fingerprint estimation. In this case, we match the PRNU noise estimated from each frame of a set of selected frames (I frames) with the image-estimated fingerprint. The video used for parameter estimation is a flat-content video acquired in motionless conditions (a camera on a tripod would be perfect) to avoid stabilization to be performed on video frames. The matching parameters of the frame having its PRNU noise giving the highest maximum correlation value when correlated with

the cropped and scaled image-based fingerprint are selected as the camera's image-based to frame-based fingerprint matching parameters. The final camera fingerprint (which will be used for query video frames registration) is the cropped and scaled image-based fingerprint.

Figure 3.6 gives our proposed scheme to estimate row-scaling and column-cropping parameters that match an image-based fingerprint to a frame noise. We proceed in two steps. In the first step, the row-scaling parameter is searched for. To find the row-scaling parameter, we look for the number of rows which gives the highest maximum crosscorrelation value when the scaled fingerprint is correlated with the (zero padded) frame noise. The scaling is performed only according to rows so that the aspect ratio of the image-based fingerprint is preserved. In the second step, we look for the row-cropping parameter, this is done simply by correlating the row-scaled image-fingerprint with the (zero padded) frame noise. The column-crop parameter is given by the position of the highest peak in the cross-correlation matrix. The column-cropping parameter is used to crop the image-based fingerprint in the column direction and to form the final matched fingerprint. Nearest neighbor interpolation is used for scaling operations. Figure 3.7 gives an insight of how the quantities involved in the parameters estimation process evolve, Algorithm 1 gives the algorithm of our proposed scheme.

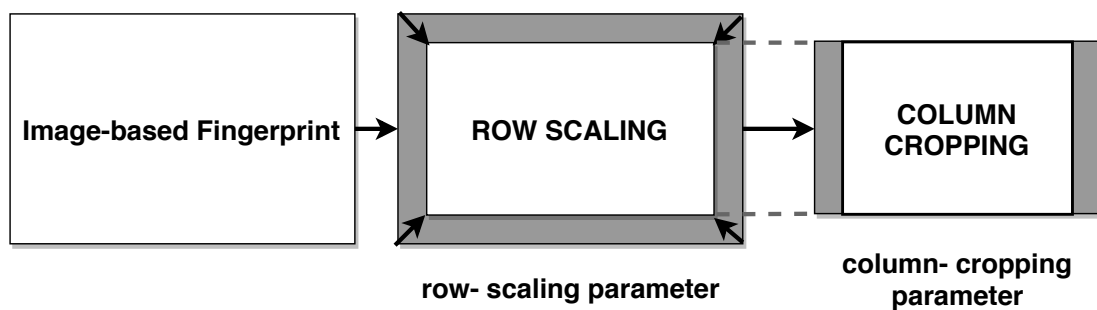


Figure 3.6. Proposed scheme for row-scaling and column-cropping parameters estimation

```

input : imgFingerprint(imgRows × ImgCols), frameNoise(vidRows × vidCols)
output: scaledCroppedFingerprint, rowScalingParam, columnCropParam
n ← 0;
for rows ← vidRows to imgRows do
    scaledImgFingerprint ← resizeKeepAspectRatio (imgFingerprint, rows);
    getSize (scaledImgFingerprint, &rowsTemp, &colsTemp);
    if colsTemp ≥ vidCols then
        if n == 0 then
            maxCorr ← allocateMemory (imgRows − rowsTemp + 1);
            firstRows ← rowsTemp;
        end
        paddedFrameNoise ← allocateMemory (rowsTemp, colsTemp);
        copyAndZeroPadd (&paddedFrameNoise, fameNoise);
        corr ← computeCrosscorrelation (paddedFrameNoise, scaledImgFingerprint);
        maxCorr[n] ← max (corr);
        n ← n + 1;
    end
end
findMaxLocation (maxCorr, &maxCorrLoc);
rowScalingParam ← firstRows + maxCorrLoc − 1;
scaledImgFingerprint ← resizeKeepAspectRatio (imgFingerprint, rowScalingParam);
getSize (scaledImgFingerprint, &rowsTemp, &colsTemp);
paddedFrameNoise ← allocateMemory (rowsTemp, colsTemp);
copyAndZeroPadd (&paddedFrameNoise, fameNoise);
crosscorrMat ← computeCrosscorrelation (scaledImgFingerprint,
    paddedFrameNoise);
findMaxLocation (crosscorrMat, &maxRow, &maxCol);
columnCropParam ← maxCol;
scaledCroppedFingerprint ← crop
    (scaledImgFingerprint, 1, rowsTemp, 1, columnCropParam);

```

Algorithm 1. Sensor scale and crop parameters estimation

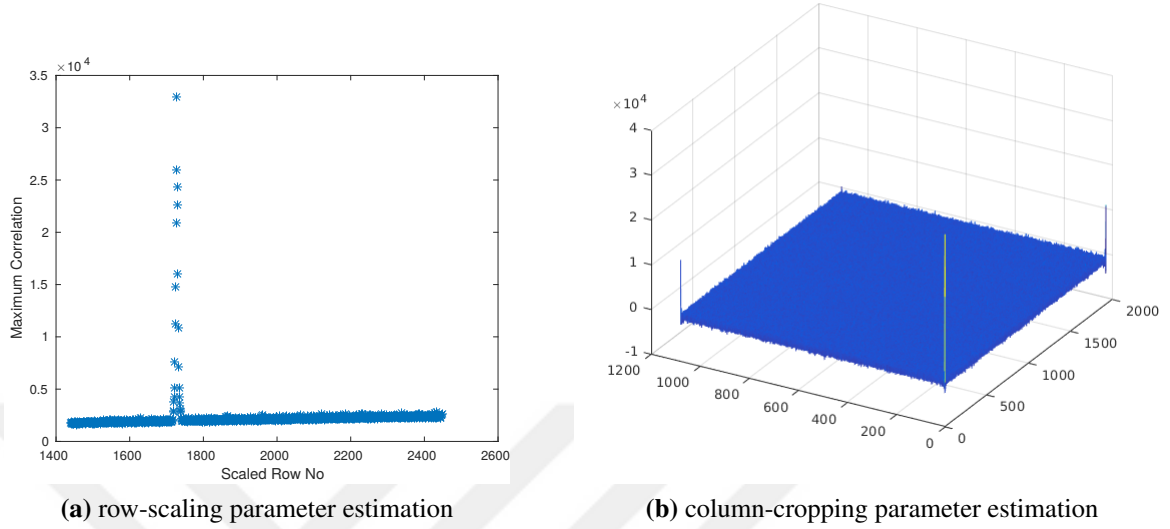


Figure 3.7. Row-scaling and column-cropping based parameters estimation : peaks represent searched parameters

3.2.2 Proposed scheme for stabilized video frames noise registration

Frames noise registration consist of estimating geometric transformations applied on video frames during the video stabilization process and applying the inverse transforms to the stabilized frames in order to re-align them with each other and with the sensor array. The transform estimation process uses a reliable estimate of the query camera’s fingerprint as reference. The reference fingerprint can be estimated using a flat-content video’s frames if the suspect camera does not feature digital video stabilization. If the suspect camera features digital video stabilization, the reference fingerprint can be estimated from still images using the approach proposed in the previous Subsection. The inverses of estimated geometric transforms are then used to register each frame noise to the reference fingerprint. The final video noise is the weighted sum of all registered frames noise.

Up to now, all related researches available in the literature register the stabilized frames noise using either a rigid transform (rotation and translation) like in (Taspinar et al. 2016) or a similarity transform (scaling, rotation, translation) like in (Iuliani et al. 2017). We propose to use an affine transform (scaling, shear, rotation, translation) which is more general than

the previous transforms and which is the one used in popular third-party video stabilization softwares like YouTube Stabilizer (Grundmann et al. 2011) or vidStab of FFmpeg.

An affine transform matrix is defined as follows:

$$\mathbf{T} = \begin{bmatrix} m_{00} & m_{01} & t_x \\ m_{10} & m_{11} & t_y \end{bmatrix} \text{ with } m_{00} = S_x(\cos\theta + Sh_x\sin\theta), m_{01} = S_xSh_y(\cos\theta + Sh_x\sin\theta) + S_x\sin\theta,$$

$$m_{10} = S_y(Sh_x\cos\theta - \sin\theta), m_{11} = S_ySh_y(Sh_x\cos\theta - \sin\theta) + S_y\cos\theta.$$

S_x and S_y are the horizontal and vertical scaling ratio respectively, θ the clockwise rotation angle, Sh_x and Sh_y the horizontal and vertical shear parameters, t_x and t_y the horizontal and vertical translation parameters.

Let us consider $P = \begin{bmatrix} x \\ y \end{bmatrix}$ a point in the plan, its affine transformed point P' is given by

$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} m_{00}x + m_{01}y + t_x \\ m_{10}x + m_{11}y + t_y \end{bmatrix}$$

An affine transform transforms a rectangle to any parallelogram. It necessitates 3 control points to estimate an affine transform as shown in Figure 3.8. Affine transforms are also effective to model small perspective transforms (Adrian Kaehler 2017). In previous works,

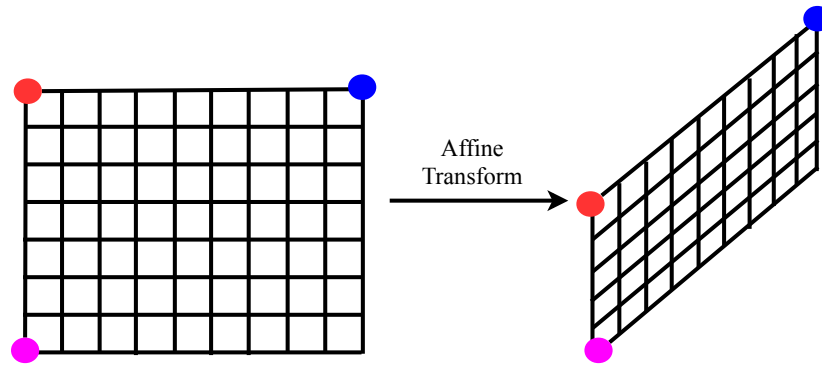


Figure 3.8. Affine transform of a rectangle with its control points

a brute force search is used to search for the transform parameters. We propose a multi-step search based on control points (moving and fixed points) to estimate affine transforms. The

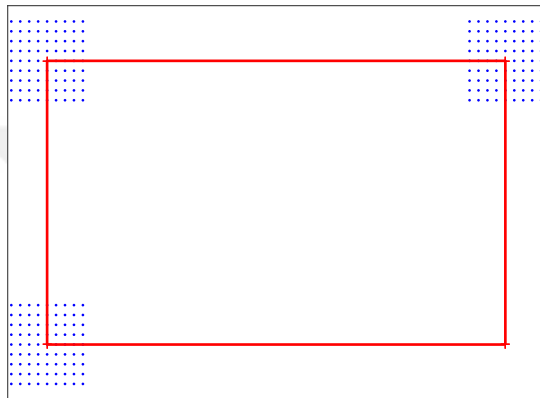
maximums of cross-correlation matrices between the warped reference fingerprint and the frame noise are used as the decision criterion. The parameters search is performed in three passes with the spacing between fixed points reducing after each pass. Best points obtained at a given pass are used as centers for point grids generated at the next pass. We choose to warp the reference fingerprint because it has a better Signal to Noise Ratio (SNR) compared to the frame noise. For warping operations, we use *inverse mapping* and *nearest neighbor* interpolation methods; warped frames noise are cropped if their sizes are above the frame size and zero padded if they are smaller; undefined pixel have zero as value. Figure 3.9 illustrates our proposed scheme for affine transform estimation and Algorithm 2 details it. In Figure 3.9, the red rectangle represents the frame noise and warped rectangles are warped fingerprint (for the purpose of simplicity few combinations are represented).

Input arguments: *camFingerprint* : rowsxcols , *frameNoise* : rowsxcols

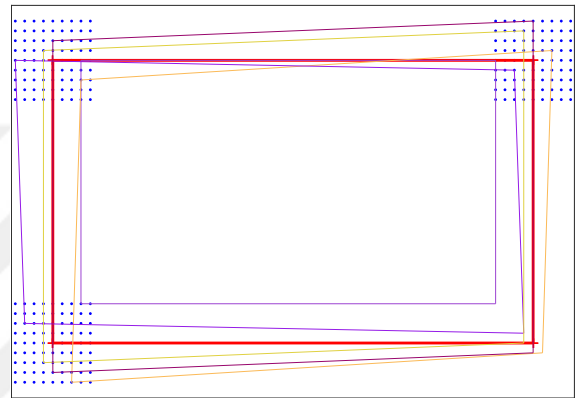
Returned values : *warpedFrameNoise*, *crosscorrMatrix*, *transformMatrix*

```
centerPointTopLeft ← movingPointTopLeft ←  $(-\frac{cols}{2}, \frac{rows}{2})$ ;
centerPointTopRight ← movingPointTopRight ←  $(\frac{cols}{2}, \frac{rows}{2})$ ;
centerPointBottomLeft ← movingPointBottomLeft ←  $(-\frac{cols}{2}, -\frac{rows}{2})$ ;
gridSpacing ← {5,2,1};
gridHalfSize ← 4;
nberSamples ← (gridHalfSize * 2 +1)*(gridHalfSize * 2 +1);
for n ← 0 to 2 do
    fixedPointsTopLeft ← generateFixedPointsGrid (centerPointTopLeft, gridSpacing
        (n), gridHalfSize);
    fixedPointsTopRight ← generateFixedPointsGrid (centerPointTopRight,
        gridSpacing (n), gridHalfSize);
    fixedPointsBottomLeft ← generateFixedPointsGrid (centerPointBottomLeft,
        gridSpacing (n), gridHalfSize);
    maxCorr ← -Inf;
    for i ← 0 to nberSamples-1 do
        for j ← 0 to nberSamples-1 do
            for k ← 0 to nberSamples-1 do
                transformMatrixTemp ← getAffineTransform (movingPointTopLeft,
                    movingPointTopRight, movingPointBottomLeft, fixedPointsTopLeft (i),
                    fixedPointsTopRight (j), fixedPointsBottomLeft (k));
                warpedFingerprint ← warp (camFingerprint,transformMatrixTemp);
                crosscorrMatrixTemp ← computeCrosscorr
                    (warpedFingerprint,frameNoise);
                corr ← max (crosscorrMatrixTemp);
                if corr > maxCorr then
                    centerPointTopLeft ← fixedPointsTopLeft (i);
                    centerPointTopRight ← fixedPointsTopRight (j);
                    centerPointBottomLeft ← fixedPointsBottomLeft (k);
                    crosscorrMatrix ← crosscorrMatrixTemp;
                    maxCorr ← corr;
                end
            end
        end
    end
    end
    transformMatrix ← getAffineTransform (movingPointTopLeft,
        movingPointTopRight, movingPointBottom-
        Left,centerPointTopLeft,centerPointTopRight,centerPointBottomLeft);
    transformMatrix ← getInvTransformMatrix (transformMatrix);
    warpedFrameNoise ← warp (frameNoise,transformMatrix);
end
```

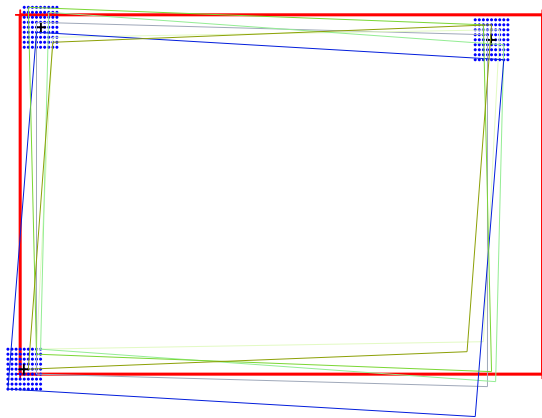
Algorithm 2. Affine transform estimation



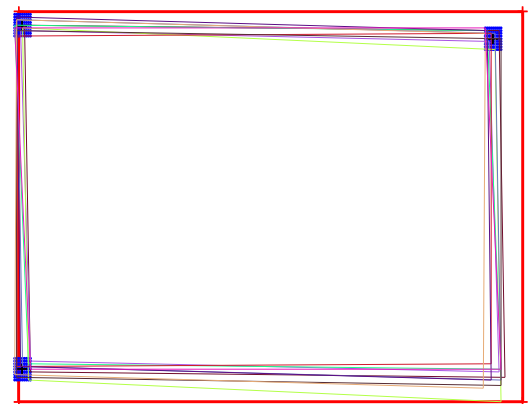
(a) Generated fixed points around the three moving points



(b) Affine transform estimation, first pass (5 pixels spacing)



(c) Affine transform estimation, second pass (2 pixels spacing)



(d) Affine transform estimation, third pass (1 pixel spacing)

Figure 3.9. Affine transform parameters estimation scheme

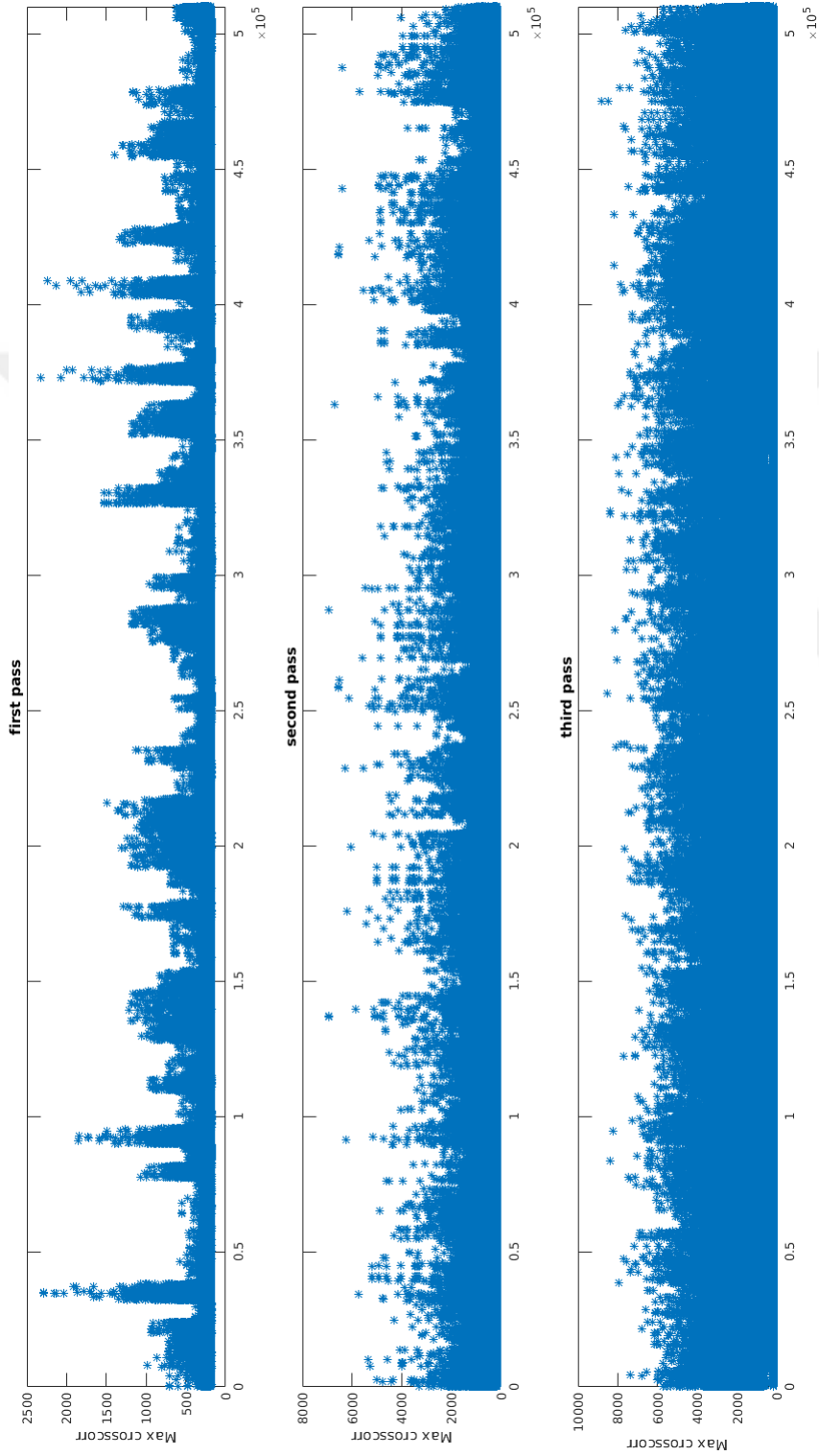


Figure 3.10. Computed maximum cross-correlation values at each step

To test our method, we estimate a camera fingerprint from 50 I frames of a flat-content non-stabilized video and the PRNU noise from an I frame of a natural-content video taken with the same camera. The frame noise is then warped and we try to estimate the original transform matrix using our proposed scheme. The results of this experiment are given as follows:

- Original transform matrix:

$$\mathbf{T} = \begin{bmatrix} 1.046 & 0.091 & 0 \\ -0.088 & 1.016 & 0 \end{bmatrix}, \text{Maximum cross-correlation : } 9434.06$$

- First pass results:

$$\mathbf{T} = \begin{bmatrix} 1.040 & 0.085 & -9.126 \\ -0.085 & 1.010 & 7.075 \end{bmatrix}, \text{Maximum cross-correlation : } 2327.8$$

- Second pass results:

$$\mathbf{T} = \begin{bmatrix} 1.047 & 0.090 & -8.638 \\ -0.090 & 1.015 & 7.075 \end{bmatrix}, \text{Maximum cross-correlation : } 6953.8$$

- Third pass results:

$$\mathbf{T} = \begin{bmatrix} 1.046 & 0.092 & -4.638 \\ -0.088 & 1.016 & 2.571 \end{bmatrix}, \text{Maximum cross-correlation : } 8807.3$$

These results show that our proposed method is capable of estimating the original transform with a high accuracy; $t_x = -4.638$ and $t_y = 2.571$ are translation parameters; they are not null because of undefined pixels introduced by warping.

In our scheme for affine transform estimation, 81 fixed points for each control point are used at each pass; this represents a total of $81 \times 81 \times 81 \times 3 = 1\,594\,323$ cross-correlation

operations which have to be performed to estimate the affine transform matrix of one frame; thus, the computations involved in our proposed scheme have to be optimized (in terms of running time) if we want to use it to perform source device attribution of digitally-stabilized videos (which may contain thousands of frames).

The first optimization we did concerns the implementation of the cross-correlation operation; Equation (2.6) is not used to compute cross-correlation but we use the Fast Fourier Transform (FFT) as it has been demonstrated in (Press et al. 1992) that cross-correlation can be efficiently implemented using FFT. Algorithm 3 details how cross-correlation is computed using FFT operations. The cross-correlation is implemented in C++ using OpenCV 3.1.1 library (Itseez 2017).

Input arguments: *matrix1, matrix2*

Returned values : *crosscorrMatrix*

```
// cross-correlation computation using FFT
matrix1 ← matrix1 - mean (matrix1);
matrix2 ← matrix2 - mean (matrix2);
fftMatrix1 ← fft (matrix1);
fftMatrix2 ← fft (matrix2);
// .* : element wise multiplication operator
// *: complex conjugate operator
crosscorrMatrix ← fftMatrix1.*fftMatrix2*;
crosscorrMatrix ← ifft (crosscorrMatrix);
```

Algorithm 3. cross-correlation computation using FFT

The second optimization we implemented is parallelization. Given that the proposed algorithm is embarrassingly parallel (all operations are totally independent), it is perfectly suited for a multi-thread implementation. We used a Message Passing Interface (MPI) library called Open MPI (Team 2017) which allows us to spawn threads on computers of a computer cluster and implement inter-process communication. The master/slave processing scheme is used with one master thread and the rest of spawned threads being slaves. Algorithm 4 and 5 give a simplified overview of operations performed by the master thread and slave threads respectively.

Input arguments: *camFingerprint* : rows×cols , *frameNoise* : rows×cols

Returned values : *warpedFrameNoise*, *crosscorrMatrix*, *transformMatrix*

```
centerPointTopLeft ← movingPointTopLeft ←  $(-\frac{cols}{2}, \frac{rows}{2})$ ;
centerPointTopRight ← movingPointTopRight ←  $(\frac{cols}{2}, \frac{rows}{2})$ ;
centerPointBottomLeft ← movingPointBottomLeft ←  $(-\frac{cols}{2}, -\frac{rows}{2})$ ;
gridSpacing ← {5,2,1};
gridHalfSize ← 4;
nberSamples ← (gridHalfSize * 2 +1)*(gridHalfSize * 2 +1);
MPI_Comm_size (MPI_COMM_WORLD, &clusterSize);
// Broadcasting the input matrices to all the nodes of the cluster
MPI_Bcast (camFingerprint, MPI_COMM_WORLD);
MPI_Bcast (frameNoise, MPI_COMM_WORLD);
for n ← 0 to 2 do
    fixedPointsTopLeft ← generateFixedPointsGrid (centerPointTopLeft, gridSpacing
        (n), gridHalfSize);
    fixedPointsTopRight ← generateFixedPointsGrid (centerPointTopRight,
        gridSpacing (n), gridHalfSize);
    fixedPointsBottomLeft ← generateFixedPointsGrid (centerPointBottomLeft,
        gridSpacing (n), gridHalfSize);
    // Broadcasting generated fixed points to all computing nodes
    MPI_Bcast (fixedPointsTopLeft, MPI_COMM_WORLD);
    MPI_Bcast (fixedPointsTopRight, MPI_COMM_WORLD);
    MPI_Bcast (fixedPointsBottomLeft, MPI_COMM_WORLD);
    slaveRank = 1;
    for i ← 0 to nberSamples-1 do
        for j ← 0 to nberSamples-1 do
            // Sending indexes of chunks to be processed by slave threads
            MPI_Send (i, j, slaveRank);
            slaveRank = slaveRank + 1;
        end
    end
    // Receiving the computed data from slave threads
    MPI_Recv (&maxCrosscorr, &crosscorrPoints);
    getBestPoints (maxCrosscorr, crosscorrPoints, &centerPointTopLeft,
        &centerPointTopRight, &centerPointBottomLeft);
    transformMatrix ← getAffineTransform (movingPointTopLeft,
        movingPointTopRight, movingPointBottom-
        Left, centerPointTopLeft, centerPointTopRight, centerPointBottomLeft);

    transformMatrix ← getInvTransformMatrix (transformMatrix);
    warpedFrameNoise ← warp (frameNoise, transformMatrix);
end
```

Algorithm 4. Master thread algorithm

Input arguments: *camFingerprint* : rows×cols , *frameNoise* : rows×cols

Returned values : *warpedFrameNoise*, *crosscorrMatrix*, *transformMatrix*

```
centerPointTopLeft ← movingPointTopLeft ←  $(-\frac{cols}{2}, \frac{rows}{2})$ ;
centerPointTopRight ← movingPointTopRight ←  $(\frac{cols}{2}, \frac{rows}{2})$ ;
centerPointBottomLeft ← movingPointBottomLeft ←  $(-\frac{cols}{2}, -\frac{rows}{2})$ ;
gridSpacing ← {5,2,1};
gridHalfSize ← 4;
nberSamples ← (gridHalfSize * 2 +1)*(gridHalfSize * 2 +1);
masterRank = 0;
// Receiving data broadcasted by the master thread
MPI_Bcast (camFingerprint, MPI_COMM_WORLD);
MPI_Bcast (frameNoise, MPI_COMM_WORLD);
while (l==1) do
    // Receiving broadcasted data from the master thread
    MPI_Bcast (&fixedPointsTopLeft, MPI_COMM_WORLD);
    MPI_Bcast (&fixedPointsTopRight, MPI_COMM_WORLD);
    MPI_Bcast (&fixedPointsBottomLeft, MPI_COMM_WORLD);
    // Receiving indexes of points to be processed from the master
    thread
    MPI_Recv (&i, &j, masterRank);
    for k ← 0 to nberSamples-1 do
        transformMatrixTemp ← getAffineTransform (movingPointTopLeft,
            movingPointTopRight, movingPointBottomLeft, fixedPointsTopLeft (i),
            fixedPointsTopRight (j), fixedPointsBottomLeft (k));
        warpedFingerprint ← warp (camFingerprint, transformMatrixTemp);
        crosscorrMatrixTemp ← computeCrosscorr (warpedFingerprint, frameNoise);
        corr ← max (crosscorrMatrixTemp);
        if corr > maxCorr then
            centerPointTopLeft ← fixedPointsTopLeft (i);
            centerPointTopRight ← fixedPointsTopRight (j);
            centerPointBottomLeft ← fixedPointsBottomLeft (k);
            crosscorrMatrix ← crosscorrMatrixTemp;
            maxCorr ← corr;
        end
    end
    // Sending results of computation to the master thread
    MPI_Send (centerPointTopLeft, centerPointTopRight, centerPointBottomLeft,
        masterRank);
    MPI_Send (maxCorr, masterRank);
end
```

Algorithm 5. Slave thread algorithm

After estimating the affine transform matrix of each frame noise, the warped frames noise have to be aggregated to form the stabilized video's query noise. The next Subsection describes our proposed scheme for warped frames noise aggregation.

3.2.3 Warped frames noise aggregation

The last step to form a stabilized video noise is the aggregation of warped frames noise. This process is done on the basis of warped frames noise and the locations of peaks in normalized cross-correlation matrices between the reference fingerprint and warped frames noise. Figure 3.11 gives an example of NCC between a reference fingerprint and a registered frame (for a correctly estimated affine transform). The location of the peak in the NCC matrix defines

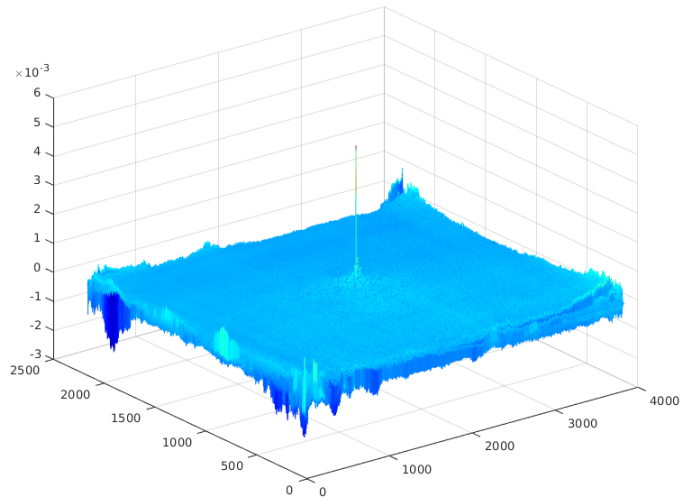


Figure 3.11. Peak in the NCC between a reference fingerprint and a registered frame noise from where (row and column) the warped frame noise should be cropped. After cropping each warped frame noise, its mask matrix is built, a mask matrix has 0 as values where there are undefined pixels in the cropped frame noise and 1 else where. The mask matrices are used to average the cropped frames noise. The code snippet given in Listing 3.1 gives the Matlab code of the function performing warped frames noise aggregation. In this chapter, we have presented novel methods to perform source device attribution for non-stabilized and

Input arguments: *camFingerprint* : $frameHeight \times frameWidth$,
warpedFramesNoise : $frameHeight \times frameWidth \times nberFrames$

Returned values : *videoPRNUnoise*

```

maskMatrices ← initializeMemory (frameHeight, frameWidth, nberFrames);
croppedFramesNoise ← initializeMemory (frameHeight, frameWidth, nberFrames);
for i ← 0 to nberFrames do
    normCx ← computeNCC (normCx, warpedFramesNoise[[i], videoPRNUnoise);
    findPeakLocation (normCx, &xPeak, &yPeak);
    if (xPeak ≥ frameWidth) && (yPeak ≥ frameHeight) then
        xCrop ← xPeak - frameWidth + 1;
        yCrop ← yPeak - frameHeight + 1;
        croppedFramesNoise[yCrop : frameHeight][xCrop : frameWidth][i] ←
            warpedFramesNoise[1 : frameHeight - yCrop + 1][1 : frameWidth - xCrop + 1][i];
    end
    if (xPeak < frameWidth) && (yPeak > frameHeight) then
        xCrop ← frameWidth - xPeak + 1;
        yCrop ← yPeak - frameHeight + 1;
        croppedFramesNoise[yCrop : frameHeight][1 : frameWidth - xCrop + 1][i] ←
            warpedFramesNoise[1 : frameHeight - yCrop + 1][xCrop : frameWidth][i];
    end
    if (xPeak > frameWidth) && (yPeak < frameHeight) then
        xCrop ← xPeak - frameWidth + 1;
        yCrop ← frameHeight - yPeak + 1;
        croppedFramesNoise[1 : frameHeight - yCrop + 1][xCrop : frameWidth][i] ←
            warpedFramesNoise[yCrop : frameHeight + 1][1 : frameWidth - xCrop + 1][i];
    end
    if (xPeak < frameWidth) && (yPeak < frameHeight) then
        xCrop ← frameWidth - xPeak + 1;
        yCrop ← frameHeight - yPeak + 1;
        croppedFramesNoise[1 : frameHeight - yCrop + 1][1 : frameWidth - xCrop +
            1][i] ← warpedFramesNoise[yCrop : frameHeight][xCrop : frameWidth][i];
    end
    // Creating mask frame to disable undefined pixels
    maskMatrices[[i] ← buildFrameMask (croppedFramesNoise[[i]);
end
// Building the matrix used to average warped frames noise
weightMatrix ← buildCroppedFramesWeight (maskMatrices);
videoPRNUnoise ← averageCroppedFramesNoise (croppedFramesNoise,
weightMatrix);

```

Algorithm 6. Registered frames noise aggregation algorithm

stabilized videos. These methods have been tested on a large dataset of videos; the results we obtained are presented in the next chapter.



4. RESULTS

In this Chapter, we present the experimental results of our approaches to source device attribution for digital videos. We first present the VISION video set which was used in our experiments, then, we present the results of proposed schemes on non-stabilized and stabilized videos.

4.1 The VISION dataset

4.1.1 VISION dataset overview

The VISION dataset is a set of images and videos built by Dasara et al. (2017) with the aim of proposing to the community of forensics scientists a large dataset of images and videos which can be used to test source device attribution methods for images and videos. The dataset is a collection of 34 427 images and 1914 videos in their native and social media-shared versions (YouTube and Whatsapp for videos and Facebook for images). Images and videos have been acquired with 35 smartphones and tablets from 11 major brands ranging from low to high price. Figure 4.1 gives the organization of folders associated to each device in the database. Images and videos may contain three type of scenes: flat scenes (skies or flat wall), indoor scenes (classrooms, offices, halls, stores...), and outdoor scenes (nature, garden, city); in this study, we will refer to indoor and outdoor content as natural content. Videos were acquired using three acquisition modes: the *still* mode where the user stands still while capturing the video; the *move* mode where he walks while capturing the scene and the *pan* mode where he performs a pan and rotation while acquiring the scene. All the videos in the dataset have approximately the same duration of about 1 minute and 15 seconds.

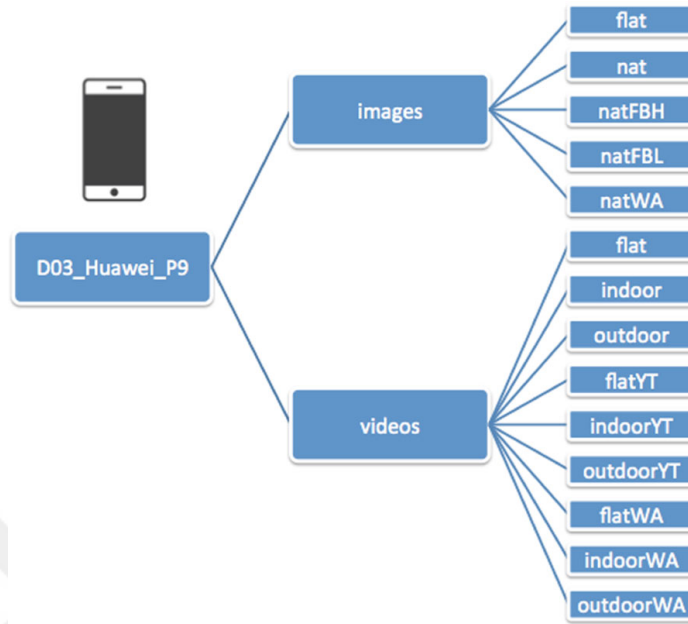


Figure 4.1. Architecture of the VISION dataset (Dasara et al. 2017)

Table 4.1. The set of native videos used in our experiments

| ID | Brand | Resolution | Container | H.264 Profile | Digital Stab | #flat | #natural | total |
|-------|-------------------------|------------|-----------|----------------------|--------------|-------|----------|-------|
| D01 | Samsung Galaxy S3 Mini | 720p | MP4 | Baseline | Off | 10 | 12 | 22 |
| D03 | Huawei P9 | 1080p | MP4 | Constrained Baseline | Off | 7 | 12 | 19 |
| D07 | Lenovo P70A | 720p | 3GP | Baseline | Off | 7 | 13 | 20 |
| D08 | Samsung Galaxy Tab3 | 720p | MP4 | Constrained Baseline | Off | 13 | 24 | 37 |
| D09 | Apple iPhone4 | 720p | MOV | Baseline | Off | 7 | 12 | 19 |
| D11 | Samsung GalaxyS3 | 1080p | MP4 | Baseline | Off | 7 | 12 | 19 |
| D13 | Apple iPad2 | 720p | MOV | Baseline | Off | 4 | 12 | 16 |
| D16 | Huawei P9Lite | 1080p | MP4 | Constrained Baseline | Off | 7 | 12 | 19 |
| D17 | Microsoft Lumia640LTE | 1080p | MP4 | Main | Off | 4 | 6 | 10 |
| D21 | Wiko Ridge4G | 1080p | MP4 | Baseline | Off | 4 | 7 | 11 |
| D22 | Samsung GalaxyTrendPlus | 720p | MP4 | Baseline | Off | 4 | 12 | 16 |
| D24 | Xiaomi RedmiNote3 | 1080p | MP4 | Baseline | Off | 7 | 12 | 19 |
| D26 | Samsung GalaxyS3Mini | 720p | MP4 | Baseline | Off | 4 | 12 | 16 |
| D27 | Samsung GalaxyS5 | 1080p | MP4 | High | Off | 7 | 12 | 19 |
| D28 | Huawei P8 | 1080p | MP4 | Constrained Baseline | Off | 7 | 12 | 19 |
| D30 | Huawei Honor5c | 1080p | MP4 | Constrained Baseline | Off | 7 | 12 | 19 |
| D31 | Samsung Galaxy S4Mini | 1080p | MP4 | High | Off | 7 | 12 | 19 |
| D33 | Huawei Ascend | 720p | MP4 | Constrained Baseline | Off | 7 | 12 | 19 |
| D35 | Samsung Galaxy TabA | 720p | MP4 | Baseline | Off | 4 | 12 | 16 |
| D15 | iPhone6 | 1080p | MOV | High | On | 1 | 1 | 2 |
| Total | | | | | | 125 | 231 | 356 |

4.1.2 Properties of videos used in our experiments

Table 4.1 gives the list of devices and the set of native videos used in our experiments; a total of 20 devices were used. Considering their native and YouTube versions, a total of 710

videos have been used in our experiments. Among these 710 videos, 249 are flat content, and, 461 are natural content videos. Tables 4.2 and 4.3 give properties of native and YouTube videos respectively. YouTube re-encodes (re-compresses) uploaded videos in order to reduce the space needed to store them. In order to reduce the size of uploaded videos, YouTube’s compression algorithms perform a better prediction and a coarser quantization of Macroblock residues’ DCT coefficients. YouTube re-encodes 720p videos using the H.264 Main profile meanwhile 1080p videos are re-encoded using the High profile which has a higher coding efficiency.

Table 4.2. Properties of native videos ([minimum **average** maximum])

| ID | Res | Container | #Frames | #I frames | #P frames | #B frames | bitrate (Kbps) |
|-----|-------|-----------|-------------------------|-------------------|-------------------------|------------------------|----------------------------|
| D01 | 720p | MP4 | [2002 2112 2206] | [67 71 74] | [1935 2041 2132] | [0 0 0] | [1558 3341 5060] |
| D03 | 1080p | MP4 | [2112 2163 2327] | [69 70 76] | [2043 2093 2251] | [0 0 0] | [11907 16567 17067] |
| D07 | 720p | 3GP | [1619 2125 2218] | [54 71 74] | [1565 2054 2144] | [0 0 0] | [4693 8212 9017] |
| D08 | 720p | MP4 | [1576 1912 2132] | [53 64 72] | [1523 1847 2060] | [0 0 0] | [9593 11459 11882] |
| D09 | 720p | MOV | [1576 2027 2291] | [12 46 77] | [581 1567 2214] | [0 414 1369] | [1758 6877 10640] |
| D11 | 1080p | MP4 | [2146 2187 2320] | [72 73 78] | [2074 2114 2242] | [0 0 0] | [1644 27466 81620] |
| D13 | 720p | MOV | [2010 2133 2207] | [14 62 74] | [716 1872 2133] | [0 198 1421] | [4326 9407 10737] |
| D16 | 1080p | MP4 | [2124 2155 2254] | [69 70 73] | [2055 2085 2181] | [0 0 0] | [10003 16110 17151] |
| D17 | 1080p | MP4 | [2110 2138 2239] | [13 47 77] | [679 879 1510] | [696 1211 1400] | [1699 11056 17692] |
| D21 | 1080p | MP4 | [1393 2032 2477] | [47 68 83] | [1346 1963 2394] | [0 0 0] | [19963 20004 20040] |
| D22 | 720p | MP4 | [2114 2174 2407] | [71 73 81] | [2043 2101 2326] | [0 0 0] | [12012 12025 12105] |
| D24 | 1080p | MP4 | [1426 2031 2192] | [48 68 74] | [1378 1963 2118] | [0 0 0] | [19964 19994 20012] |
| D26 | 720p | MP4 | [2080 2161 2253] | [13 64 76] | [1015 1931 2177] | [0 167 1080] | [4109 10901 12166] |
| D27 | 1080p | MP4 | [763 2088 2282] | [26 70 77] | [737 2017 2205] | [0 0 0] | [16990 17010 17075] |
| D28 | 1080p | MP4 | [2068 2152 2296] | [67 70 75] | [2001 2082 2221] | [0 0 0] | [15812 15942 15973] |
| D30 | 1080p | MP4 | [2075 2162 2327] | [67 70 76] | [2008 2092 2251] | [0 0 0] | [17078 17096 17116] |
| D31 | 1080p | MP4 | [2169 2198 2274] | [73 74 76] | [2008 2092 2251] | [0 0 0] | [16977 17005 17018] |
| D33 | 720p | MP4 | [1765 2022 2141] | [59 68 72] | [1706 1954 2069] | [0 0 0] | [7794 8032 8154] |
| D35 | 720p | MP4 | [2099 2150 2208] | [13 66 74] | [779 1939 2134] | [0 145 1307] | [2137 10903 12009] |
| D15 | 1080p | MOV | [1875 1906 1937] | [63 64 65] | [1812 1842 1872] | [0 0 0] | [15486 15494 15503] |

4.2 Source device attribution for non-stabilized videos

Here, we consider the case on non-stabilized videos, three scenarios will be studied: the case where we perform source attribution on native videos (by native video we mean a video which has not been through any processing after its acquisition), the case where we perform source device attribution between native videos and video downloaded from YouTube, and, finally, the case where both videos are downloaded from YouTube. For each of these scenarios, the

Table 4.3. Properties of YouTube videos ([minimum **average** maximum])

| ID | Res | Container | #Frames | #I frames | #P frames | #B frames | bitrate (Kbps) | Profile |
|-----|-------|-----------|-------------------------|--------------------|------------------------|----------------------|-------------------------|---------|
| D01 | 720p | MP4 | [2002 2113 2189] | [13 22 56] | [690 1366 2074] | [0 726 1422] | [546 1741 2319] | Main |
| D03 | 1080p | MP4 | [2111 2161 2325] | [14 52 150] | [735 1520 2054] | [1 589 1414] | [1558 3341 5060] | High |
| D07 | 720p | MP4 | [2083 2153 2218] | [14 42 172] | [713 1307 2081] | [0 803 1401] | [501 1401 2468] | Main |
| D08 | 720p | MP4 | [1576 1920 2132] | [12 31 137] | [581 1216 1914] | [17 674 1369] | [707 1923 2708] | Main |
| D09 | 720p | MP4 | [1997 2115 2291] | [13 50 123] | [747 1675 2024] | [0 390 1324] | [160 1582 2717] | Main |
| D11 | 1080p | MP4 | [2146 2187 2320] | [14 22 63] | [716 1177 2123] | [54 988 1421] | [2026 3332 4346] | High |
| D13 | 720p | MP4 | [2010 2128 2207] | [13 35 166] | [742 1416 2029] | [0 677 1383] | [1026 1906 2317] | Main |
| D16 | 1080p | MP4 | [2110 2141 2239] | [13 49 211] | [722 1508 2092] | [0 584 1400] | [1426 3176 5080] | High |
| D17 | 1080p | MP4 | [2110 2125 2136] | [13 53 132] | [704 1411 1963] | [99 661 1393] | [2182 3654 5090] | High |
| D21 | 1080p | MP4 | [1970 2174 2508] | [13 97 619] | [754 1511 2019] | [0 565 1299] | [1058 3050 5006] | High |
| D22 | 720p | MP4 | [2112 2172 2404] | [14 27 80] | [707 1415 2105] | [0 730 1526] | [1115 1997 2428] | Main |
| D24 | 1080p | MP4 | [1426 2029 2191] | [13 39 152] | [719 1228 2006] | [0 763 1388] | [1276 3228 5104] | High |
| D26 | 720p | MP4 | [2080 2156 2236] | [14 24 81] | [717 1421 2143] | [0 712 1422] | [837 1973 2325] | Main |
| D27 | 1080p | MP4 | [763 2088 2282] | [9 46 141] | [710 1429 2168] | [0 613 1417] | [1533 3210 5031] | High |
| D28 | 1080p | MP4 | [2068 2152 2296] | [13 21 72] | [701 1288 2057] | [42 843 1385] | [1617 3835 5075] | High |
| D30 | 1080p | MP4 | [2064 2151 2315] | [13 34 127] | [710 1339 2078] | [19 779 1417] | [1688 3608 5040] | High |
| D31 | 1080p | MP4 | [2168 2198 2274] | [14 31 128] | [721 1296 2139] | [3 871 1440] | [1932 3668 4354] | High |
| D33 | 720p | MP4 | [1766 2080 2212] | [13 20 42] | [657 1568 2080] | [5 492 1307] | [700 1988 2422] | Main |
| D35 | 720p | MP4 | [2107 2155 2207] | [14 29 165] | [710 1359 2111] | [0 767 1444] | [699 1876 2321] | Main |

four cases described in Table 3.1 are evaluated. For each device, the sets of matching and non-matching videos do not have the same size since we have much more non-matching videos than matching videos; Table 4.4 gives the number of matching and non-matching videos for each device. The results are given separately for 720p and 1080p videos.

4.2.1 Source device attribution for native videos

Table 4.5 gives the AUC (Area Under Curve) values for all 720p videos in our dataset, and, Table 4.6 for 1080p videos. Two cases are presented: the case where native-flat-content videos are matched with native-natural-content videos, and, the case where native-natural-content videos are matched with native-natural-content videos. PCE values statistic distributions are given in box plots (in blue for matching videos and in red for non-matching videos). In each of the box plots, the circle represent the statistical distribution’s median value. The boxes’ bottom and top edges represent respectively the 25th and 75th percentiles’ of the distribution. The whiskers represent the minimum and maximum values of the statistical distribution. Incomplete boxes (without a whisker at the bottom) correspond to distribution in which the minimum values are negative and thus cannot be represented on a Log scale.

Table 4.4. Number of correlation for matching and non-matching videos for each device

| Device | #Matching correlations | #Non-matching correlations |
|--------|------------------------|----------------------------|
| D01 | 120 | 2180 |
| D03 | 84 | 1526 |
| D07 | 91 | 1519 |
| D08 | 312 | 2678 |
| D09 | 84 | 1526 |
| D11 | 84 | 1526 |
| D13 | 48 | 872 |
| D16 | 84 | 1526 |
| D17 | 24 | 896 |
| D21 | 28 | 892 |
| D22 | 48 | 872 |
| D24 | 84 | 1526 |
| D26 | 48 | 872 |
| D27 | 84 | 1526 |
| D28 | 84 | 1526 |
| D30 | 84 | 1526 |
| D31 | 84 | 1526 |
| D33 | 84 | 1526 |
| D35 | 48 | 872 |
| Total | 1607 | 26913 |

Table 4.5. AUC values of 720p native videos matched to native videos

| Case | NativeFlat-NativeNatural | NativeNatural-NativeNatural |
|-------|--------------------------|-----------------------------|
| FB_C1 | 1.00 | 1.00 |
| FB_C2 | 1.00 | 0.99 |
| FB_C3 | 1.00 | 0.99 |
| FB_C4 | 1.00 | 0.99 |
| BB | 1.00 | 1.00 |

Figure 4.2 gives PCE values' statistic distributions of 720p native videos and Figure 4.3 the ones of 1080p videos. These figures and tables show that high accuracies are obtained on native videos using frame-based or block-based approaches. Nevertheless, the block-based approach is the one having the best discriminative capabilities.

Table 4.6. AUC values of 1080p native videos matched to native videos

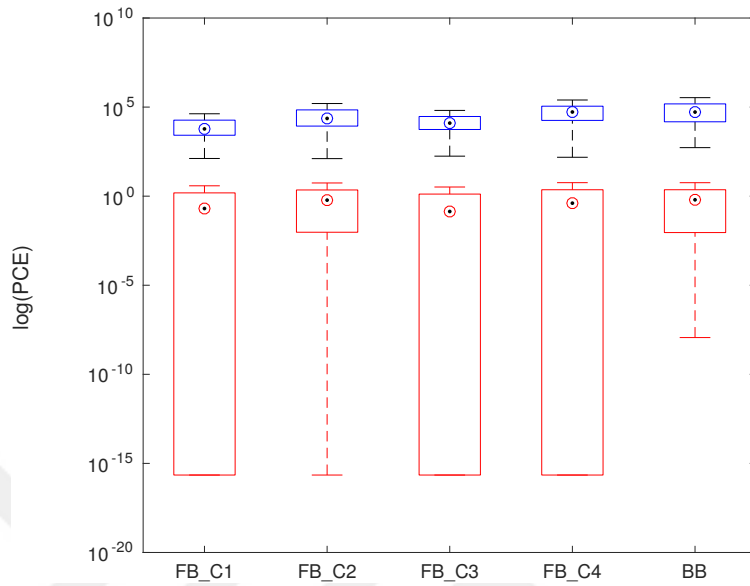
| Case | NativeFlat-NativeNatural | NativeNatural-NativeNatural |
|-------|--------------------------|-----------------------------|
| FB_C1 | 1.00 | 1.00 |
| FB_C2 | 1.00 | 1.00 |
| FB_C3 | 1.00 | 1.00 |
| FB_C4 | 1.00 | 0.99 |
| BB | 1.00 | 1.00 |

4.2.2 Source device attribution for YouTube videos with fingerprints estimated from the native videos

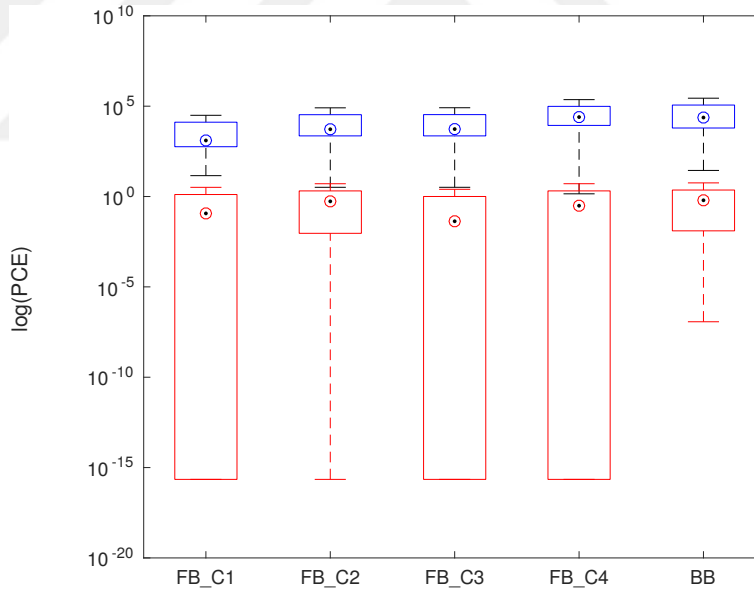
Here, we consider the case where we want to perform source device attribution for videos downloaded from YouTube. Query videos are YouTube videos and videos used to estimate camera fingerprints are native videos (with flat or natural content). Table 4.7 and Table 4.8 give the overall accuracy for 720p and 1080p videos respectively. Figure 4.4 and Figure 4.5 gives the PCE distributions. Globally, we obtain high accuracies and, highest accuracies are achieved when fingerprints are estimated from flat-content native videos. Figure 4.6 and Figure 4.7 gives ROC curves obtained. These ROC curves show that, for the frame-based approach, the best accuracy is obtained for C2 (fingerprints estimated from I frames of native videos and query videos' noise estimated from all YouTube videos' frames). The block-based approach brings a great improvement to the accuracy and gives an accuracy of 100% on 1080p videos. Table 4.9 and Table 4.10 give the AUC values for each device.

Table 4.7. AUC values of 720p native videos matched to YouTube videos

| Case | NativeFlat-YoutubeNatural | NativeNatural-YoutubeNatural |
|-------|---------------------------|------------------------------|
| FB_C1 | 0.99 | 0.97 |
| FB_C2 | 0.99 | 0.99 |
| FB_C3 | 1.00 | 0.98 |
| FB_C4 | 0.99 | 0.98 |
| BB | 1.00 | 0.99 |



(a) Native flat - Native natural

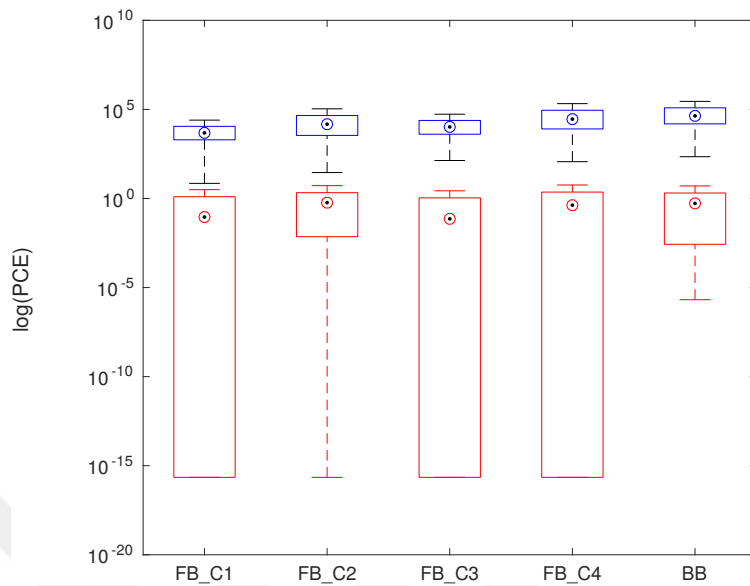


(b) Native natural - Native natural

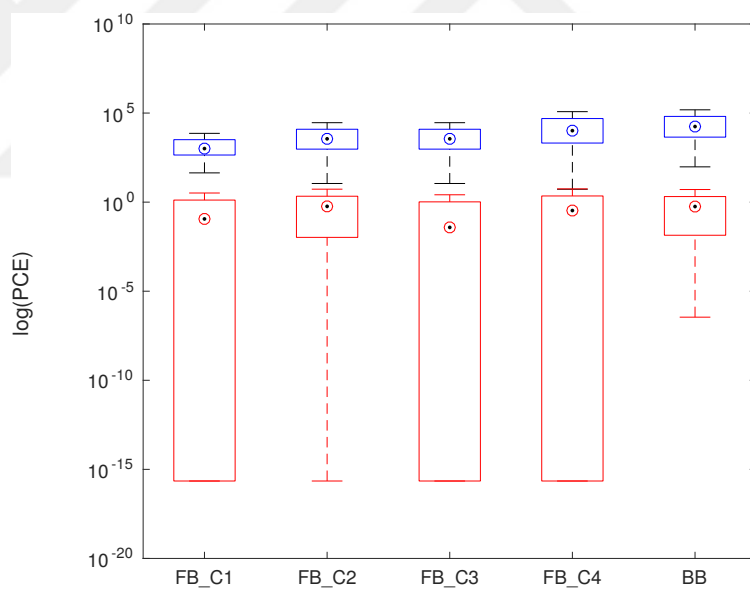
Figure 4.2. PCE distributions of 720p native videos matched to native videos

4.2.3 Source device attribution for YouTube videos with reference fingerprints estimated from YouTube videos

Finally, we consider the case where we desire to link YouTube videos. This represents the most difficult case since camera fingerprints and query videos' noise have to be estimated



(a) Native flat - Native natural



(b) Native natural - Native natural

Figure 4.3. PCE distributions of 1080p native videos matched to native videos

from highly compressed YouTube videos. Table 4.11 and Table 4.12 give the overall accuracy for 720p and 1080p videos respectively; Figure 4.8 and Figure 4.9 give PCE values' statistic distributions; matching and non-matching distributions are no more separable. Figure 4.10 and Figure 4.11 give ROC curves obtained, and, AUC for each device are given by Table 4.13 and Table 4.14.

Table 4.8. AUC values of 1080p native videos matched to YouTube videos

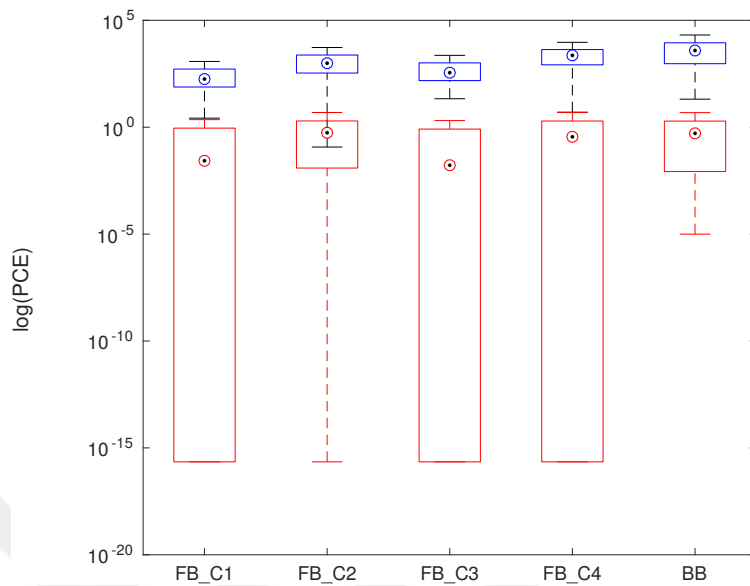
| Case | NativeFlat-YoutubeNatural | NativeNatural-YoutubeNatural |
|-------|---------------------------|------------------------------|
| FB_C1 | 0.99 | 0.94 |
| FB_C2 | 0.99 | 0.98 |
| FB_C3 | 0.99 | 0.96 |
| FB_C4 | 0.99 | 0.98 |
| BB | 1.00 | 1.00 |

Table 4.9. AUC for each device when matching 720p native videos to YouTube videos

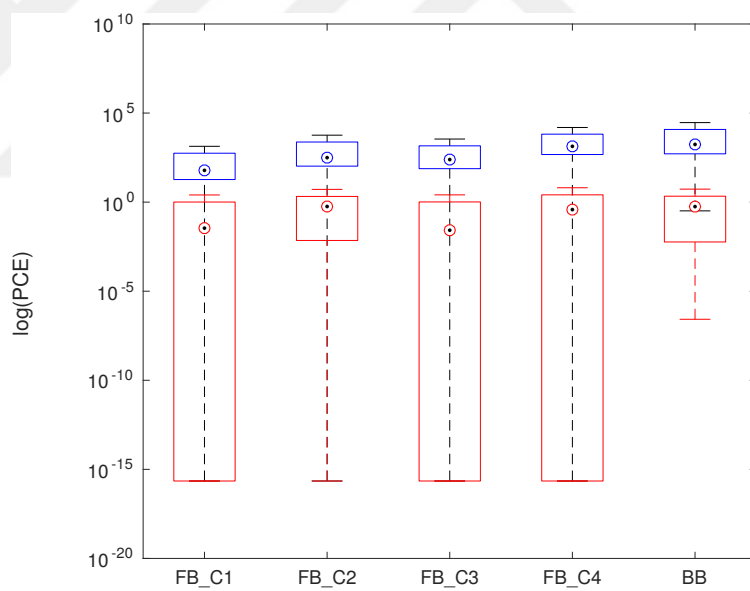
| Device ID | Native flat - YouTube natural | | | | | Native natural - YouTube natural | | | | |
|-----------|-------------------------------|------|------|------|------|----------------------------------|------|------|------|------|
| | C1 | C2 | C3 | C4 | BB | C1 | C2 | C3 | C4 | BB |
| D01 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.99 | 1.00 | 1.00 | 1.00 | 1.00 |
| D07 | 0.99 | 0.98 | 1.00 | 0.99 | 1.00 | 0.83 | 0.93 | 0.90 | 0.92 | 0.98 |
| D08 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.99 | 1.00 | 1.00 | 1.00 | 1.00 |
| D09 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| D13 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| D22 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| D26 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.99 | 1.00 | 1.00 | 1.00 | 1.00 |
| D33 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.98 | 1.00 | 0.99 | 1.00 | 1.00 |
| D35 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.98 | 0.99 | 0.99 | 0.99 | 1.00 |

Table 4.10. AUC values for each device when matching 1080p native videos to YouTube videos

| Device ID | Native flat - YouTube natural | | | | | Native natural - YouTube natural | | | | |
|-----------|-------------------------------|------|------|------|------|----------------------------------|------|------|------|------|
| | C1 | C2 | C3 | C4 | BB | C1 | C2 | C3 | C4 | BB |
| D03 | 0.98 | 1.00 | 1.00 | 1.00 | 1.00 | 0.95 | 0.99 | 0.96 | 0.99 | 1.00 |
| D11 | 1.00 | 0.99 | 1.00 | 0.99 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| D16 | 0.96 | 1.00 | 0.98 | 1.00 | 1.00 | 0.89 | 0.98 | 0.94 | 0.99 | 1.00 |
| D17 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| D21 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.99 | 0.98 | 0.99 | 0.98 | 1.00 |
| D24 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.99 | 1.00 | 0.99 | 1.00 | 1.00 |
| D27 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.98 | 0.99 | 0.98 | 0.98 | 1.00 |
| D28 | 0.99 | 1.00 | 1.00 | 1.00 | 1.00 | 0.90 | 0.97 | 0.94 | 0.97 | 1.00 |
| D30 | 0.98 | 1.00 | 0.99 | 1.00 | 1.00 | 0.88 | 0.98 | 0.93 | 0.97 | 1.00 |
| D31 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.98 | 1.00 | 0.99 | 1.00 | 1.00 |



(a) Native flat - YouTube natural

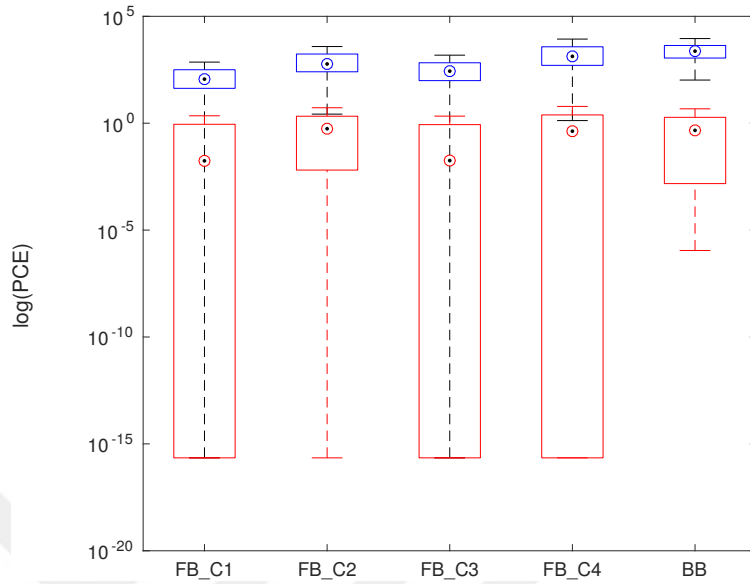


(b) Native natural - YouTube natural

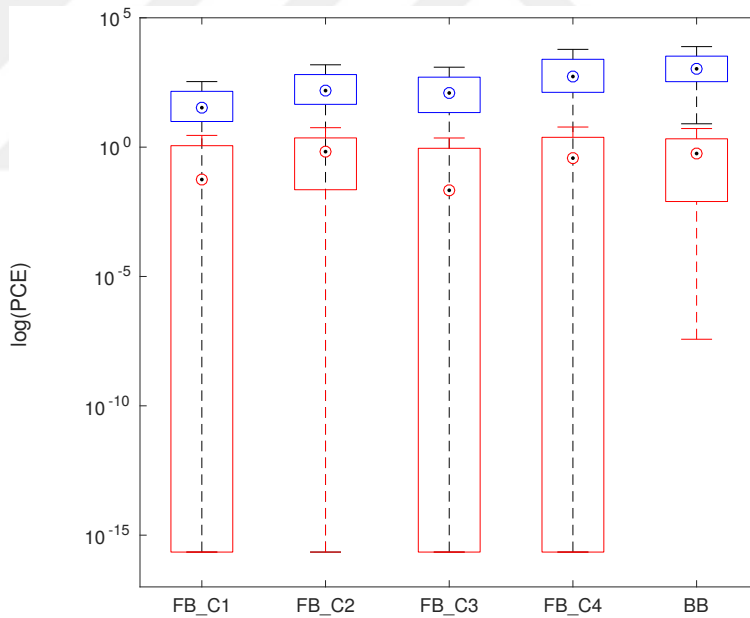
Figure 4.4. PCE distributions of 720p native videos matched to YouTube videos

Figure 4.10 and Figure 4.11 show that:

- when matching flat-content-YouTube videos to natural-content-YouTube videos, performances obtained with C2 and C4 are almost equivalent. This is explained by the fact that inter-frame prediction replaces the PRNU noise in inter-predicted frames (P



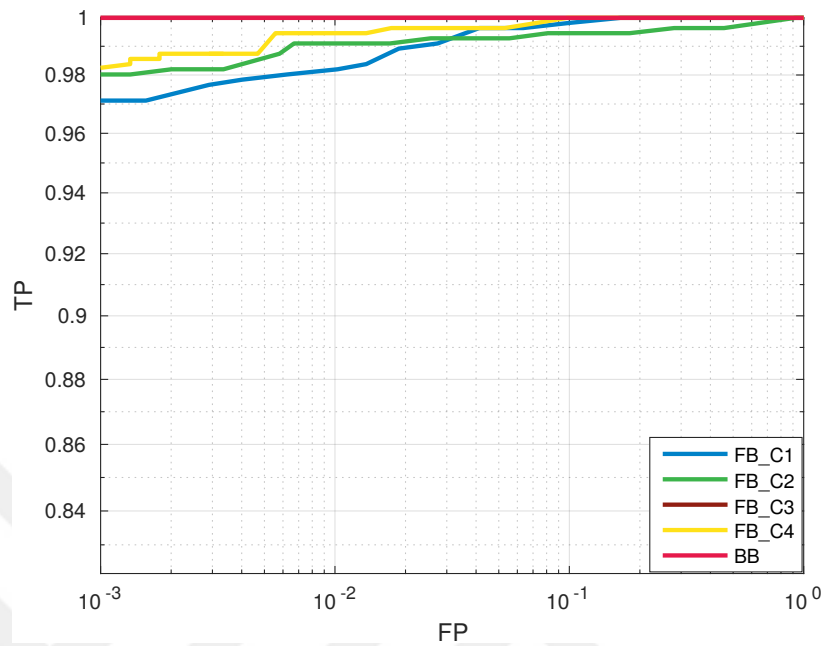
(a) Native flat - YouTube natural



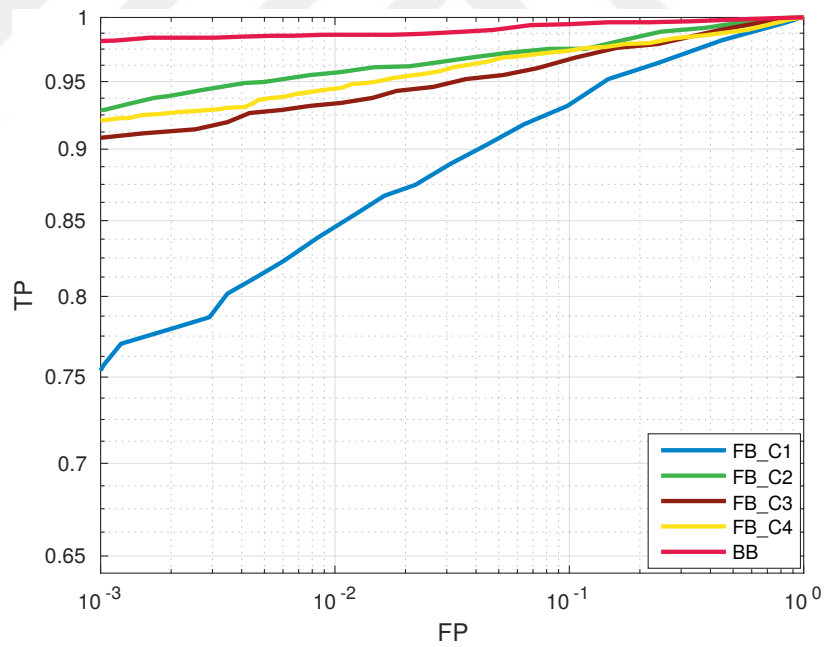
(b) Native natural - YouTube natural

Figure 4.5. PCE distributions of 1080p native videos matched to YouTube videos

or B) frames by the one in I frames because of the steadiness of scenes (leading to prediction residue having only DC component). If we consider a GOP, the PRNU noise in P (and B) frames is redundant to the PRNU noise in the I frame at the beginning of the GOP. As result, the fingerprint estimated from I frames is almost equivalent to the one



(a) Native flat - YouTube natural

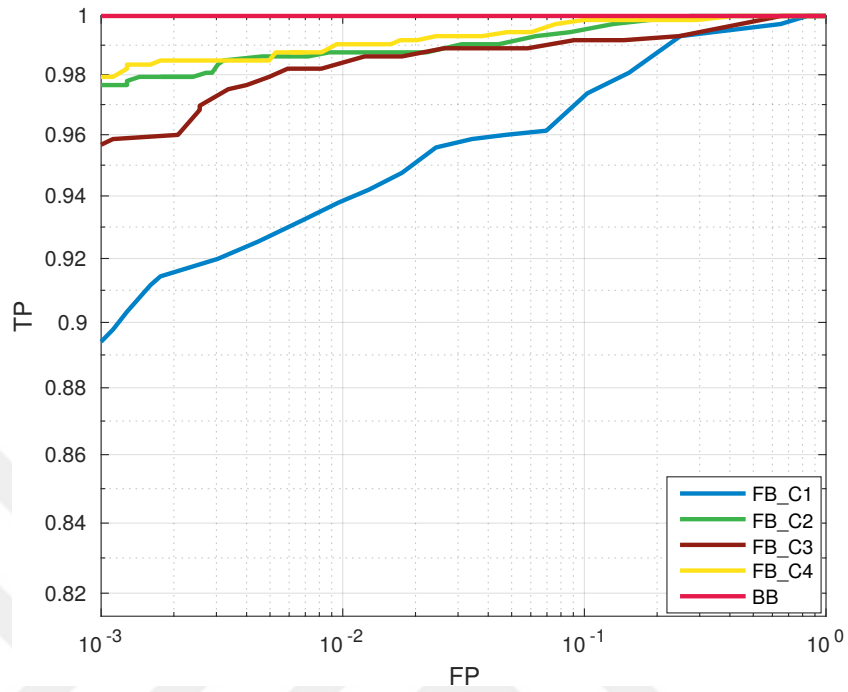


(b) Native natural - YouTube natural

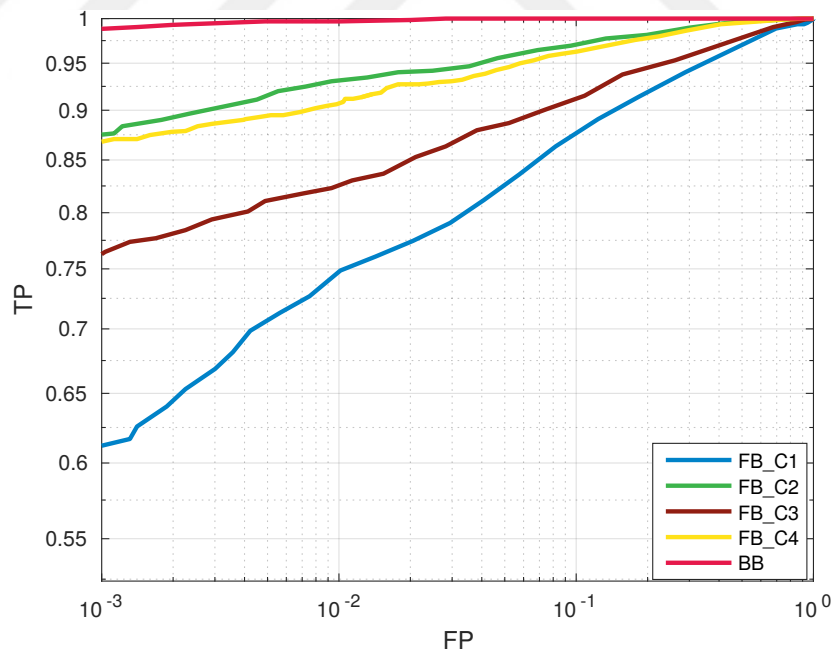
Figure 4.6. ROC curves of 720p native videos matched to YouTube videos

estimated from all the frames of the video.

- when matching natural-content-YouTube videos to natural-content-YouTube videos,



(a) Native flat - YouTube natural



(b) Native natural - YouTube natural

Figure 4.7. ROC curves of 1080p native videos matched to YouTube videos

the best performance (which is really poor) for the frame-based approach is obtained with C4 (C2 and C3 are almost equivalent). This means that, for natural scenes, there

Table 4.11. AUC values of 720p YouTube videos matched to YouTube videos

| Case | YoutubeFlat-YoutubeNatural | YoutubeNatural-YoutubeNatural |
|-------|----------------------------|-------------------------------|
| FB_C1 | 0.90 | 0.78 |
| FB_C2 | 0.95 | 0.88 |
| FB_C3 | 0.94 | 0.87 |
| FB_C4 | 0.95 | 0.90 |
| BB | 0.96 | 0.97 |

Table 4.12. AUC values of 1080p YouTube videos matched to YouTube videos

| Case | YoutubeFlat-YoutubeNatural | YoutubeNatural-YoutubeNatural |
|-------|----------------------------|-------------------------------|
| FB_C1 | 0.86 | 0.69 |
| FB_C2 | 0.91 | 0.79 |
| FB_C3 | 0.89 | 0.79 |
| FB_C4 | 0.90 | 0.83 |
| BB | 0.98 | 0.98 |

Table 4.13. AUC values for each device when linking 720p YouTube videos

| Device ID | YouTube flat - YouTube natural | | | | | YouTube natural - YouTube natural | | | | |
|-----------|--------------------------------|------|------|------|------|-----------------------------------|------|------|------|------|
| | C1 | C2 | C3 | C4 | BB | C1 | C2 | C3 | C4 | BB |
| D01 | 0.92 | 1.00 | 0.97 | 0.99 | 1.00 | 0.75 | 0.88 | 0.90 | 0.92 | 1.00 |
| D07 | 0.71 | 0.77 | 0.75 | 0.79 | 0.74 | 0.62 | 0.64 | 0.64 | 0.69 | 0.82 |
| D08 | 1.00 | 1.00 | 1.00 | 0.99 | 1.00 | 0.84 | 0.95 | 0.94 | 0.96 | 1.00 |
| D09 | 1.00 | 0.99 | 1.00 | 0.99 | 1.00 | 0.96 | 0.96 | 0.96 | 0.95 | 0.98 |
| D13 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.97 | 1.00 | 1.00 | 1.00 | 1.00 |
| D22 | 0.96 | 0.99 | 0.99 | 0.98 | 1.00 | 0.81 | 0.94 | 0.93 | 0.97 | 1.00 |
| D26 | 0.93 | 0.98 | 0.98 | 0.98 | 1.00 | 0.74 | 0.93 | 0.93 | 0.98 | 1.00 |
| D33 | 0.88 | 0.98 | 0.98 | 0.99 | 1.00 | 0.68 | 0.89 | 0.88 | 0.94 | 1.00 |
| D35 | 0.88 | 0.98 | 0.95 | 0.96 | 1.00 | 0.75 | 0.77 | 0.77 | 0.76 | 0.97 |

is still PRNU noise in P (and B) frames and they contribute to estimate a better PRNU video noise compared to the case where only I frames are used (C1). Again, the block-based approach brings a great improvement to the accuracy particularly for 1080p videos.

On the basis of all the results presented in this section, we can take the following conclusions

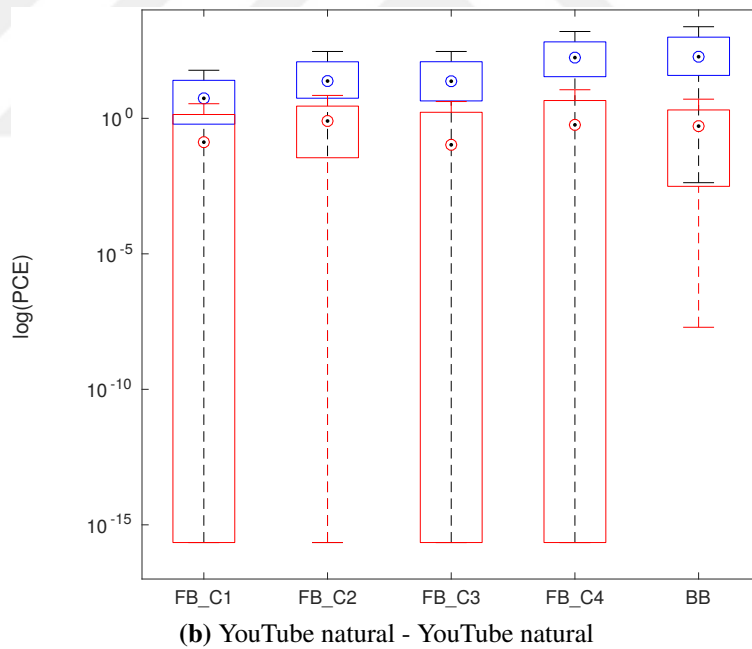
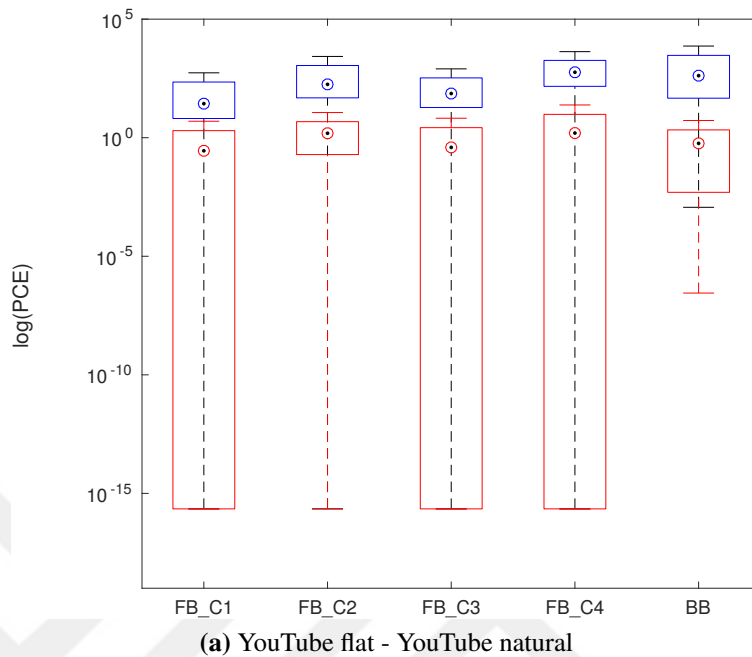
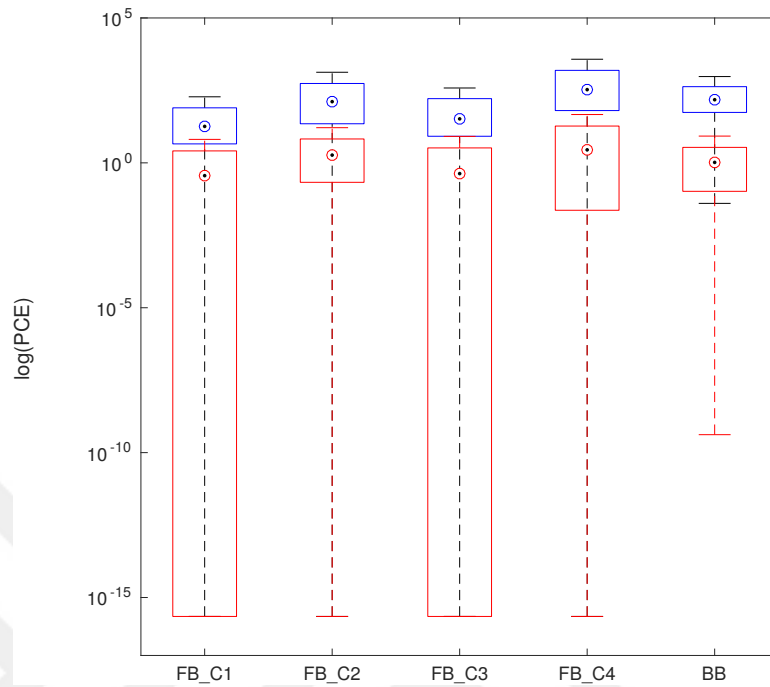
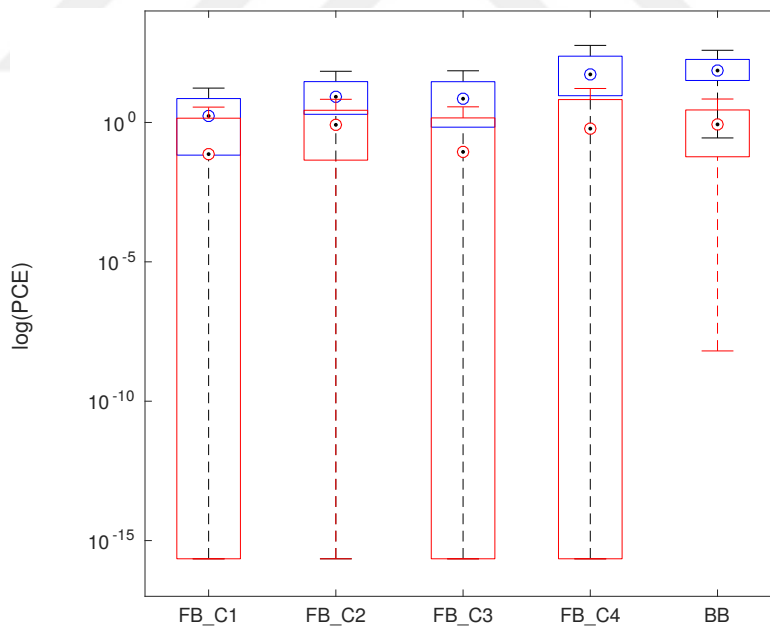


Figure 4.8. PCE distributions of 720p YouTube videos matched to YouTube videos concerning the type of frames which should be used to achieve the best accuracy with the least computing time when performing device source attribution for H.264/AVC videos :

- If we are to perform source device attribution for native videos (source identification



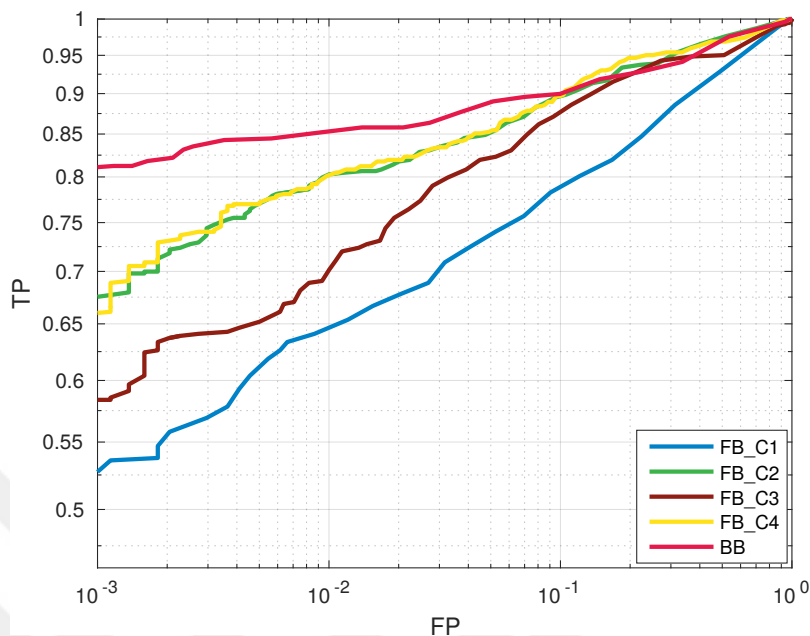
(a) YouTube flat - YouTube natural



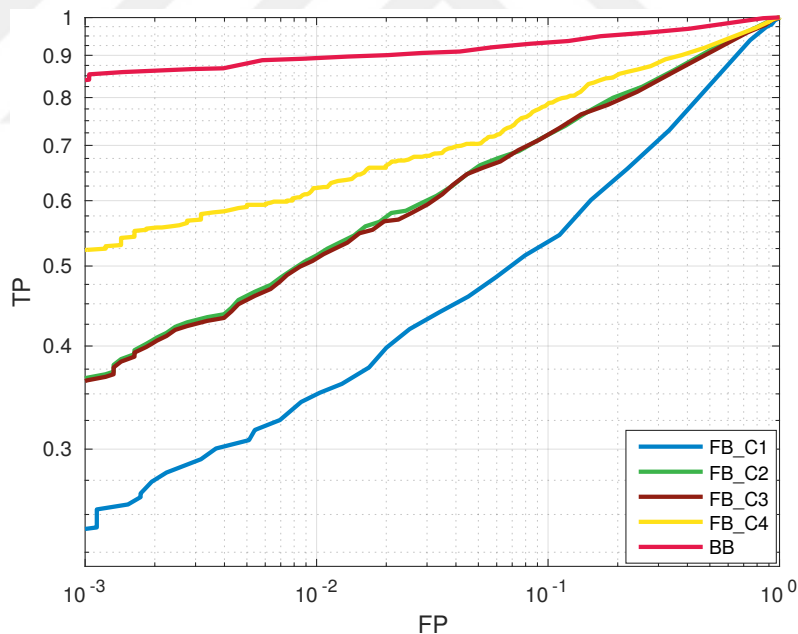
(b) YouTube natural - YouTube natural

Figure 4.9. PCE distributions of 1080p YouTube videos matched to YouTube videos

and device linking), using I frames only with the frame based approach for camera fingerprints and query video noise estimation achieves a high accuracy with the least



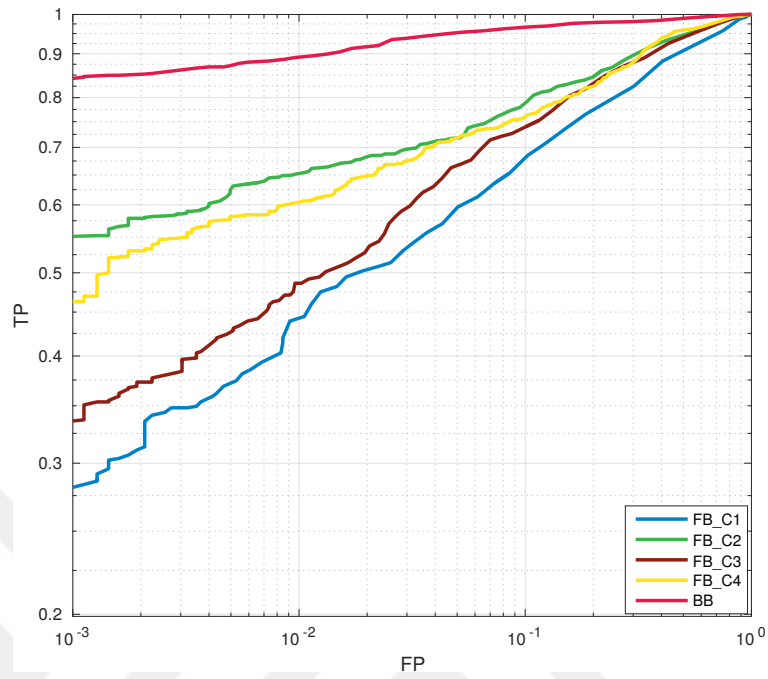
(a) YouTube flat - YouTube natural



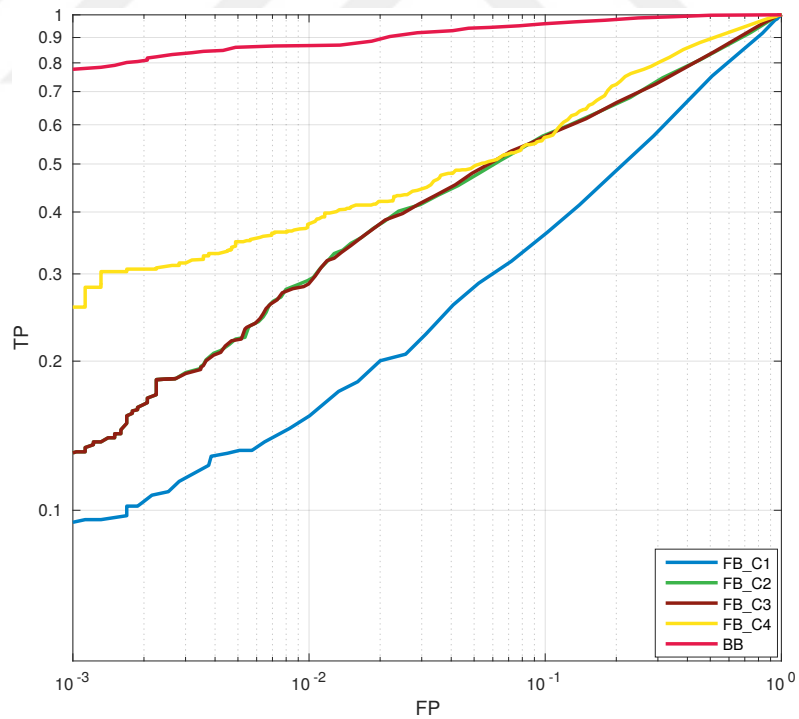
(b) YouTube natural - YouTube natural

Figure 4.10. ROC curves of 720p YouTube videos matched to YouTube videos computing time.

- When we are to match YouTube videos to native videos, using I frames to estimate fingerprint/noise from native videos and all the frames of query YouTube videos achieves



(a) YouTube flat - YouTube natural



(b) YouTube natural - YouTube natural

Figure 4.11. ROC curves of 1080p native videos matched to YouTube videos

Table 4.14. AUC values for each device when linking 1080p YouTube videos

| Device ID | YouTube flat - YouTube natural | | | | | YouTube natural - YouTube natural | | | | |
|-----------|--------------------------------|------|------|------|------|-----------------------------------|------|------|------|------|
| | C1 | C2 | C3 | C4 | BB | C1 | C2 | C3 | C4 | BB |
| D03 | 0.88 | 0.94 | 0.88 | 0.93 | 0.94 | 0.62 | 0.79 | 0.79 | 0.83 | 0.99 |
| D11 | 0.95 | 0.95 | 0.98 | 0.94 | 1.00 | 0.86 | 0.93 | 0.92 | 0.91 | 1.00 |
| D16 | 0.85 | 0.97 | 0.85 | 0.95 | 0.96 | 0.67 | 0.76 | 0.74 | 0.79 | 0.98 |
| D17 | 1.00 | 1.00 | 1.00 | 0.99 | 1.00 | 0.93 | 0.95 | 0.94 | 0.95 | 1.00 |
| D21 | 0.96 | 0.98 | 0.95 | 0.88 | 1.00 | 0.80 | 0.88 | 0.88 | 0.87 | 1.00 |
| D24 | 0.95 | 0.97 | 0.91 | 0.93 | 1.00 | 0.77 | 0.87 | 0.89 | 0.93 | 1.00 |
| D27 | 0.92 | 0.94 | 0.97 | 0.91 | 0.98 | 0.70 | 0.78 | 0.79 | 0.83 | 1.00 |
| D28 | 0.68 | 0.80 | 0.85 | 0.85 | 1.00 | 0.54 | 0.69 | 0.68 | 0.76 | 0.97 |
| D30 | 0.66 | 0.76 | 0.76 | 0.72 | 0.96 | 0.56 | 0.69 | 0.68 | 0.72 | 0.95 |
| D31 | 0.95 | 0.97 | 0.94 | 0.96 | 1.00 | 0.75 | 0.85 | 0.84 | 0.91 | 1.00 |

the best accuracy of the frame-based approach. As previously, the block-based approach gives the highest accuracy.

- If we are to link YouTube videos, using all frames will achieve the best accuracy of the block-based approach. Compared to the frame-based approach, the block-based approach is much more effective and brings a great improvement to the classification accuracy.

4.3 Source device attribution for digitally stabilized videos

For digitally stabilized videos, we consider three cases: the case where video stabilization is performed inside the camera, the case where video stabilization is performed outside the camera using FFmpeg, and, the case where video stabilization is performed outside the camera using YouTube Stabilizer (Grundmann et al. 2011). Because of the huge computing time requested by our methods, we will test each case using only one test video: A video from D15 (an exemplar of iPhone6) for the first case and one video from D03 (Huawei P9) for the last two cases. The video frames noise registration process is run on a cluster of computers which properties are given in Table 4.15. Table 4.16 gives the time it takes to register one

frame’s noise according to its resolution.

Table 4.15. Properties of computers in the cluster used for frames noise registration

| Property | Value |
|---------------------|---------------------------------|
| Number of computers | 30 |
| CPU | Intel Core i7-6700, 3.40GHz × 8 |
| RAM | 16 GB |
| OS | Ubunutu 16.04 LTS |
| MPI runtime | Open MPI 1.10.2 |

Table 4.16. Processing time for the registration of the PRNU noise in one frame

| Resolution | Computing time |
|------------|----------------|
| 720p | ≈ 20 min |
| 1080p | ≈ 45 min |

4.3.1 Source device identification for in-camera-stabilized videos

Here we consider the case where the suspect camera features digital video stabilization; in this case, camera fingerprint is obtained by scaling and cropping an image-based fingerprint as described in Section 2.1.3. Figure 4.12 gives the maximum PCE values obtained when matching a fingerprint estimated from 65 flat content images (acquired with the same exemplar of iPhone) with the PRNU noise of I frames from a flat content video (acquired with the same exemplar) acquired in still conditions. The best matched fingerprint corresponds to the one which gave the highest PCE value (which corresponds to the 34th I frame). Once a good camera fingerprint is obtained, it can be used to match a test video. The test video used in our experiment has been acquired in outdoor conditions with the user walking while capturing the video (D15_V_outdoor_move_0002.mov). The scheme proposed in Section 3.2.2 is then used to estimate affine transforms that register each of the first 25 I frames of the test video to the camera fingerprint. To check the efficiency of our affine transform estimation scheme, we give in Figure 4.13 PCE values of cross-correlations between the camera fingerprint and

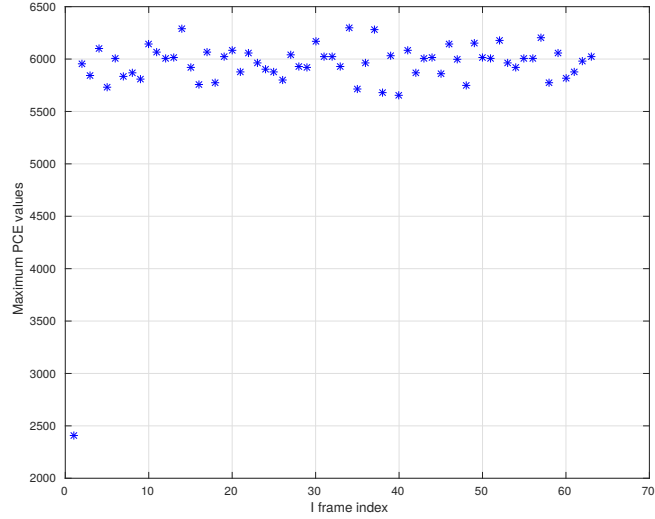


Figure 4.12. PCE values obtained when matching an image-based fingerprint to the PRNU noise extracted from I frames of a flat-content stabilized video (iPhone6)

non-registered frames noise and, cross-correlations between the camera fingerprint and registered frames noise. Figure 4.13 shows that our scheme registers frames noise in most cases.

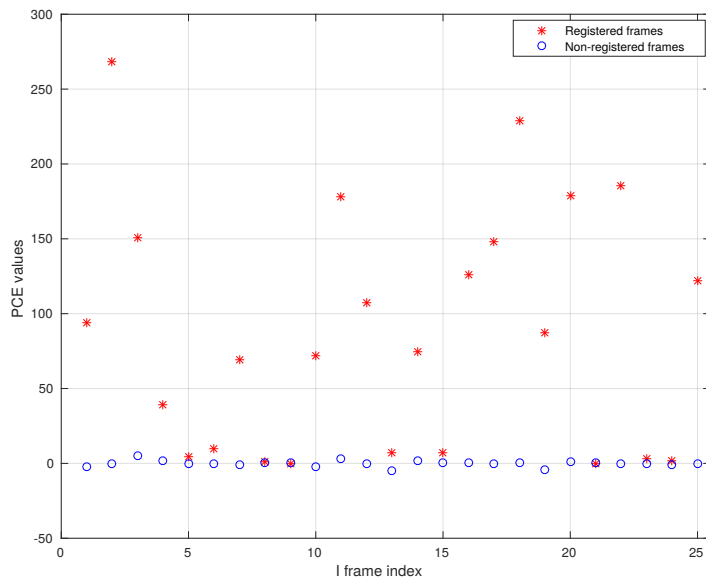


Figure 4.13. PCE values of non-registered and registered PRNU frames noise

Cases where high PCE values have not been obtained correspond to frames warped above the 40×40 pixels grid used as search space in our affine transform estimation scheme. For these

frames, the maximum peak values in normalized-cross-correlation matrices are not located in the vicinity of NCC matrices central points. For a correctly estimated affine transform matrix, the maximum peak value in the NCC matrix is located in a 40×40 square centered at the middle of the NCC matrix. Figure 4.14 illustrates this, it gives the NCC matrices for a correctly and a wrongly estimated matrix. Wrongly registered frames noise are eliminated during the registered frames noise aggregation process. Figure 4.15 gives the cross-correlation matrix

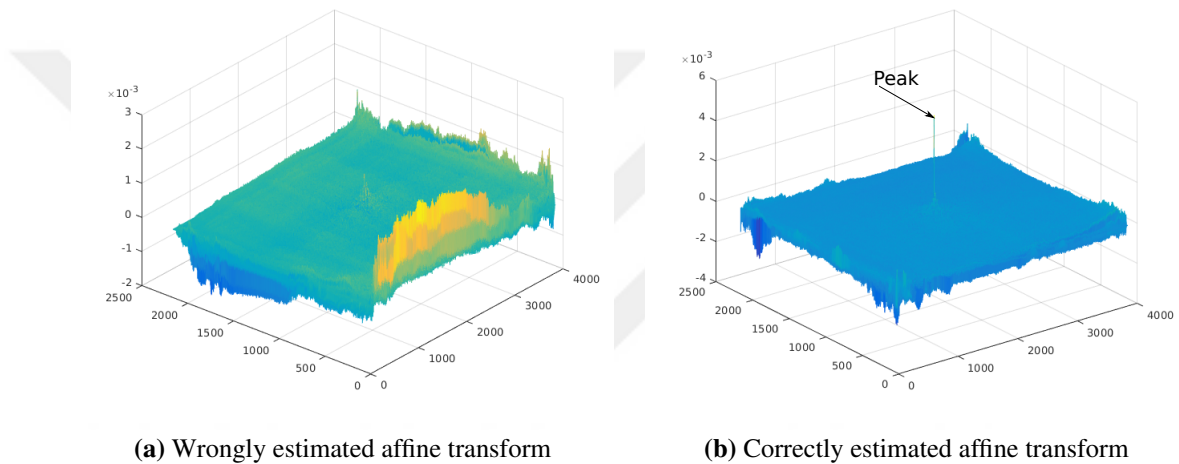


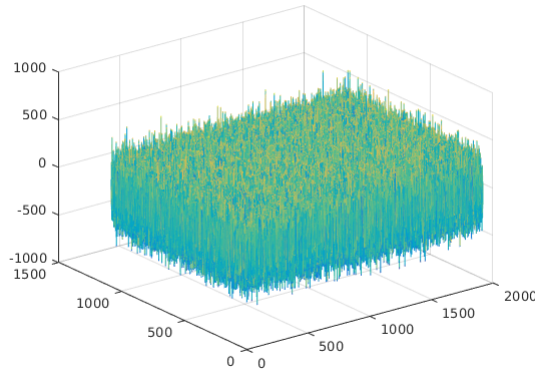
Figure 4.14. NCC corresponding to wrongly and correctly estimated affine transform

between the camera fingerprint and the video noise obtained by averaging non-registered frames, and, the cross-correlation matrix between the camera fingerprint and the video noise obtained after the aggregation of registered frames noise. Table 4.17 gives numerical results of the source device attribution process on the iPhone6 test video. Table 4.17 shows how

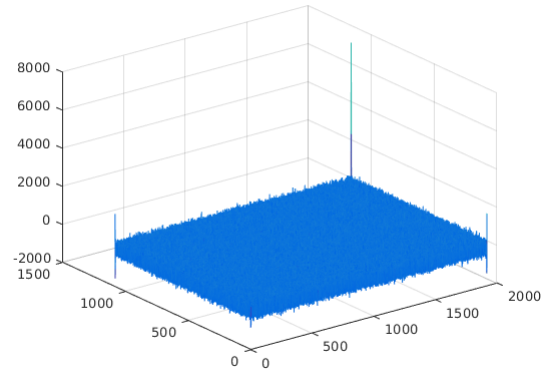
Table 4.17. iPhone6 test video results

| | Before registration | After registration |
|---------------------|---------------------|--------------------|
| Maximum Correlation | 762.88 | 7.45×10^3 |
| PCE Value | 0.015 | 1.73×10^3 |

efficient has been the registration process, we can see that from a PCE of 0.015 we could obtain after registration a PCE value of 1.73×10^3 .



(a) Cross-correlation before registration



(b) Cross-correlation after registration

Figure 4.15. Cross-correlation before and after frame noise registration (iPhone6 test video)

4.3.2 Source device identification for videos stabilized with *FFmpeg*

Here, we consider the case where video stabilization is performed outside the camera, the tool *deshake* of *FFmpeg* is used to stabilize the test video. The reference camera fingerprint is estimated from all the I frames of a flat content video. The test video is a shaky outdoor video. Table 4.18 gives the main properties of the video used for fingerprint estimation and the stabilized test video. Figure 4.16 gives the PCE values of non-registered and registered

Table 4.18. Properties of fingerprint and test videos used in the *FFmpeg* experiment

| Property | flat-content video | Stabilized test video |
|------------|---------------------------|-----------------------------|
| File | D03_V_flat_still_0002.mp4 | D03_V_outdoor_move_0002.mp4 |
| Resolution | 1080p | 1080p |
| # I frames | 69 | 10 |

frames noise. Figure 4.17 gives the cross-correlation matrix between the camera fingerprint and the video noise from non registered frames noise, and, the cross-correlation matrix between the camera fingerprint and the video noise from registered frames noise. Numerical results are given by Table 4.19. Here again our frame noise registration scheme works pretty well and improve the PCE of all the 10 I frames. Compared to the previous case, we obtain a smaller PCE value, this is due to the limited number of I frames in the test video and also

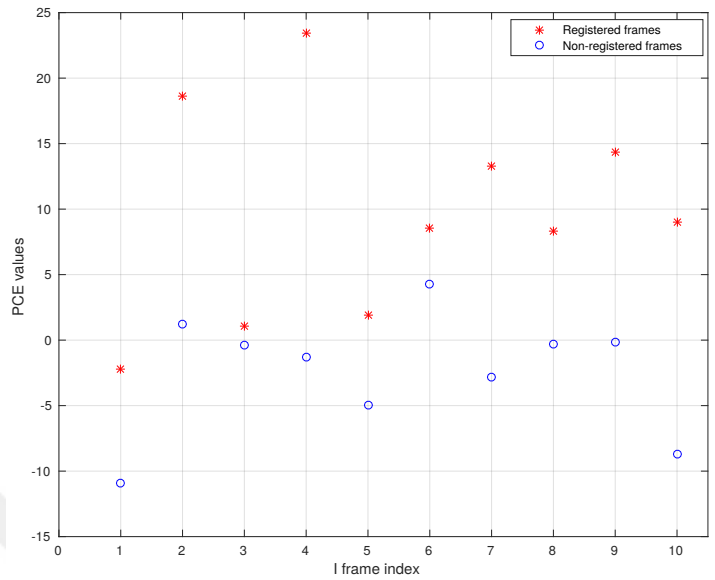
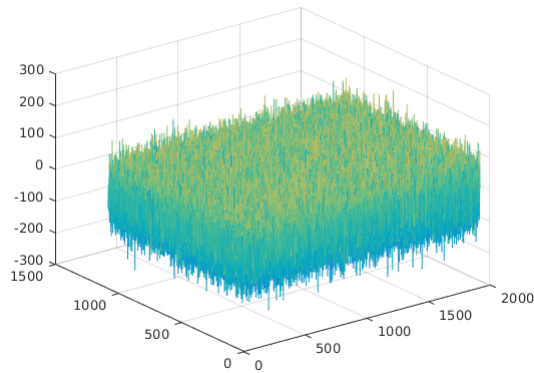


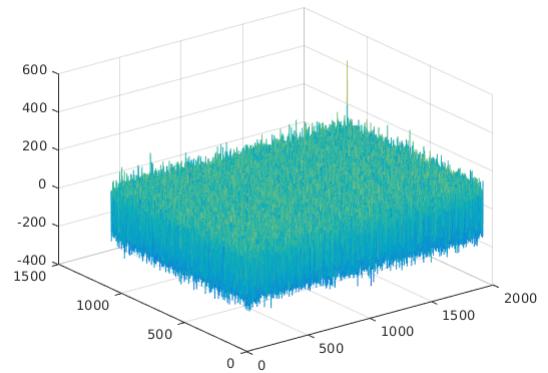
Figure 4.16. PCE of non-registered and registered frames noise (video stabilized with FFmpeg)

Table 4.19. FFmpeg-deshake test video results

| | Before registration | After registration |
|---------------------|---------------------|--------------------|
| Maximum Correlation | 240.06 | 461.72 |
| PCE Value | -9.37 | 72.28 |



(a) Cross-correlation before registration



(b) Cross-correlation after registration

Figure 4.17. Cross-correlation before and after frame noise registration (FFmpeg-deshake test video)

because FFmpeg re-encodes the video after stabilization.

4.3.3 Source device identification for videos stabilized with YouTube stabilizer

Finally, we consider the case where the query video has been stabilized using YouTube Stabilizer which is a module of YouTube studio. YouTube stabilizer uses the algorithm described in (Grundmann et al. 2011). This case presents two level of difficulties: the destruction of the PRNU due to an aggressive compression, and, the frames noise misalignment due to video stabilization. The same query video used in the previous experiment is uploaded to YouTube, stabilized with YouTube stabilizer, and, downloaded. Table 4.20 gives the properties of the flat-content video used to estimate the camera fingerprint, and, the properties of the test video. YouTube downscales the resolution of the stabilized video from 1080p to 720p, thus, the reference fingerprint is also scaled to this resolution using the nearest neighbor interpolation. Figure 4.18 gives PCE values of registered and non registered frames noise. Figure 4.19 gives plots of cross-correlation matrices before and after frames noise registration. Numerical results on the YouTube-stabilized query video are given in Table 4.21.

Table 4.20. Properties of fingerprint and test videos used in the Youtube-Stabilizer experiment

| Property | flat-content video | Stabilized test video |
|------------|---------------------------|-----------------------------|
| File | D03_V_flat_still_0002.mp4 | D03_V_outdoor_move_0002.mp4 |
| Resolution | 1080p | 720p |
| # I frames | 69 | 37 |

Table 4.21. YouTube-stabilized test video results

| | Before registration | After registration |
|---------------------|---------------------|--------------------|
| Maximum Correlation | 65.01 | 296.82 |
| PCE Value | 1.17 | 231.27 |

Results presented in this section show the effectiveness of our proposed scheme for source de-

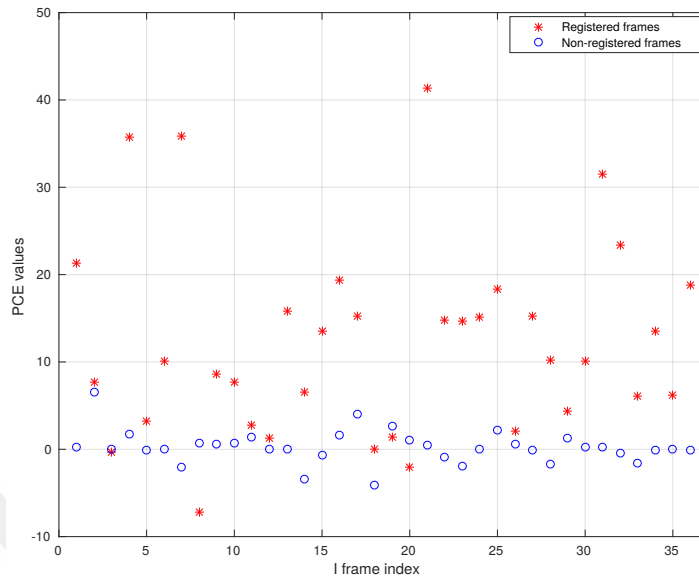


Figure 4.18. PCE of non-registered and registered frames noise (video stabilized with YouTube Stabilizer)

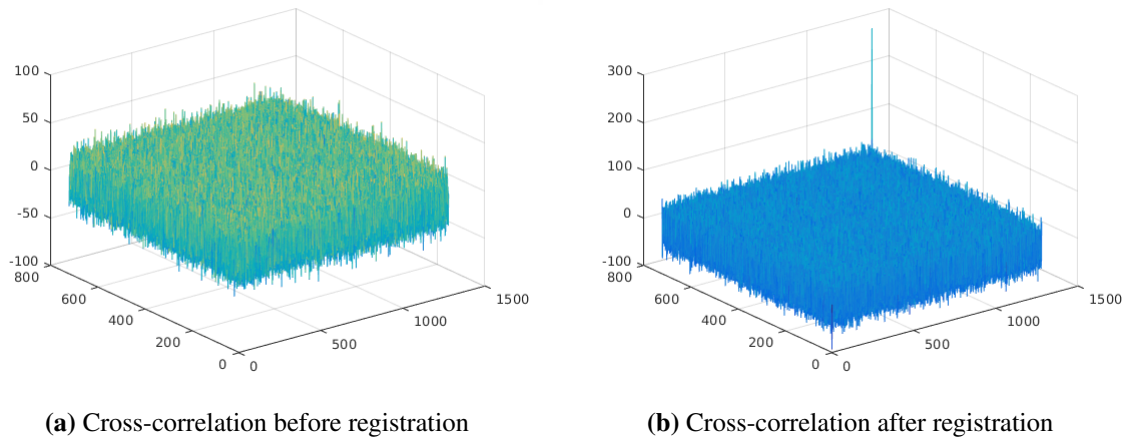


Figure 4.19. Cross-correlation before and after frame noise registration (YouTube-stabilized test video)

vice attribution of digitally stabilized videos. Through the three cases experimented, we prove that our proposed scheme can efficiently estimate geometric transforms applied to frames of a digitally stabilized video and aggregate the PRNU noise from registered frames even when the query video has been highly compressed.

In this chapter, we presented experimental results obtained using our proposed methods for

source device attribution. A large set of non-stabilized videos was used to determine the set of frames which should be used to achieve the best accuracy with the least computing time. Concerning our scheme for digitally-stabilized videos, we prove its effectiveness by testing it in three conditions: the case where the stabilization is performed inside the camera, the case where stabilization is performed using FFmpeg and, the case where YouTube Stabilizer is used (which implies re-compression and stabilization).



5. DISCUSSION AND CONCLUSION

In this research work, we focused on source device attribution for digital videos. On the basis of well established PRNU-based state of the art techniques for source device attribution of digital images, we propose novel schemes to perform source device attribution for digital videos.

In the first part of our research, we studied the effects of video compression on the PRNU noise in video frames. It comes out that operations involved in video compression (prediction, transform and quantization) can destroy the PRNU noise in encoded frame block (the PRNU noise in the encoded block is replaced by the one in its prediction block(s)). This happens when the DCT transform of the block's prediction residue has no AC component; in this case, the PRNU noise in the encoded block is replaced by the one in its prediction block(s).

Knowing the effect that video compression has on PRNU noise in video frames, we propose two approaches to estimate fingerprint and noise from non-stabilized videos: a frame-based approach and a block-based. The frame-based approach aims to determine the set of frames which should be used in order to achieve the highest accuracy meanwhile in the block-based approach, we filter blocks used in fingerprint and noise estimation to use only block in which there still remaining PRNU noise.

It comes out from our experiment that:

- Using I frames only with the frame-based approach is enough to achieve nearly 100% of accuracy when performing source device attribution for native videos.
- If we are to match native videos to YouTube videos, an accuracy of nearly 100% is achieved with the frame-based approach using I frames to estimate fingerprints from native videos and using all video frames to estimate YouTube videos' noise.
- When YouTube videos are to be matched, the best accuracy using the frame-based

approach (which is really low) is obtained using all the frames in query videos.

- The block-based approach gives high classification accuracies even when linking highly compressed YouTube videos.
- The results given here represent the lower bound since more videos can be combined to estimate a better fingerprint.

Digital video stabilization is more and more used in hand-held devices. This camera feature misaligns the PRNU noise in video frame and makes video noise estimation more complicated. We propose a novel scheme to register the PRNU noise in stabilized video frames to a fingerprint estimated from still images or video frames. The efficiency of this scheme has been tested in various conditions and gave satisfying results even when the stabilized video have been highly compressed.

The research presented in this thesis is far from being complete, but instead, represents the foundation for future researches. These future researches will focus on :

- Use state-of-the-art techniques such as MACE (Minimum Average Correlation Energy) filter to improve the PRNU fingerprint estimated with the block-based approach in order to improve the accuracy of the block-based approach when linking highly compressed videos.
- Use deep learning to remove frames' high frequency content from the estimated frame PRNU noise.
- Implement the affine transform estimation algorithm on GPUs in order to accelerate computations. Since the algorithm computes a lot of cross-correlations which are implemented using FFT, a great speed up could be achieved if these operations are moved from CPUs to GPUs.
- Test our scheme for source device attribution of digitally stabilized videos on a large

set of videos.

All these aspects will be addressed in future researches in our research team.



REFERENCES

- Adrian ,K., Gary, B. 2017.** Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library. O Reilly Media., USA, 990 pp.
- Alexa 2018.** Alexa internet. <http://www.alexa.com/topsites> (10/05/2018).
- Bayram, S., H. Sencar, N. Memon, and I. Avcibas. 2005.** Source camera identification based on cfa interpolation. In IEEE International Conference on Image Processing 2005.
- BBC 2016.** South korea's hidden camera-hunting squad. <http://www.bbc.com/news/blogs-trending-37911695> (10/05/2018).
- Chang, H.-C., S.-H. Lai, and K.-R. Lu. 2006.** A robust real-time video stabilization algorithm. *Journal of Visual Communication and Image Representation*, 17(3):659 – 673.
- Chen, M., J. Fridrich, M. Golja, and J. Lukáš 2007.** Source digital camcorder identification using sensor photo response non-uniformity. In Proceedings of SPIE - *The International Society for Optical Engineering* 6505, volume 6505.
- Choi, K. S., E. Y. Lam, and K. K. Y. Wong 2006.** Feature selection in source camera identification. In 2006 IEEE International Conference on Systems, Man and Cybernetics, volume 4, Pp. 3176–3180.
- Dasara, S., F. Marco, I. Massimo, S. Omar, and P. Alessandro 2017.** Vision: a video and image dataset for source identification. *EURASIP Journal on Information Security*, 2017(1):15.
- Farid, H. 2006.** Digital image ballistics from jpeg quantization. <http://www.ists.dartmouth.edu/library/204.pdf> (22/02/2018).
- FFmpeg 2018.** Ffmpeg. <https://www.ffmpeg.org/> (10/01/2018).

- Filler, T., J. Fridrich, and M. Goljan 2008.** Using sensor pattern noise for camera model identification. In 2008 15th IEEE International Conference on Image Processing, Pp. 1296-1299.
- Fridrich, J. 2009.** Digital image forensics. *IEEE Signal Processing Magazine*, 26(2):26-37.
- Goljan, M., M. Chen, and J. Fridrich 2007.** Identifying common source digital camera from image pairs. In 2007 IEEE International Conference on Image Processing, volume 6, Pp. VI-125-VI-128.
- Goljan, M. and J. Fridrich 2008.** Camera identification from cropped and scaled images. *In Proceedings of SPIE -The International Society for Optical Engineering.*
- Goljan, M., J. Fridrich, and T. Filler 2009.** Large scale test of sensor fingerprint camera identification. *In Proceedings of SPIE- The International Society for Optical Engineering.*
- Grundmann, M., V. Kwatra, and I. Essa 2011.** Auto-directed video stabilization with robust 11 optimal camera paths. In *CVPR 2011*, Pp. 225–232.
- Harvey, P. 2018.** Exiftool. <https://metacpan.org/pod/exiftool> (01/03/2018).
- Hlmg, L., K. Somasekhar Reddy, and S. 2014.** A survey on video stabilization algorithms.
- Houten, W. V. and Z. Geradts 2009.** Source video camera identification for multiply compressed videos originating from YouTube. *Digital Investigation.*
- Hyun, Dai-Kyung, Choi, Chang-Hee, Lee, and Heung-Kyu 2012.** Camcorder identification for heavily compressed low resolution videos. *In Computer Science and Convergence: CSA 2011 & WCC 2011 Proceedings*, Pp. 695–701.
- Itseez 2017.** Open source computer vision library. <https://github.com/itseez/opencv>.
- Iuliani, M., M. Fontani, D. Shullani, and A. Piva 2017.** A hybrid approach to video source identification. *CoRR*, abs/1705.01854.

JVT 2016. jm 16.1 source code. http://iphome.hhi.de/suehring/tml/download/old_jm/
(07/08/2017)

Kurosawa, K., K. Kuroki, and N. Saitoh 1999. Ccd fingerprint method-identification of a video camera from videotaped images. In Proceedings 1999 International Conference on Image Processing (Cat. 99CH36348), volume 3, Pp. 537–540 vol.3.

Li, C. T. 2010. Source camera identification using enhanced sensor pattern noise. *IEEE Transac-tions on Information Forensics and Security*, 5(2):280–287.

Lim, A., B. Ramesh, Y. Yang, C. Xiang, Z. Gao, and F. Lin 2017. Real-time optical flow-based video stabilization for unmanned aerial vehicles. *Journal of Real-Time Image Processing*.

Lukas, J., J. Fridrich, and M. Goljan 2005. Determining digital image origin using sensor imperfections.

Lukas, J., J. Fridrich, and M. Goljan 2006. Digital camera identification from sensor pattern noise. *IEEE Transactions on Information Forensics and Security*, 1(2):205–214.

Marpe, D., T. Wiegand, and G. J. Sullivan 2006. The h.264/mpeg4 advanced video coding standard and its applications. *IEEE Communications Magazine*, 44(8):134–143.

Matsushita, Y., E. Ofek, W. Ge, X. Tang, and H.-Y. Shum 2006. Full-frame video stabilization with motion inpainting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(7):1150–1163.

Mihcak, M. K., I. Kozintsev, and K. Ramchandran 1999. Spatially adaptive statistical modeling of wavelet image coefficients and its application to denoising. In 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258), volume 6, Pp. 3253–3256 vol.6.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery 1992. Numerical Recipes in C: The Art of Scientific Computing. New York, NY, USA: Cambridge University Press.

Rafael C. Gonzalez, Richard E. Woods, S. L. E. 2009. Digital Image Processing Using MATLAB, 2nd edition edition. Gatesmark Publishing.

Richardson and I. E. 2010. The H.264 Advanced Video Compression Standard. USA: John Wiley & Sons, Ltd.

Su, Y., J. Xu, and B. Dong 2009. A source video identification algorithm based on motion vectors. In 2009 Second International Workshop on Computer Science and Engineering, volume 2, Pp. 312–316.

Taspinar, S., M. Mohanty, and N. Memon 2016. Source camera attribution using stabilized video. In 2016 IEEE International Workshop on Information Forensics and Security (WIFS), pp. 1–6.

Open MPI Development Team. 2017. Open mpi: Open source high performance computing. <https://www.open-mpi.org/> (09/01/2018).

Van, L. T., S. Emmanuel, and M. S. Kankanhalli 2007. Identifying source cell phone using chromatic aberration. In 2007 IEEE International Conference on Multimedia and Expo, Pp. 883–886.

Villalba, L. J. G., A. L. S. Orozco, R. R. López, and J. H. Castro 2016. Identification of smartphone brand and model via forensic video analysis. *Expert Systems with Applications*.

CURRICULUM VITAE

Full Name : Emmanuel Kiegaing Kouokam
Birth Place and Date : Yaounde (Cameroon), 1987
Languages : French, English, Turkish, German
Education Status (Institute and Year) :
High school : Government High School Garoua 2005
Bachelor : University of Douala 2010
Master Of Science : University of Douala 2012
Workplaces and Years :
University of Douala : 2012-2014
IETR Rennes (France) : 2016-2017
E-mail : kiegainemmanuel@gmail.com