Tampering Detection In Low-Power Smart Cameras

Adriano Gaibotti[1], Claudio Marchisio[1], Alexandro Sentinelli[1], and Giacomo Boracchi[2]

[1] STMicroelectronics, Advanced System Technology, Via Camillo Olivetti 2, 20864, Agrate Brianza (MB), Italy
{adriano.gaibotti, claudio.marchisio, alexandro.sentinelli}@st.com
[2] Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Via Ponzio 34/5, 20133, Milano (MI), Italy
giacomo.boracchi@polimi.it

Abstract. A desirable feature for smart cameras is the ability to autonomously detect any tampering event/attack that would prevent a clear view over the monitored scene. No matter whether tampering is due to atmospheric phenomena (e.g., few rain drops over the camera lens) or to malicious attacks (e.g., the device displacement), these have to be promptly detected to possibly activate countermeasures. Tampering detection becomes particularly challenging in battery-powered cameras, where it is not possible to acquire images at video-like frame-rates, nor use sophisticated image-analysis algorithms.
We here introduce a tampering-detection algorithm that has been specifically designed for low-power smart cameras: the algorithm leverages very simple indicators that are then monitored by an outlier-detection scheme. Any frame yielding anomalous indicator is detected as a tampering attempt. Core of the algorithm is the partitioning of the scene into adaptively defined regions, that are preliminarily defined by segmenting the image during the algorithm-configuration phase, and which shows to substantially improve the detection of camera displacements. Our experiments show that the proposed algorithm can successfully operate on sequences acquired at very low-frame rate, such as one frame every minute, and at a very small computational complexity.

Keywords: tampering detection, defocus, displacement detection

1 Introduction

When cameras operate outdoor and in harsh environments, dust, rain drops or snow flakes might lie on the camera lens resulting in blurry pictures, as in Fig. 1(a), or in partial occlusions of the scene, as in Fig. 1(b). Similarly, intentional attacks like displacing the camera, changing its focus, or spraying some opaque or glossy liquid over the lenses, would result in heavily compromised pictures, that would be surely useless for monitoring purposes. We refer to these events/attacks as tampering. In some cases, tampering is easy to detect, e.g.,

when the camera integrity is affected and the device goes out-of-order. However, in many other situations, namely when the device is not physically damaged, tampering detection is not straightforward, and sometimes image analysis is the only viable option.

Tampering prevents the correct scene interpretation and causes the loss of small, important details such as licence plates. Tampering detection is therefore an essential feature in surveillance systems [3], where cameras are expected to autonomously detect any tampering, and promptly reporting alerts. Surveillance cameras are typically connected to the power supply and operates at normal frame-rates (e.g. around few frames per second): in these conditions, several tampering-detection algorithms have been presented [**Giacomo**: Adriano add REFERENCES].

[**Giacomo**: Qui ST puó aggiungere qualche esempio di applicazione qui o menzionare dispositivi di riferimento (con tanto di link a datasheet). Magari aggiungendo che le batterie recenti permettono operativitá di due anni a questi regimi.][**Claudio**: Verifico cosa possiamo dire del SecSoc.] In this work we expressly target low-power and ultra-low-power smart cameras, devices that operate at very low frame-rates (e.g. possibly less than one frame every minute), that are characterized by a constrained computational power and memory, and that are battery powered (typically lasting one or two years). As an example, consider Wireless Multimedia Sensor Networks (WMSN) [4] where nodes are wirelessy connected smart-cameras, that can acquire and transmit images at regular time interval or upon requests. Often, in WMSN, the units are not connected to the power supply and have to operate with batteries and possibly rely on energy harvesting mechanisms [**Giacomo**: REFERENCES, Adriano, guarda se c'e' qualche reference dal nostro paper di TIM]. Therefore, the power consumption is a serious concern in these devices.

Even though low-power and ultra-low-power cameras not employed in critical surveillance applications, these devices are becoming popular in distributed systems for monitoring wide environments due to their low cost and maintenance requirements [**Giacomo**: Adriano, Claudio, possiamo mettere qualche REFERENCE?]. Tampering detection is therefore very important in low-power smart cameras, where the algorithm are expected to be particularly prompt and have low false-alarm rates to prevent unnecessary energy-demanding operations like the local processing or radio activation for transmitting corrupted images (or false alarms) over the network. Unfortunately, tampering detection in such a resource-constrained scenario, has been much less investigated.

Tampering detection in low-power smart cameras is much more challenging than in conventional surveillance cameras [5]. Beside computational aspects – such as the number of operations per pixels allowed – the big issue is that low-power smart cameras typically operate at very low-frame rates (e.g., less than one frame per minute), thus the acquired sequence does not evolve smoothly. This prevents the use of learned background models and the analysis of foreground variations [6]. When dynamic environments are acquired at low frame rates, two consecutive frames might be very different because of changes in the scene and

(a) (b)

Fig. 1. Examples of tampering events due to atmospheric phenomena. **(a)** Blur due to rain drops on the camera lens. **(b)** Occlusion due to snow on the camera lens.

in the light conditions, resulting in sequences those depicted in Figure 2: smart cameras have to correctly distinguish between these *normal* changes and changes due to camera tampering.

We present an algorithm to detect both camera defocus and displacement, as stated in Section 2. The algorithm relies on two simple indicators that can be easily computed (Section 3.2) and that are monitored by an outlier-detection technique to detect tampering (Section 3.3). In particular, we show that tampering-detection performance can be improved by performing a preliminarily partitioning of the scene into nonoverlapping regions, and then separately computing and monitoring indicators over different regions (Section 3.1). Remarkably, these regions need to be defined during an initial configuration phase of the smart camera, thus there is no relevant computational overhead w.r.t. monitoring the whole image. Our experiments in Section 4 show that operating on image regions can substantially improve the detection of camera displacements. Concluding remarks and discussions are given in Section 5.

1.1 Related Works

The literature concerning tampering detection is mostly focused on video surveillance applications and operates at few frames per second [**Giacomo**: Adriano, é vero?]. Background models are typically leveraged to identify defocus and occlusions; in particular [7] performs detect defocus by analyzing the wavelet domain of each frame and performs histogram comparison for detecting occlusions, while camera displacements are not considered. A background-subtraction technique is employed in [8] to identify defocus, occlusions, and displacements. [**Giacomo**: FIXME Comparison in the Fourier domain for defocus detection, histogram comparison for occlusion detection, comparison between current background and delayed background for displacement detection]. Background subtraction is also used in [9] to detect defocus, occlusions, and displacements. [**Giacomo**: FIXME Comparison of edges pixels count for defocus detection, entropy comparison for occlusion

detection, block matching algorithm for displacement detection]. In contrast, no background models are used in [10], where a sequential monitoring scheme based on a change-detection test is employed to detect changes in the average gradient energy of each frame to detect defocus due to external disturbances on the camera lens. [11]: comparison between frames belonging to a buffer in order to find high values of dissimilarity, associated to tampering. [12]: implementation in a FPGA of a solution based on background modeling, histograms comparisons, edges comparisons. [13]: tampering detection inside a moving vehicle; uses background subtraction methods in order to identify defocus, occlusions, and displacements. Comparison of edges pixels count for defocus detection, entropy comparison for occlusion detection, block matching algorithm for displacement detection. [14]: monitoring of the number of key points extracted by SURF in order to detect defocus events, partition in blocks and HOG descriptors matching for each block in order to detect occlusions. These types of solutions requires a lot of computations

2 Problem Formulation

Let z_t be the frame acquired at time t, which we model as

$$z_t(x) = \mathcal{D}_t[y_t](x) \quad \forall x \in X \tag{1}$$

where $\mathcal{D}_t$ denotes the degradation operator that transforms the original image y_t in the frame z_t; $X \subset \mathbb{Z}^2$ denotes the regular pixels grid and $x \in \mathbb{Z}^2$ indicates the pixel coordinates. As far as there is no tampering attacks/events,

$$\mathcal{D}_t[y_t](x) = y_t(x) + \eta_t(x) \quad \forall x \in X \tag{2}$$

where η_t is a random variable accounting for different noise sources (e.g., thermal, quantization, photon-counting). In normal conditions, all the images y_t (thus also the frames z_t) might show different content but are acquired from the same viewpoint and the same camera orientation.

When at time τ^* an external disturbance introduces *blur/defocus*, the image y_t is degraded by a spatially variant blur operator, and z_t becomes

$$\mathcal{D}_t[y_t](x) = \int_{\mathcal{X}} y(s) h_t(x, s) ds \; + \eta_t(x) \quad \forall x \in X, t \geq \tau^* \tag{3}$$

where $h_t(x, \cdot) > 0$ is the point-spread function at pixel $x \in X$. A *camera displacement* at frame τ^* is instead modeled as

$$z_t(x) = \begin{cases} y_t(x) + \eta(x) \text{ per } t < T^* \\ w_t(x) + \eta(x) \text{ per } t \geqslant T^* \end{cases}, \tag{4}$$

where w_t relates to a different viewpoint and/or camera orientation than y_t.

The proposed tampering-detection algorithm analyzes a sequence of frames $\{z_t\}_t$ to detect time τ^* when any tampering like (3) or (4) occurs.

Algorithm 1: Tampering detection algorithm

Input: γ, T_o, $\{R_k\}, k = 1, \ldots, K$
Configuration:
1. **for** $t = 1, \ldots, T_o$ **do**
2. Get frame z_t
3. **for** $k = 1, \ldots, K$ **do**
4. Compute $l^k(t)$, $\partial l^k(t)$ for the region R_k
end
end
5. **for** $k = 1, \ldots, K$ **do**
6. Compute σ_l^k
end
Operational phase:
7. **for** $t = T_o, \ldots, \infty$ **do**
8. Get frame z_t
9. $n_l = 0$
10. **for** $k = 1, \ldots, K$ **do**
11. Compute $l^k(t)$, $\partial l^k(t)$ for the region R_k
12. **if** $\partial l^k(t) < -\gamma\sigma_l^k \vee \partial l^k(t) > \gamma\sigma_l^k$ **then**
13. $n_l = n_l + 1$
end
end
14. **if** $n_l \geq K - 1$ **then**
15. z_t is a tampered frame
end
end

3 The Proposed Algorithm

The proposed tampering-detection algorithm relies on two indicators that are very easy to compute: the average intensity values (also refer to *luma* and denoted by l) and the average image gradient (denoted by ∇z)

3.1 Scene Segmentation

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

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

dove:

- $r(x)$ rappresenta il numero di riga del pixel x.
- $c(x)$ rappresenta il numero di colonna del pixel x.

– $\mu_\nabla(x)$ rappresenta il valore del gradiente nel pixel x mediato nel tempo:

$$\mu_\nabla(x) = \frac{\sum_{t=1}^{T_c}(\|\nabla z_t\|_2^2 \circledast f)(x)}{T_c}, \tag{6}$$

dove abbiamo indicato con $\|\nabla z_t\|_2^2$ la norma del gradiente per l'immagine z_t, definita in (11), e con f il filtro gaussiano discreto derivato dal campionamento di (10).

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

$$\sigma_\nabla(x) = \sqrt{\frac{1}{T_c - 1}\sum_{t=1}^{T_c}\left(\left(\|\nabla z_t\|_2^2 \circledast f\right)(x) - \mu_\nabla(x)\right)^2}. \tag{7}$$

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

$$\bar{z}(x) = \frac{\sum_{t=1}^{T_c}(z_t \circledast f)(x)}{T_c}. \tag{8}$$

3.2 Indicators

$$\begin{aligned} g^k(t) &= \mathcal{G}^k[z_t] = \frac{\sum_{R_k}\|\nabla z_t(x)\|_2^2}{|R_k|} \\ \partial g^k(t) &= g^k(t) - g^k(t-1) \end{aligned}, \tag{9}$$

In particolare, per il calcolo delle derivate orizzontali abbiamo utilizzato il seguente filtro f_h:

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

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

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

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

$$h(i,j) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{i^2+j^2}{2\sigma^2}\right), \tag{10}$$

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

$$\|\nabla z_t(x)\|_2^2 = \left(z_t \circledast f_h\right)(x)^2 + \left(z_t \circledast f_v\right)(x)^2. \tag{11}$$

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

$$\begin{array}{l} l^k(t) \ = \mathcal{L}^k[z_t] = \frac{\sum_{R_k} z_t(x)}{|R_k|} \\ \partial l^k(t) = \ \ l^k(t) - l^k(t-1) \end{array}, \tag{12}$$

FRAME DIFFERENCE:

$$\varphi^k(t) = \frac{\sum_{x \in R_k} (z_t(x) - z_{t-1}(x))^2}{|R_k|}, k = 1, \ldots, K. \tag{13}$$

3.3 Outlier Detection

$$\begin{array}{l} \Gamma^k_{min} = \hat{\mu}^k_g - \gamma \hat{\sigma}^k_g \\ \Gamma^k_{max} = \hat{\mu}^k_g + \gamma \hat{\sigma}^k_g \end{array}, \tag{14}$$

dove $\hat{\mu}^k_g$ indica il valore medio delle osservazioni del training set

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

$\hat{\sigma}^k_g$ indica la deviazione standard delle osservazioni del training set

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

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

$$\begin{array}{l} \Gamma^k_{min} = \hat{\mu}^k_l - \gamma \hat{\sigma}^k_l \\ \Gamma^k_{max} = \hat{\mu}^k_l + \gamma \hat{\sigma}^k_l \end{array}, \tag{15}$$

dove $\hat{\mu}^k_l$ indica il valore medio delle osservazioni del training set

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

$\hat{\sigma}^k_l$ indica la deviazione standard delle osservazioni del training set

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

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

Fig. 2. Sequences taken from webcams: **(a)** no tampering events; **(b)** defocus event on 4-th and 5-th frames, created using gaussian filtering; **(c)** displacement event on 4-th and 5-th frames, created using concatenation between similar sequences; **(d)** real displacement event on 4-th frame.

3.4 Algorithm Summary

4 Experiments

The proposed algorithm has been implemented in MATLAB, and has been tested on two datasets. The first one refers to frame sequences taken from webcams recording some parts of cities (as in Figure 2(a)), where we have introduced some synthetically generated tampering events: defocus has been simulated using gaussian filters (2(b)), while displacement has been created with concatenation of similar frame sequences (Figure 2(c)). In some cases these sequences contained real tampering events, as in Figure 2(d).

The second dataset was taken using a Raspberry Pi Model B+, with its camera module, and the tampering was introduced by moving the device or putting water on the camera.

Four figures of merit have been suggested to assess the performance of the proposed algorithm:

TP True positives. It measures the number of tampering events correctly detected by the algorithm.

TN True negatives. It measures the number of frames without tampering that are not detected by the algorithm.

FP False positives. It measures the number of tampering events erroneously detected by the algorithm.

FN False negatives. It measures the number of tampering events not detected by the algorithm.

These indicators are computed varying the parameter γ, which defines the thresholds for the one-shot monitoring. This permits us to generate *ROC curves*, where on the x-axis there is:

$$1 - \text{SPECIFICITY}_\gamma = 1 - \frac{\text{TN}_\gamma}{\text{TN}_\gamma + \text{FP}_\gamma} = \frac{\text{FP}_\gamma}{\text{TN}_\gamma + \text{FP}_\gamma},$$

while on th y-axis there is:

$$\text{RECALL}_\gamma = \frac{\text{TP}_\gamma}{\text{TP}_\gamma + \text{FN}_\gamma}.$$

The construction of these curves permits us to make a comparison with respect to other methods. In particular the alternative approaches that we have considered working:

- considering the whole scene for the features computation;
- considering adaptive region, as described in Section 3, for the feature computation;
- considering voronoi regions [15], which are easier to compute with respect to our solution but don't consider the scene content, for the feature computation.

Experimental results are shown in Figures 4 and 4. As we could see there is an improvement, in the detection of camera displacements, with our approach which separately analizes the behavior of indicators in adaptive regions. On the other side, as illustrated in Figure 4, when monitoring indicators meant to detect blur/defocus, it is more convenient to consider the whole scene at once.

4.1 Discussion

[**Giacomo**: Adriano: Aggiungi qua la complessitá computazionale]

The algorithm uses low computational techniques: the biggest effort is made by computation of the feature $g(t)$, which needs 34 operations per pixel, due to the Sobel filter used during the convolution.

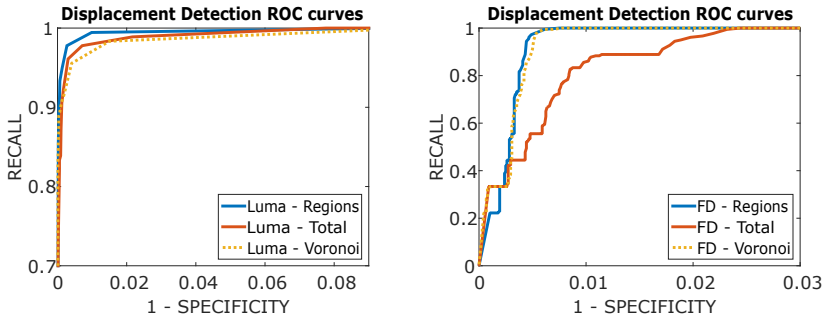

Fig. 3. ROC curves for displacement detection, considering three alternative approaches. **(a)** Analysis of the mean luma energy. **(b)** Analysis of the frame differencing.

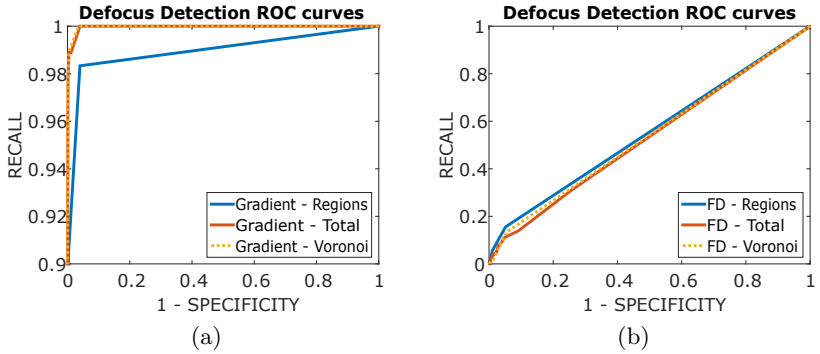

Fig. 4. ROC curves for defocus detection, considering three alternative approaches. **(a)** Analysis of the mean gradient energy. **(b)** Analysis of the frame differencing.

5 Conclusion

[**Giacomo**: Adriano: butta in inglese gli ongoing works (come ultima cosa)] .
Random Toughs:

- The problem of false alarms, radio module activation
- Other tampering attacks like obfuscation (??) which might be due to environmental phenomena such as rain, fog and mist over the camera lenses have to be detected by image analysis methods
- Displacement can be perceived by MEMS as well but these device alone are prone to false alarms. Visual inspection is necessary to reduce false alarms

Ongoing work regards the extension of our solution to other types of tampering, as occlusions or imaging sensor degradations. Furthermore, we are investigating strategies to improve the detection performance and reduce the number of *FP*s

	Displacement Detection			Defocus Detection		
	Regions	**Total**	**Voronoi**	**Regions**	**Total**	**Voronoi**
Luma	0.9994	0.9989	0.9985			
Gradient				0.9895	0.9996	0.9996
FD	0.9974	0.9944	0.9974	0.5526	0.5322	0.5391

Table 1. Area under curve of the analized solutions

by combination of our solution with other techniques: for example we could use sequential techniques on the features, or integrate the frame analysis with the data extracted from MEMS inertial sensors.

Acknowledgments

Authors would like to thank ST for supporting Adriano Gaibotti.

References

1. J. Darbon, A. Cunha, T.F. Chan, S. Osher, and G.J. Jensen. Fast nonlocal filtering applied to electron cryomicroscopy. In *Biomedical Imaging: From Nano to Macro, 2008. ISBI 2008. 5th IEEE International Symposium on*, pages 1331–1334, May 2008.
2. Jin Wang, Yanwen Guo, Yiting Ying, Yanli Liu, and Qunsheng Peng. Fast non-local algorithm for image denoising. In *Image Processing, 2006 IEEE International Conference on*, pages 1429–1432, October 2006.
3. Arun Hampapur, Lisa Brown, Jonathan Connell, Ahmet Ekin, Norman Haas, Max Lu, Hans Merkl, and Sharath Pankanti. Smart video surveillance: exploring the concept of multiscale spatiotemporal tracking. *Signal Processing Magazine, IEEE*, 22(2):38–51, 2005.
4. Ian F Akyildiz, Tommaso Melodia, and Kaushik R Chowdhury. A survey on wireless multimedia sensor networks. *Computer networks*, 51(4):921–960, 2007.
5. Adrian Perrig, John Stankovic, and David Wagner. Security in wireless sensor networks. *Communications of the ACM*, 47(6):53–57, 2004.
6. Massimo Piccardi. Background subtraction techniques: a review. In *Systems, man and cybernetics, 2004 IEEE international conference on*, volume 4, pages 3099–3104. IEEE, 2004.
7. Anil Aksay, Alptekin Temizel, and A. Enis Cetin. Camera tamper detection using wavelet analysis for video surveillance. In *Advanced Video and Signal Based Surveillance, 2007. AVSS 2007. IEEE Conference on*, pages 558–562. IEEE, 2007.
8. Ali Saglam and Alptekin Temizel. Real-time adaptive camera tamper detection for video surveillance. In *Advanced Video and Signal Based Surveillance, 2009. AVSS'09. Sixth IEEE International Conference on*, pages 430–435. IEEE, 2009.
9. Pedro Gil-Jiménez, R. López-Sastre, Philip Siegmann, Javier Acevedo-Rodríguez, and Saturnino Maldonado-Bascón. Automatic control of video surveillance camera sabotage. *Nature Inspired Problem-Solving Methods in Knowledge Engineering*, pages 222–231, 2007.

10. Cesare Alippi, Giacomo Boracchi, Romolo Camplani, and Manuel Roveri. Detecting external disturbances on the camera lens in wireless multimedia sensor networks. *Instrumentation and Measurement, IEEE Transactions on*, 59(11):2982–2990, 2010.
11. Evan Ribnick, Stefan Atev, Osama Masoud, Nikolaos Papanikolopoulos, and Richard Voyles. Real-time detection of camera tampering. In *Video and Signal Based Surveillance, 2006. AVSS'06. IEEE International Conference on*, pages 10–10. IEEE, 2006.
12. T Kryjak, M Komorkiewicz, and M Gorgon. Fpga implementation of camera tamper detection in real-time. In *Design and Architectures for Signal and Image Processing (DASIP), 2012 Conference on*, pages 1–8. IEEE, 2012.
13. Sebastien Harasse, Laurent Bonnaud, Alice Caplier, and Michel Desvignes. Automated camera dysfunctions detection. In *Image Analysis and Interpretation, 2004. 6th IEEE Southwest Symposium on*, pages 36–40. IEEE, 2004.
14. Theodore Tsesmelis, Lars Christensen, Preben Fihl, and Thomas B Moeslund. Tamper detection for active surveillance systems. In *Advanced Video and Signal Based Surveillance (AVSS), 2013 10th IEEE International Conference on*, pages 57–62. IEEE, 2013.
15. Franz Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.