Tampering Detection In Low-Power Smart Cameras

Adriano Gaibotti[1,2], 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 in 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., occlusions or device displacements), these have to be promptly detected to possibly activate countermeasures. Tampering detection is particularly challenging in battery-powered cameras, where it is not possible to acquire images at full-speed frame-rates, nor use sophisticated image-analysis algorithms.
We here introduce a tampering-detection algorithm 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 an outlier is detected as tampered. 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 improve the detection of camera displacements. Experiments show that the proposed algorithm can successfully operate on sequences acquired at very low-frame rate, such as one frame every minute, with a very small computational complexity.

Keywords: tampering detection, smart cameras, displacement detection, blurring detection, low-power cameras, low-frame rate.

1 Introduction

Cameras operating outdoor and in harsh environments are exposed to atmospheric phenomena and intentional attacks that might prevent the correct image acquisition. Rain, snow, dust lying on the camera lens cause blurry (Figure 1(a)) or partially occluded (Figure 1(b)) pictures, while wind might displace the camera (Figure 1(c) - 1(d)). Similarly, an attacker can intentionally change the camera focus, spray some opaque or glossy liquid, displace or occlude the camera. We refer to these events/attacks as tampering. In many situations, tampering is

(a) (b) (c) (d)

Fig. 1. Examples of tampering events due to atmospheric phenomena. **(a)** rain drops resulting in a blurry picture, **(b)** snow partially occluding the camera view, **(c)-(d)** wind displacing the camera.

not straightforward to detect (e.g. when the camera is not physically damaged and does not go out-of-order), and image analysis techniques are the only viable option.

Tampering detection is an essential feature in surveillance systems [1], which are expected to autonomously detect any tampering, and promptly report alerts. Tampering, in fact, might result in images that are useless for monitoring purposes: even a mild blur might hinder the identification of important details such as licence plates. Surveillance cameras most often operate at normal frame-rates (e.g. around few frames per second) and are connected to the power supply. Tampering detection for surveillance cameras have been quite investigated in the literature; in particular, camera displacements and occlusions are typically detected by comparing the current frame against an estimate of the scene background, while blurring is detected by monitoring the high-frequency components of images, as these are expected to drop. In [2] and [3] tampering is detected by comparing the histograms of background and current frame, and by monitoring the energy in wavelet or Fourier domain. Two background models, estimated over different time intervals, are used in [3]. In [4], tampering detection is performed by combining background subtraction together with edge detection and normalized cross-correlation. Background estimates and block matching in [5] enable the displacement detection, while the number of SURF [6] keypoints is used to detect blurring. A buffer of recent frames rather than an explicit background can be used as in [7]. All the above solutions, including [8, 9], are computationally demanding and are not viable options for low-power cameras.

In this work, we expressly target low-power and ultra-low-power smart cameras, like *SecSoC* (Security System on Chip), an innovative prototype based on a cluster of ReISC (Reduced Energy Instruction Set Computer) cores, designed and produced by STMicroelectronics. SecSoC is a battery-powered device, characterized by a constrained computational power (clock rates 82.5 MHz at 1.2V, and sub-1MHz at 0.6V) and reduced memory (1.25 MB). While low-power cameras are not meant for critical surveillance applications, they might be easily employed for monitoring wide environments thanks to their low cost and maintenance requirements.

Tampering detection in low-power cameras (eventually organized in wireless sensor network) is more challenging than in conventional surveillance systems, and this problem has not been much investigated so far [10, 11]. 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. These aspects prevent the use of learned background models and the analysis of foreground variations. In fact, when dynamic environments are monitored at low frame-rates, changes in the scene and in the light conditions might produce consecutive frames that are very different (see Figure 2). Smart cameras have to promptly distinguish between *normal* changes (due to illumination or movements in the scene), and changes due to camera tampering, to raise an alert and eventually avoid the transmission of tampered frames.

We address the detection of camera blurring and displacement (Section 2), and propose an algorithm (Section 3) that relies on indicators that can be easily computed in low-power smart cameras. We monitor the average image intensity or frame difference (to detect camera displacement), and the average gradient norm (to detect blurring), and detect tampered frames by analyzing outliers in these indicators. In particular, we show that separately monitoring these indicators over different regions of the image can substantially improve the displacement-detection performance. Image regions are defined during an initial configuration phase (Section 3.1), thus the algorithm operates at a negligible computational overhead with respect to monitoring the whole image (Section 3.2). Our algorithm thus represents a prompt trigger, to be possibly combined with other sequential monitoring techniques. Experiments (Section 4) show that leveraging image regions can substantially improve the displacement-detection performance.

2 Problem Formulation

Let z_t be frame acquired at time t

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

where $\mathcal{D}_t$ denotes an operator transforming the original image y_t, and $x \in \mathbb{Z}^2$ indicates the pixel coordinates belonging to the regular pixel grid $X \subset \mathbb{Z}^2$. As far as there are 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 image noise and y_t are acquired from the same viewpoint and camera orientation, even though typically $y_t \neq y_{t-1}$ because the depicted scene changes.
When, at time τ^*, an external disturbance introduces *blurring*, the image y_t is degraded by an unknown blur operator, and z_t becomes

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

Fig. 2. Examples of synthetically generated blurring (3-rd frame in **(a)**) and camera displacement (3-rd frame in **(b)**).

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 and y_t refer to different viewpoints and/or camera orientations.

The proposed tampering-detection algorithm analyzes a sequence of frames $\{z_t, t = 1, \dots\}$ to detect the time instant τ^* when tampering like (3) or (4) occurs. We assume that T_0 tampering-free frames are provided for training. For simplicity, we consider grayscale frames: extensions to color images are straightforward.

3 Tampering Detection

Algorithm 1 presents the proposed tampering detection, which relies on simple indicators such as the average intensity (the *luma*, denoted by l), the average frame difference (*FD*, denoted by d) and the average norm of the gradient (denoted by g). The first two are meant to detect camera displacements, which would substantially change the image content, while the latter the blurring, which would attenuate the high frequencies of the image. As anticipated in Section 1, during the initial configuration, the scene is segmented in K disjoint regions $\{R_k, k = 1, \dots, K\}$, namely $R_k \subset \mathcal{X}, R_i \cap R_j = \emptyset, \forall i \neq j$. The employed segmentation is detailed in Section 3.1.

For each z_t, we compute the luma and frame difference separately on each of the K regions,

$$l_k(t) = \frac{1}{\#R_k} \sum_{x \in R_k} z_t(x) \tag{5}$$

$$d_k(t) = \frac{1}{\#R_k} \sum_{x \in R_k} \left(z_t(x) - z_{t-1}(x)\right)^2, \quad k = 1, \dots, K, \tag{6}$$

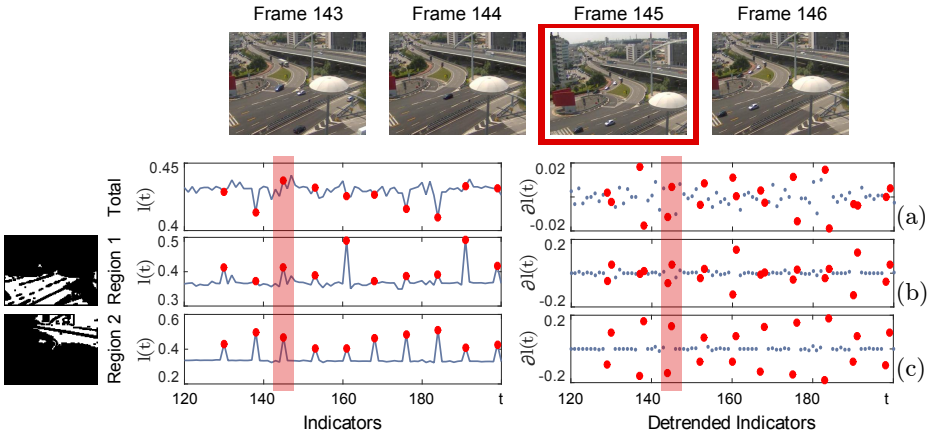

Fig. 3. Example of camera displacements. Left plots report the luma values, while right plots depict the detrended luma: we show both indicators computed over the full image (**a**) and on two reported regions (**b** and **c**). Red dots indicate a single displaced frame. Values in the highlighted areas refer to the frames depicted on top of the figure. The highlighted displacement yields a small peak in $l(t)$ and two outliers in the sequence of detrended indicators $\partial l(t)$, which can be clearly detected.

where $\#(\cdot)$ denotes the cardinality of a set. We can simultaneously monitor both luma and frame difference, even though computing d_k requires to store the previous frame and this also depends on memory availability of the device. In what follows, including Algorithm 1, we consider only the luma l, but the same procedures apply to the frame difference d as well. The average gradient norm is instead computed over the whole image

$$g(t) = \sum_{x \in X} \left(\sqrt{(z_t \circledast f_h)^2 (x) + (z_t \circledast f_v)^2 (x)} \right), \tag{7}$$

where f_h and f_v are the horizontal and vertical derivative filters, respectively, and $\circledast$ denotes the 2d convolution.

Figure 3 shows how a camera displacement affects the luma. First of all, we observe that a displaced frame changes $l(t)$ (introducing a peak in Figure 3(a)) and that this change is also visible in the indicators $l_k(t)$ for the regions (Figure 3(b) and Figure 3(c)). Second, we observe that outlier-detection methods [12] based on density estimates or on confidence intervals –that are very efficient to run– cannot be straightforwardly applied here. In fact, these methods are meant for independent and identically distributed (i.i.d.) random variables, while here indicators follow an unpredictable trend because of changes in the scene or in the illumination. Therefore, we perform a detrending [13] of the indicator sequence by a temporal derivative

$$\partial l_k(t) = l_k(t) - l_k(t-1), \quad k = 1, \ldots, K, \tag{8}$$

and we similarly define $\partial d_k(t)$ and $\partial g(t)$.

Algorithm 1: The Proposed Tampering-Detection Algorithm

Input: $\gamma_l, \gamma_g, \Gamma_l$, training frames $\{z_t, t = 1, \ldots, T_0\}$, regions $\{R_k,\ k = 1, \ldots, K\}$
Training phase:
1. Compute $\partial l_k(t)$ and $\partial g(t), t = 1, \cdots, T_o$
2. Compute $\overline{\partial l}_k$, $\overline{\partial g}$, σ_{l_k} and σ_g.

Operational phase:
3. **for** $t = T_o + 1, \ldots, \infty$ **do**
4. Get frame z_t, set $n_l = 0$;
5. Compute $\partial g(t)$
6. **if** $\partial g(t) < -\gamma_g \sigma_g \vee \partial l(t) > \gamma_g \sigma_g$ **then**
7. raise a blurring alert in z_t
   end
8. **for** $k = 1, \ldots, K$ **do**
9. Compute $\partial l_k(t)$
10. **if** $\partial l_k(t) < -\gamma_l \sigma_{l_k} \vee \partial l_k(t) > \gamma_l \sigma_{l_k}$ **then**
11. $n_l = n_l + 1$
    end
    end
12. **if** $n_l \geq \Gamma_l$ **then**
13. raise a camera displacement alert in z_t
    end

end

The sequences of detrended indicators (reported in Figure 3) can be suitably monitored by the following confidence intervals:

$$[\overline{\partial l}_k(t) - \gamma_l \sigma_{l_k}, \overline{\partial l}_k(t) + \gamma_l \sigma_{l_k}], \quad k = 1, \ldots, K\,, \tag{9}$$

where $\overline{\partial l}_k(t)$ denotes the mean and σ_{l_k} the standard deviation of ∂l over R_k (Algorithm 1, line 1), computed from tampering-free frames provided for training (i.e. $z_t\,, t = 1, \ldots, T_o$) and $\gamma_l > 0$ is a tuning parameter. Similar intervals are built for $\partial d_k(t)$ and ∂g (line 2).

During operations, indicators are computed (lines 5 and 9), and any indicator falling outside its confidence region is considered an outlier. In particular, any outlier in ∂g yields a blurring alert (line 6), while camera-displacement alerts are raised when at least Γ_l indicators ∂l_k simultaneously yield outliers (line 12). The threshold Γ_l together with γ_l determine the displacement-detection promptness. The extreme configurations correspond to $\Gamma_l = 1$, where it is sufficient that a single region fires an outlier to raise a camera-displacement alert, and $\Gamma_l = K - 1$ where all but one region[3] have to simultaneously fire an outlier to raise an alert.

It is important to remark that tampering yields outliers in the transient of the detrended indicators (8): namely, a single tampered frame yields two outliers (as shown in Figure 3) and only the first and the last of a sequence of consecutive

[3] It is better to exclude a region since, for instance, the sky-region typically does not change when the camera is displaced, either horizontally or vertically

tampered frames yield outliers in the detrended indicators. This is the reason why detrended indicators are monitored in a *one-shot* manner, targeting the detection of the first tampered-frame. On the one hand, this one-shot monitoring can provide prompt detections, which is particularly important at the low frame-rates we consider, on the other hand, the persistence of tampering is disregarded. To take this valuable information into account, some form of sequential monitoring should be applied to the indicator sequence as in [11], possibly combined with Algorithm 1.

3.1 Scene Segmentation

Scene has to be preliminarily segmented to define regions that Algorithm 1 takes as input. To this purpose, we use part of the training frames before T_0 to compute the feature vector $\mathbf{f}(x) \in \mathbb{R}^5$ for each pixel $x \in \mathcal{X}$

$$\mathbf{f}(x) = \left[r(x); c(x); \bar{l}(x); \sigma_l(x); \overline{g}(x); \sigma_g(x)\right] \ , \ \forall x \in X \, . \tag{10}$$

In (10), $r(x)$ and $c(x)$ denotes the row and column of x, respectively, $\bar{l}(x)$ and $\sigma_l(x)$ the mean and standard deviation of the intensity at x, computed over time, and $\overline{g}(x)$ and $\sigma_g(x)$ the mean of gradient norm and its standard deviation, computed over time. These feature vectors are meant to cluster pixels in regions having, over training frames, similar spatial appearance and temporal behavior.

As in superpixel methods [14], segmentation is performed by k-means clustering. Feature vectors (10) over the whole image are clustered by a weighted k-means [15] that scales each component of the Euclidean distance between a feature vector and a cluster centroid of a weight that is inversely proportional to the standard deviation over the cluster. This scaling compensates the fact that the components of the feature vector might span very different ranges. The number of clusters is defined by testing several values and then choosing the best solution according to the Calinski-Harabasz criterion [16].

Finally, morphological image-processing operations are executed to remove boundaries between different regions, and eventually regions that are too small. This defines the regions $\{R_k \, , \ k = 1, \ldots, K\}$, and two examples are reported in Figure 3.

3.2 Computational Complexity

The most computationally demanding operations of Algorithm 1 consist in computing the indicators, in particular g. Computing g requires, when using the Sobel filters in (7), 34 operations per pixels[4], while computing l and d requires 1 and 3 operations per pixel, respectively. Detecting outliers in the indicators requires two comparisons per frame: thus, monitoring regions instead of the whole image has a negligible impact on the overall computational complexity. Algorithm 1 has very low memory requirements: l and g indicators can be computed

[4] Execution can be accelerated whether hardware implementations of the FFT transform are available

online, while d requires to store a single frame in memory. Segmentation is executed only during the initial configuration and can be performed on an external device connected to the smart camera. As such, Algorithm 1 can be properly executed on low-power and ultra low-power cameras.

4 Experiments

Experiments are meant to assess the advantages of separately monitoring regions in the considered tampering-detection framework. To this purpose, we show that Algorithm 1 detects camera displacements better than monitoring indicators on the whole image (*Whole Image* in Figure 4). We also show that it is important to adapt regions to the image content, since operating on K regions obtained by clustering $[r(x), c(x)]$ leads to performance loss (*Voronoi* in Figure 4). For both Voronoi and regions defined as in Section 3.1, we consider the two extreme configurations where alerts are raised at the first region firing an outlier or when $K-1$ regions simultaneously fire outliers.

We recorded 8 sequences from webcams monitoring different urban areas, yielding overall 12200 frames. From each frame, we have cropped the central area, removing the 50 top-most and bottom-most rows, and the 50 left-most and right-most columns[5]. We have synthetically introduced 10% of tampered frames: displacements have been simulated by moving the central cropping area of a random shift having magnitude between 20 and 50 rows and columns (as in Figure 2(b)), while blurring (as in Figure 2(a)) was simulated by convolution against a Gaussian kernel, with standard deviation randomly defined in the range $[1, 5]$.

A typical figure of merit for assessing detection performance is the Receiver Operating Characteristic (ROC) curve, where each point corresponds to a pair (FPR_γ, TPR_γ) defined as

$$\mathrm{FPR}_\gamma = \frac{\mathrm{FP}_\gamma}{\mathrm{TN}_\gamma + \mathrm{FP}_\gamma}, \quad \text{and} \quad \mathrm{TPR}_\gamma = \frac{\mathrm{TP}_\gamma}{\mathrm{TP}_\gamma + \mathrm{FN}_\gamma}.$$

Here, TP_γ represents the number of true positives (tampered frames correctly detected), FP_γ of false positives (tampering-free frames detected), TN_γ of true negatives (tampering-free frames not detected) and FN_γ of false negatives (missed tampered frames), for a given value of γ. ROC curves have been computed by varying γ_l, γ_d and γ_g in a range of values between 0.1 and 50.

Discussion ROC curves in Figure 4(a)-(b) and the Area Under the Curve (AUC) values reported in the caption confirm that camera displacements can be better detected by monitoring FD than luma, and that it is convenient to define regions by segmenting the scene. As an example, if we set $\gamma_d = 15.6$, Algorithm 1 operates at $\mathrm{FPR} = 1\%$ and detects nearly all tampering since $\mathrm{TPR} = 99.92\%$. In contrast, to operate at $\mathrm{FPR} = 1\%$ when monitoring the whole image or Voronoi regions we obtain $\mathrm{TPR} = 91.67\%$ ($\gamma_d = 6.5$) and $\mathrm{TPR} = 89.05\%$ ($\gamma_d = 18.64$),

[5] Sequences can be provided upon request.

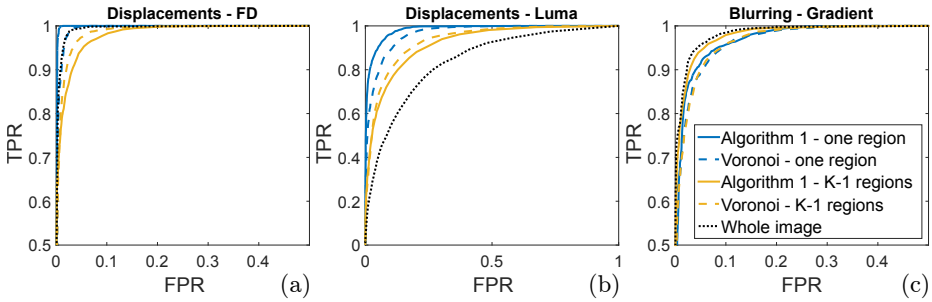

Fig. 4. **(a)** ROC curves relative to displacement detection based on FD: the AUC obtained by "Algorithm 1 - One region" is 99.89%, while the AUC obtained by on the whole image is 99.65%. **(b)** ROC curves relative to displacement detection based on luma: the AUC obtained by "Algorithm 1 - One region" is 98.44%, while the AUC obtained on the whole image is 84.07%. **(c)** ROC curves relative to blurring detection based on gradient: as opposed to **(a)** and **(b)**, "Algorithm 1 - One region" (AUC 98.4%) is outperformed by monitoring the whole image, AUC = 99.13%. To highlight differences, the ROC curves in **(a)** and **(c)** are drawn over a smaller FPR, TPR range.

respectively. When monitoring l, Algorithm 1 achieves TPR = 73.98% at FPR = 1% ($\gamma_d = 5.8$), while monitoring the whole image is quite ineffective (TPR = 17.85%, $\gamma_d = 3.7$), and similarly perform Voronoi's regions (TPR = 53.02%, $\gamma_d = 6$). We remark that, in the considered low-power scenario, it is important to operate at low FPR to prevent useless data transmission (that would reduce the battery time-life) and the activation of eventual countermeasures. As far as the values of Γ_d and Γ_l are concerned, the most effective configuration consists in raising an alert as soon as a single region fires a very clear outlier, which implies setting quite wide intervals (9). This option is also preferable because it enables the detection of partial occlusions of the scene. We observe that displacements typically change all the Voronoi regions, and this is why Voronoi outperforms Algorithm 1 in the configuration $K - 1$ in both luma and FD.

Figure 4(c) confirms that blurring is more effectively detected by monitoring the whole image at once. This is probably due to the fact that, typically, regions do not include the most prominent edge of the scene, which are indeed the most informative parts to detect blurring.

5 Conclusions

We presented a tampering-detection algorithm that leverages an image segmentation to improve the detection of camera displacements, and that at the same time can detect blurring. The algorithm has low-computational complexity and has to be considered a prompt trigger for detecting tampering in low-power and ultra low-power cameras operating at low frame rates.

Ongoing work concerns approaching other types of tampering, such as degradations of the imaging sensor, and the integration of sequential monitoring schemes [11] to detect subtle tampering that persists over time. Moreover, we

will investigate the use of superpixels methods [14] to segment the scene, by including suitable temporal information to the feature vectors.

References

1. 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.
2. Anil Aksay, Alptekin Temizel, and A. Enis Cetin. Camera tamper detection using wavelet analysis for video surveillance. In *IEEE Int. Conf. on Advanced Video and Signal Based Surveillance, AVVS 2007*, pages 558–562.
3. 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.
4. 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.
5. Theodore Tsesmelis, Lars Christensen, Preben Fihl, and Thomas B Moeslund. Tamper detection for active surveillance systems. In *IEEE Int. Conf. on Advanced Video and Signal Based Surveillance, AVVS 2013*, pages 57–62, 2013.
6. Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European Conference on Computer Vision, ECCV*, pages 404–417. Springer, 2006.
7. Evan Ribnick, Stefan Atev, Osama Masoud, Nikolaos Papanikolopoulos, and Richard Voyles. Real-time detection of camera tampering. In *IEEE Int. Conf. on Video and Signal Based Surveillance, AVSS 2006.*, pages 10–10, 2006.
8. Sebastien Harasse, Laurent Bonnaud, Alice Caplier, and Michel Desvignes. Automated camera dysfunctions detection. In *IEEE Southwest Symp. on Image Analysis and Interpretation*, pages 36–40, 2004.
9. T Kryjak, M Komorkiewicz, and M Gorgon. FPGA implementation of camera tamper detection in real-time. In *Int. Conf. on Design and Architectures for Signal and Image Processing DASIP*, pages 1–8, 2012.
10. Adrian Perrig, John Stankovic, and David Wagner. Security in wireless sensor networks. *Communications of the ACM*, 47(6):53–57, 2004.
11. Cesare Alippi, Giacomo Boracchi, Romolo Camplani, and Manuel Roveri. Detecting external disturbances on the camera lens in wireless multimedia sensor networks. *IEEE Trans. on Instr. and Meas.*, 59(11):2982–2990, 2010.
12. Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.
13. Frederik Gustafsson. *Adaptive Filtering and Change Detection*. Wiley, Oct 2000.
14. Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Susstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(11), November.
15. Dane P Kottke and Ying Sun. Motion estimation via cluster matching. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(11):1128–1132, 1994.
16. Tadeusz Caliński and Jerzy Harabasz. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27, 1974.