Efficient Camera Tampering Detection with Automatic Parameter Calibration

Alexey Sidnev[1,2], Marina Barinova[1,2], Sergei Nosov[1]
[1]Internet of Things Group, Intel, Nizhny Novgorod, Russia
[2]Lobachevsky State University of Nizhny Novgorod, Russia
{alexey.sidnev, marina.barinova, sergei.nosov}@intel.com

Abstract

Camera anomaly detection, an important part of modern digital security and surveillance cameras, captures malicious actions against the camera like spray paint, occlusion, and displacement. Multiple challenges exist for solving camera anomaly detection in a practical way, such as limited computational budget and difficult tampering events. In this article, we present a fast and accurate approach for camera anomaly detection, which consists of the application of low-level computer vision techniques combined with a unique automatic parameter calibration method that eliminates the need for manual parameter tuning. Based on the results of our experiments, the proposed algorithm detects more than 96 percent of malicious events without false alarms for 13 hours of video with 350 tampering events from more than 20 different scenes. On the TCAD dataset [15], we achieved state-of-the-art results.

1. Introduction

Today it is difficult to imagine public places without security cameras. But, despite the proliferation and obvious value of security cameras in public places, most of them are not secure devices and do not recognize tampering impacts on them. The Digital Security and Surveillance (DSS) camera is intended to verify that it is working correctly. For example, the image from the camera should have good quality, the camera should cover the area of interest, should not be blocked by a malicious object, and should work as designed. For those reasons, we must be able to recognize and to signal malicious impacts on the camera.

The goal of the camera anomaly detection algorithm is to trigger an alarm when a tampering event occurs. We define three general types of tampering events (Fig. 1):

- Occlusion – object obstructs a large part of camera view;
- Defocus – camera is out of focus;
- Displacement – camera is redirected.

978-1-5386-9294-3/18/$31.00 ©2018 IEEE

2. Previous work

Extensive research has been conducted on the camera tampering detection problem. For example, 23 research-based articles which were published on the subject in the last 14 years are shown in Fig. 2. Some of the articles and research methods are more current, some have become classic and are often referred to. Due to performance considerations, however, almost all of these research approaches are based on lightweight computer vision kernels. Only [20] use a neural-network based approach with four convolution layers and one fully connected layer for detection of occlusion and defocus events.

Feature thresholding is the most common approach for tampering detection. The threshold value can be absolute or relative to the normal value of a feature that can be calculated from a previous frame or background image (for example, in [23], a Gaussian mixture model is used for modeling normal operation view). Some approaches use adaptive threshold calculation [1, 7, 19] or complex rules for tampering detection, like Markov chain [15] and mixture of features [16, 21].

The main approach for defocus detection is based on the assumption that a small number of features in an image indicates a blur. Sobel, Canny, and Wavelet-based edge detectors are quite popular for estimating different levels of defocus ([1, 3, 4, 6-8, 11, 15, 19, 22, 23]). Some papers use more complex descriptors like SURF, HOG [13], and SIFT [14] with a similar idea. These features are more computationally complex but are invariant to some transformations and are partially invariant to illumination and light changes. The sum of high-frequency specter components with Cosine or Fourier transforms has a very strong correlation with the number of details on the image. This approach is the most time-consuming and is used in [5, 16, 21].

Block-matching methods, like NCC [1], ZNCC [4], TSS [8], RANSAC [13, 18], are quite natural for displacement detection. In most cases, however, you do not need precise

No tampering Occlusion Defocus Displacement

Figure 1: Camera tampering events.

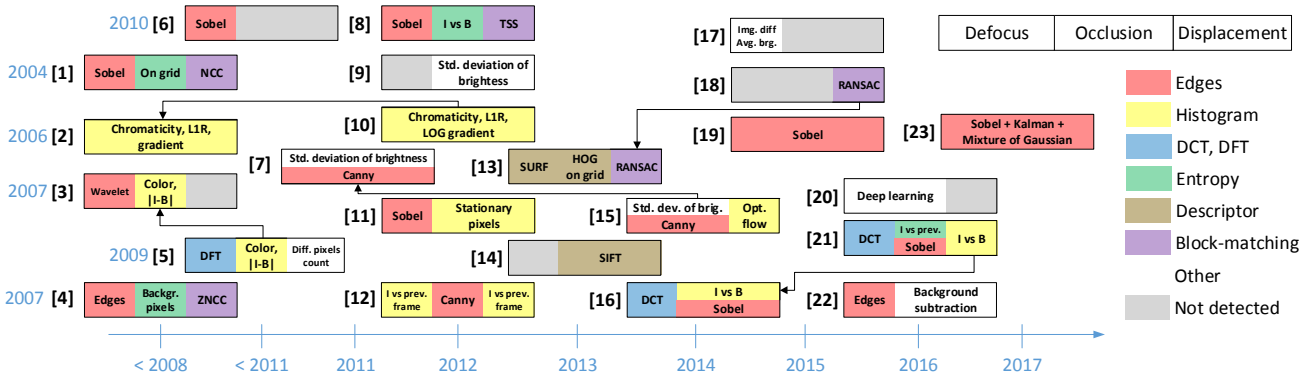

Figure 2: Features for camera anomaly detection in papers. I – current frame, B – background image.

motion estimation, so some papers use fast and less accurate approaches. For example, the count of non-equal pixels between background image and delayed background image is used in [5], standard deviation of brightness is used in [7, 9], and edge count is used in [7, 16, 19].

A histogram-based analysis is widely used for detecting different tampering events. Detectors in [2, 10] are built only on histogram analysis and use short-term and long-term frame pools. Each new frame is pushed into the short-term pool; if the short-term pool is full, the oldest frame from it is pushed to the long-term pool. Image dissimilarity based on histogram absolute difference is computed between every frame in the short-term pool and every frame in the long-term pool. In their turn, the frames from the long-term pool are compared among themselves. The ratio of two medians is a primary feature for tampering detection. Histograms are computed based on image brightness, color components transformation, and gradient direction.

In [3, 5], two types of the histogram for occlusion detection are used. The absolute difference histogram of the frame and the background image is calculated, and then the sum of the first K bins is compared to the sum of all bins. The second histogram is calculated for the brightness level of the current frame and background image. The maximum histogram value is used as a feature. The same idea, but with a sum of maximum value neighbors in the histogram, is used in [16, 21] for occlusion and displacement detection.

Short-term and long-term background images are used in [11]. The short-term image is updated with the current frame, and the long-term image is updated with the short-term model. Motion mask is used for stationary pixels detection. Only stationary pixels are accumulated in the histograms for long-term and short-term background images. The first and the last bins are excluded. The correlation ratio between those histograms is used for occlusion and displacement detection. In [12], histograms for current and previous frames are calculated, and the sum of histogram bins with low and high pixel intensity (from 1 to 96 and from 160 to 255) is used for defocus and displacement detection.

Authors of [6] use CUSUM algorithm for anomaly detection in sequence of blur estimations based on Sobel edges. In [9], standard deviation of brightness is used for occlusion and displacement detection. In [17], derivations of average brightness and gradient are used for defocus detection.

Background subtraction method is used for occlusion detection in [22]. The background model is built, and foreground objects are interpreted as occlusion.

Low-textured or textureless backgrounds and darkness or low quality video are the most challenging scenarios for defocus detection. In most approaches, these scenarios lead to false alarms. We use standard deviation to detect anomaly for such cases and Canny edges for normal cases.

Difficult cases also exist for occlusion detection. Textured occlusion can be missed by some approaches, but background subtraction methods can handle it well. If the grayscale brightness of object is very similar to the background, the object cannot be distinguished from the background. For that reason, we use a multichannel background model. Very slow occlusion becomes a challenging task in this case, but our approach detects it as defocus for textureless occlusion or displacement for textured occlusion. Large objects or crowd in front of the camera view are typical difficult cases. To solve this problem, we use separate thresholds for moving foreground objects and static objects.

Sluggish displacement is a very difficult anomaly to detect because background models cannot handle it well. We propose the following approach to solve that problem.

3. Proposed approach

Input frame is downscaled to improve performance without significant changes in accuracy since all tampering events are high-level and do not require fine-grained details. Color image (RGB) is used for occlusion size estimation. Grayscale image is used for defocus-level and displacement estimation.

Detectors work independently. If the amount of consequent tampering events exceeds a predetermined threshold (for example, 20 frames), an alarm is triggered. The values of all parameters presented in our approach are

based on the results of our experiments.

3.1. Defocus detection

Canny edge detector is quite popular for estimating different levels of defocus. A small number of edges indicates a blur, but if an absolute number of edges is used for every scene, the threshold value must be adjusted. Our proposed method finds a ratio between a number of Canny edges on the current frame and a median of values for the last K frames. Moreover, we add histogram equalization prior to Canny edges evaluation. This approach is less sensitive to lighting change, is more robust, has an easily interpretable relative threshold ratio, and can be used to detect different levels of defocus.

As illustrated in [15], we compute standard deviation on the whole frame and on the grid 5x5 to detect very strong blur or occlusion placed right in front of the camera lens. In this case, an image has almost the same value in most of the pixels.

3.2. Camera motion detection based on the optical flow histogram analysis

We use a histogram analysis of sparse optical flow computed on Harris corners to estimate camera motion. Optical flow histogram is computed in the following way: a bin number corresponds to displacement direction with the value of the sum of all displacement magnitudes in a corresponding direction.

We propose three key ideas of histogram analysis. There is no motion if a stationary point count is greater than the threshold. Stationary point is a point with almost zero magnitude (we use threshold 0.5). A radial standard deviation is used to estimate that all key points move in one direction (camera motion). Motion direction is computed through interpolation of the max value bin and its neighbors.

The standard deviation of histogram values cannot be computed naively because the first and the last bins of the histogram must have small distance between them. We compute the standard deviation of projections onto axes and use normalized L2 distance to compute a final value. If the computed value is close to 1 (more than 0.95), we interpret that as movement in one direction; if the value is less than 0.95 and the amount of stationary points is greater than 30 percent, then we interpret that as no motion. If standard deviation is not approached with this in mind, motion estimation cannot be accurately computed.

3.3. Sluggish motion detection

A camera motion estimation based on two consequent frames has two primary disadvantages: it cannot detect sluggish motion, and it can detect false motions due to the movement of large objects, especially if the objects are close to the camera lens. To solve this issue, we propose a pipeline with motion estimation for multiple frames from history pool.

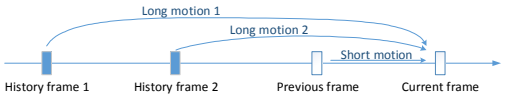

Figure 3: Motion estimation.

History pool contains the first frame in the beginning. After that, for each new frame, we estimate motion between current and previous frames (short motion), current frame and frame from history pool (long motion). With slow camera motion, the short motion is close to zero, but after several frames, even a very small displacement becomes apparent by the long-motion metric, so we use long motion for primary estimation (Fig. 3).

Movement of large objects is the main reason for false camera motion events. However, after the object leaves the camera view, the camera observes the original scene, and long motion estimation returns zero value.

If there is no motion estimation for a frame from the history pool, we add the last estimated frame to the history pool (Fig. 3), which provides motion estimation from history frame 1 to history frame 2 and from history frame 2 to current frame. For the next frame from the sequence, we will estimate long motion based on the history frame 1. If the long motion is estimated, then we do not need history frame 2 anymore, thereby removing it from the history pool. If the long motion is not estimated, we perform the second long motion estimation based on the history frame 2. If we cannot estimate the long motion based on frames from history for a long time (we use 50 frames), we erase frames history, add the current frame to the history pool, and use motion estimation based on two consequent frames as primary.

3.4. Occlusion detection

Background subtraction methods are very useful for occlusion detection. We introduce a modification of the background model from [11], which works on every channel of RGB image and can detect static and moving objects separately. Occlusion size is estimated as the largest foreground object.

Two background models are built for each RGB image channel. The first is a short-term model, and the second is a long-term model. The short-term background model is built and updated every frame, and the long-term background model is built and updated every fifth short-term background image, which ensures that the last model is more stable and does not react to moving objects.

We use motion mask for static foreground object detection. For every pixel, motion value is increased by 15 if the pixel belongs to the foreground and the difference with the pixel intensity from the previous frame is more than 5; otherwise, motion value is decreased by 4. If the

motion value is more than 15, the pixel belongs to the moving object.

The short-term model is used to get the stationary mask (pixels that are not moving for all channels). The long-term model is used for finding foreground mask (pixels that are foreground at least for one channel). After that, the largest stationary and non-stationary foreground objects are found. The proposed approach is sensitive to colors of occlusion objects. Different thresholds for static and moving objects allow you to customize the approach to a wider range of use cases.

4. Automatic calibration

One of the most significant disadvantages of a typical computer vision solution is the presence of configurable parameters that can significantly affect quality in each specific use case (for example, an outdoor or indoor scene). For this reason, the parameters in camera tampering algorithm are usually difficult to use in practice because they require careful manual tuning for each camera installation. Our proposed solution allows for automatic parameter configuration and can be used in different use cases without manual parameter calibration.

4.1. Defocus and occlusion thresholds

Tampering occlusion event is triggered only if an area of tampering object is greater than the threshold for the specified amount of time; in other words, every frame during that time contains tampering object with an area greater than the threshold, which can be used to select the optimal threshold automatically. For example, suppose we have a video without events of malicious effects on the camera, and occlusion estimation was computed for every processed frame (blue line in Fig. 4). We must find occlusion threshold that will not produce tampering event for provided detection time (continuous sequence of frames). Low occlusion threshold leads to false alarm (lower dotted line in Fig. 4); high occlusion threshold leads to an overestimation (upper dotted line in Fig. 4). As a result, we can miss some of the tampering events. Therefore, we should find the optimal value of the threshold (solid line in the middle in Fig. 4). We use binary search to find the optimal occlusion threshold. It can be done for multiple videos in a dataset without tampering events, and the maximum overall threshold is the optimal threshold. When the optimal value of the threshold is found, it is sufficient to increase slightly the threshold or the detection time to suppress false alarms. We prefer to use the latter in our experiments. On an actual installation, when there are no previously recorded data available, the calibration might happen during the first several minutes or hours after the installation.

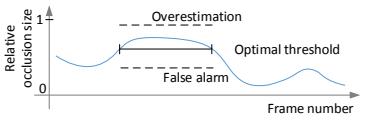

Figure 4: Automatic threshold estimation.

Defocus threshold can be found in a similar way, except that the low threshold leads to overestimation, and the high threshold leads to a false alarm, and then we must find a minimum overall threshold and slightly decrease it or increase detection time to suppress false alarms.

4.2. Ignore angles for displacement

There are some difficult use cases for displacement detection (especially for traffic cameras), such as the movement of a large number of big objects in the same direction. We introduce automatic estimation of angles that can produce false alarms, and thus, should be ignored during displacement detection.

Ignore angles for displacement are estimated with the following approach:

1. Provide displacement threshold. Quantize angles from $-\pi$ to π. Ignored angles list is empty.
2. Estimate maximum displacement, exclude directions in the ignored angle list. If maximum displacement is less than the threshold value, then ignored angle list is returned.
3. If maximum displacement is more than threshold value, then compute histogram of displacement vectors and add direction with the highest magnitude to the list of ignored angles. Go to step 2.

Fig. 5 provides an example of ignored angles for Traffic: Single lane dataset.

5. Results

5.1. Datasets

We have collected a large dataset (more than 13 hours of video) with different tampering events and use cases: the traffic cameras with single lane (TCAD dataset from [15]) and multiple lanes, and indoor and outdoor DSS cameras (see table 1). Every video in each dataset contains only one tampering event at the end.

We have identified several subcategories of tampering events as well. Defocus and occlusion can be partial (e.g. when only part of the camera lens is closed) or full. Depending on the blur effect, defocus can be low or high. Occlusion can be caused by textured of textureless (solid) object. Displacement can be slow (several pixels per second), medium, and fast (with motion blur). Textureless

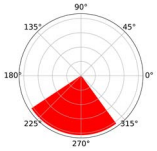


Figure 5: Ignore angles for displacement for the Traffic: Single lane dataset and the image example from the dataset.

Dataset	Videos count	Videos length, hours	Tampering events				Tampering events details										
			Def.	Displ.	Occl.	Total	Defocus				Displacement			Occlusion			
							Full		Partial		Slow	Med.	Fast	Full		Partial	
							L	H	L	H				T	S	T	S
DSS: Outdoor	216	7.4	97	9	110	216	3	86	4	4	3	3	3	27	46	11	26
DSS: Indoor	62	1.5	31	7	24	62	7	5	6	13	4	3	0	4	2	6	12
Traffic: Multi lanes	20	0.3	5	4	11	20	2	0	0	3	2	0	2	3	2	3	3
Traffic: Single lane	52	4.2	26	13	13	52	0	26	0	0	0	10	3	0	11	0	2
Total	350	13.4	159	33	158	350	12	117	10	20	9	16	8	34	61	20	43

Table 1 Dataset summary, L – low defocus, H – high defocus, T – textured occlusion, S – textureless occlusion.

backgrounds, large objects in front of the camera (like a crowd or vehicle), and texture occlusion pose the greatest challenges.

5.2. Accuracy

We use following definition for every video in a dataset:

- TP (true positive) – We detect tampering on the video when tampering event occurs.
- FN (false negative) – We do not detect tampering on the video.
- FP (false positive) – Tampering is detected before tampering event occurs.

The algorithm parameters in the previous works are not provided completely; therefore, it is difficult to assess a quality of those approaches on our dataset. We took the results from [15] and performed experiments on the Traffic: Singe lane dataset with frame downscaled to several different resolutions (Table 2). Parameters were configured automatically using the approach described above. Moreover, we used ignore angles for displacement for the Traffic: Single lane dataset (Fig. 5) to prevent displacement false alarms.

Algorithm	Precision	Recall
Saglam et al. [5]*	0.47	0.88
Ribnick et al. [2]*	0.5	0.98
Wang et al. [7]*	0.5	0.98
Wang et al. [15]*	1	0.96
Proposed approach (40x30)	1	0.75
Proposed approach (80x60)	1	0.90
Proposed approach (160x120)	1	0.96
Proposed approach (240x180)	1	1

Table 2 Accuracy results on the Traffic: Single Lane dataset (* - taken from [15]).

	DSS: Outdoor	DSS: Indoor	Traffic: Multi lanes	Traffic: Single lane
Precision	1	1	1	1
Recall	0.98	0.97	0.96	1
Occlusion min area, %	52.5	39.5	45.5	99.5
Stat. occlusion min area, %	41.0	7.5	19.0	76.0
Defocus max ratio	0.761	0.784	0.881	0.105

Table 3 Accuracy results for 240x180 frames.

Frame size	Atom E3900, avg. FPS	Core i7-6700K, avg. FPS
40x30	807	2230
80x60	377	852
160x120	179	515
240x180	125	355

Table 4 Performance results.

Our proposed approach detects only 75 percent of tampering events from Traffic: Single lane dataset when processing frames at 40x30 but has very high performance of ~807 FPS on Intel Atom® E3900 processor series. At a resolution of 240x180 frames, this approach achieves the best results on the Traffic: Single lane dataset.

At resolution 240x180 frames, some tampering events are not detected (four events in DSS: Outdoor, two events in DSS: Indoor, and one event in Traffic: Multi lanes). All of these events are either light defocus events or partial defocus events in the non-textured area of the frame. We can conclude, therefore, that our approach can skip only complex tampering events under a zero false alarm rate condition. Notice that there seems to be no value in increasing the resolution beyond QVGA since it does not seem to result in any accuracy gains. That makes sense since the method does not use any fine-grained details on the image to evaluate metrics.

In Table 3, we provide the algorithm parameters for every dataset. For example, for the DSS: Outdoor dataset, we define all objects with area more than 52.5 percent as occlusion objects, and we signal defocus event if edge count is less than 76.1 percent of normal (median) level.

The Traffic: Single lane dataset has contrasting parameters compared to the others because it is very difficult to distinguish tampering and non-tampering events in some cases. For example, a large bus in front of the camera looks like an occlusion, and asphalt without markings looks like defocus or occlusion event.

Fast tampering detection time is quite important for real applications. Our proposed approach detects almost all tampering events in 2.5 seconds (Fig. 6). Some events, like slow displacement, take a long time to be detected due to their specifics.

5.3. Performance

Performance and accuracy are a typical trade-off. We performed experiments on Intel® Core™ i7-6700K

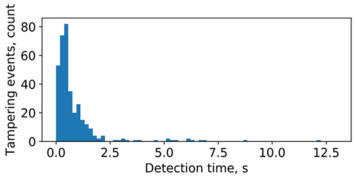

Figure 6: Detection time of tampering events.

4.0GHz and Intel Atom® E3900 1.60GHz processors with different scales (see Table 4). Our proposed approach performance on the Intel Atom® E3900 is ~377 FPS at 80x60 scale (more than 15 FPS consuming only 5 percent of the CPU time) and has relatively high accuracy. On a big enough frame (240x180), the algorithm can work at 6 FPS consuming only 5 percent of the CPU time.

6. Conclusion

We have presented camera tampering detection approaches for different surveillance scenarios. Application of low-level computer vision techniques with our unique parameter adaptation method achieves state-of-the-art results on TCAD dataset [15]. Our approach detects more than 96 percent of malicious events without false alarms for 13 hours of video with 350 tampering events from more than 20 different scenes. Almost all tampering events are detected in 2.5 seconds. The algorithm can process 80x60 frames at more than 15 FPS consuming only 5 percent of the Intel Atom® E3900 CPU time. Implementation of the proposed algorithm is available in Intel® OpenVINO™.

References

[1] S. Harasse, L. Bonnaud, A. Caplier, and M. Desvignes. Automated camera dysfunctions detection. 6th IEEE Southwest Symposium on Image Analysis and Interpretation: 36-40, 2004.

[2] E. Ribnick, S. Atev, O. Masoud, N. Papanikolopoulos, and R. Voyles. Real-Time Detection of Camera Tampering. IEEE International Conference on Advanced Video and Signal Based Surveillance: 10-15, 2006.

[3] A. Aksay, A. Temizel, and A. E. Cetin. Camera tamper detection using wavelet analysis for video surveillance. IEEE Conference on Advanced Video and Signal Based Surveillance: 558-562, 2007.

[4] P. Gil-Jimenez, R. Lopez-Sastre, P. Siegmann, J. Acevedo-Rodriguez, and S. Maldonado-Bascon. Automatic Control of Video Surveillance Camera Sabotage. in Processings of Nature Inspired Problem-Solving Methods in Knowledge Engineering: 222-231, 2007.

[5] A. Saglam and A. Temizel. Real-time Adaptive Camera Tamper Detection for Video Surveillance. IEEE Conference on Advanced Video and Signal Based Surveillance: 430-435, 2009.

[6] C. Alippi, G. Boracchi, R. Camplani, M. Roveri. Detecting External Disturbances on the Camera Lens in Wireless Multimedia Sensor Networks. IEEE Transactions on Instrumentation and Measurement, 59(11): 2982-2990, 2010.

[7] Y.K. Wang, C.T. Fan, K.Y. Cheng, and P. S. Deng. Real-time Camera Anomaly Detection for Real-world Video Surveillance. International Conference on Machine Learning and Cybernetics: 1520-1525, 2011.

[8] D. Ellwart, P. Szczuko, A. Czyzewski. Camera sabotage detection for surveillance systems. Securrentity and Intelligent Information Systems, Springer: 45-53, 2012.

[9] C.-L. Tung, P.-L. Tung, and C.-W. Kuo. Camera tamper detection using codebook model for video surveillance. 2012 International Conference on Machine Learning and Cybernetics, vol. 5: 1760-1763, 2012.

[10] S. Liang, N. Liu, G. Wang, and W. Guo. Camera Sabotage Detection Based on LOG Histogram and Bhattacharyya Distance. Lecture Notes in Electrical Engineering Future Wireless Networks and Information Systems: 197-203, 2012.

[11] T. Kryjak, M. Komorkiewicz, and M. Gorgon. FPGA implementation of camera tamper detection in real-time. IEEE Conference on Design and Architectures for Signal and Image Processing: 1-8, 2012.

[12] D.-T. Lin and C.-H. Wu. Real-Time Active Tampering Detection of Surveillance Camera and Implementation on Digital Signal Processor. 2012 Eighth International Conference on Intelligent Information Hiding and Multimedia Signal Processing: 383-386, 2012.

[13] T. Theodore, L. Christensen, P. Fihl and T. B. Moeslund. Tamper Detection for Active Surveillance Systems. IEEE International Conference on 10th Advanced Video and Signal Based Surveillance: 57-62, 2013.

[14] H. Yin, X. Jiao, X. Luo and C. Yi. Sift-based camera tamper detection for video surveillance. IEEE 25th Chinese Control and Decision Conference: 665-668, 2013.

[15] Y.K. Wang, C.T. Fan, J.F. Chen. Traffic Camera Anomaly Detection. 22nd International Conference on Pattern Recognition (ICPR '14): 4642-4647, 2014.

[16] D.-Y. Huang, C.-H. Chen, T.-Y. Chen, W.-C. Hu, and B.-C. Chen. Rapid detection of camera tampering and abnormal disturbance for video surveillance system. Journal of Visual Communication and Image Representation, 25(8): 1865-1877, 2014.

[17] A. Gaibotti, C. Marchisio, A. Sentinelli, and G. Boracchi. Tampering Detection in Low-Power Smart Cameras. Engineering Applications of Neural Networks Communications in Computer and Information Science: 243-252, 2015.

[18] M. S. Javadi, Z. Kadim, H. H. Woon, M. J. Khairunnisa, and N. Samudin. Video stabilization and tampering detection for surveillance systems using homography. 2015 International Conference on Computer, Communications, and Control Technology (I4CT): 275-279, 2015.

[19] G.-B. Lee, M.-J. Lee, and J. Lim. Unified Camera Tamper Detection Based on Edge and Object Information. Sensors, 15(5): 10315-10331, 2015.

[20] L. Dong, Y. Zhang, C. Wen, and H. Wu. Camera anomaly detection based on morphological analysis and deep learning. 2016 IEEE International Conference on Digital Signal Processing (DSP): 266-270, 2016.

[21] M. Hagui, A. Boukhris, and M. A. Mahjoub. Comparative Study and Enhancement of Camera Tampering Detection Algorithms. 2016 13th International Conference on Computer Graphics, Imaging and Visualization (CGiV): 226-231, 2016.

[22] K. Sitara and B. M. Mehtre. Real-Time Automatic Camera Sabotage Detection for Surveillance Systems. Advances in Intelligent Systems and Computing Advances in Signal Processing and Intelligent Recognition Systems:75-84, 2016.

[23] P. Mantini and S. K. Shah. A signal detection theory approach for camera tamper detection. 2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS): 1-6, 2017.