# Real-time Adaptive Camera Tamper Detection for Video Surveillance

Ali Sağlam and Alptekin Temizel
Graduate School of Informatics
METU
Ankara, Turkey
ali.saglam@tubitak.gov.tr, atemizel@ii.metu.edu.tr

*Abstract* - **Criminals often resort to camera tampering to prevent capture of their actions. Real-time automated detection of video camera tampering cases is important for timely warning of the operators. Tampering is generally done by obstructing the camera view by a foreign object, displacing the camera and changing the focus of the camera lens. In automated camera tamper detection systems, low false alarm rates are important as reliability of these systems is compromised by unnecessary alarms and consequently the operators start ignoring the warnings. We propose adaptive algorithms to detect and identify such cases with low false alarms rates in typical surveillance scenarios where there is significant activity in the scene.**

***Keywords – video surveillance; camera tampering; camera sabotage; covered camera; defocused camera; moved camera***

## I. INTRODUCTION

Surveillance system operators often watch high number of cameras simultaneously and these systems are left unattended at certain times. This results in important events that require immediate actions to be taken are missed such as deliberate attempts to tamper with a camera. In the recent years, computer assisted algorithms which analyze the images from the cameras and warn the operator regarding such events are found to be useful. In such systems, low number of false alarms is necessary. The high numbers of false alarms result in alarms being ignored by the operator and render the system ineffective. Hence, such systems are expected to have low false alarm rate while having high true alarm success rate. Also low computational complexity is required as high number of cameras needs to be observed simultaneously or the algorithms are run on a camera or embedded system with limited computational power.

Camera tampering, which is also often referred to as camera sabotage in the literature, can be defined as deliberate physical actions on a video surveillance camera which compromises the captured images. In this paper we cover the following tampering types:

- **Defocused Camera View:** Focus setting of a camera is changed and this results in reduced visibility.
- **Moved Camera:** Turning a camera to make it point to a different direction.
- **Covered Camera View:** Camera view is covered with a foreign object.

There are only a few methods in the literature which are aimed to detect the above sabotages [1-3]. In [1], wavelet based methods are proposed to detect defocused camera view and covered camera view. However no method is proposed to detect camera displacement. In [2], an edge background model is used to detect camera tampering. However large objects or crowd of people moving in front of the camera could change the image characteristics significantly and cause false alarms. In [3], the authors propose keeping a short term and a long term pool of recent images. With each new image, these pools are updated and three dissimilarity functions are calculated for all the images in these pools. This makes real-time operation difficult due to the high number of computations required. Also this method doesn't identify the type of tampering as any kind of dissimilarity between the short term pool and the long term pool is flagged as tampering.

In this paper we propose algorithms to detect camera tampering. We keep an adaptive background image which is compared with the incoming frames from the video camera and with a delayed background image. We also keep track of the moving areas of the image and use a region based operation to reduce false alarms. This adaptive method is robust to moving objects in front of the camera.

Rest of this paper is organized as follows. In the next section, we explain the background subtraction method. Then the algorithms used to detect different camera tampering types are proposed. In Section 3, detection of defocused camera view is presented. In Section 4, we explain the detection of changed camera view. In Section 5, detection of covered camera view is explained. Following these, in Section 6 the performance of methods in [1-3] are compared to the methods which are proposed in this paper. In the last section conclusions and future work are summarized.

## II. BACKGROUND SUBTRACTION

Background can be defined as stationary parts of the video. Among various methods for subtraction of background in the literature, we based ours on the adaptive background subtraction method of *Video Surveillance And Monitoring (VSAM) System* [4] as it provides a simple, low computational cost alternative. Even though there are more sophisticated background subtraction methods in the literature, we chose this method as the lower computational cost is more important than accuracy for camera tampering

978-0-7695-3718-4/09 $25.00 © 2009 IEEE
DOI 10.1109/AVSS.2009.29

detection applications.

Let $I_n(x,y)$ represent the intensity value at pixel position $(x,y)$ in the $n^{th}$ frame. Estimated background image is represented as $B_{n+1}$ and value at the same pixel position, $B_{n+1}(x,y)$ is calculated as follows:

$$B_{n+1}(x,y) = \begin{cases} \alpha B_n(x,y) + (1-\alpha) I_n(x,y) \\ \quad ,if\ (x,y) is\ not\ moving \\ B_n(x,y), if\ (x,y) is\ moving \end{cases} \quad (1)$$

where $B_n(x,y)$ is the previous estimate of the background intensity value at pixel position $(x,y)$ and $\alpha$ is a positive real number where $0< \alpha <1$. $\alpha$ is selected close to 1 and $B_0(x,y)$ is set to the first image frame $I_0(x,y)$. A pixel is said to be moving if the corresponding intensity values in $I_n$ and $I_{n-1}$ satisfy the following:

$$|I_n(x,y) - I_{n-1}(x,y)| > T_n(x,y) \quad (2)$$

where $T_n(x,y)$ is an adaptive threshold value for pixels positioned at $(x,y)$. After each background update, all the threshold values corresponding to each pixel position is also updated as follows:

$$T_{n+1}(x,y) = \begin{cases} \alpha T_n(x,y) + (1-\alpha)(c|I_n(x,y) - B_n(x,y)|) \\ ,if\ (x,y) is\ not\ moving \\ T_n(x,y), if\ (x,y) is\ moving \end{cases} \quad (3)$$

where $c$ is a real number greater than one. In this study we set initial values of all threshold values to 128. To increase the sensitivity the parameter $c$ is reduced.

## III. Detection of Defocused Camera View

Defocusing the lens of a camera or reduced visibility due to atmospheric conditions such as fog results in degradation of edges in the captured image. In such a case, the high frequency data of the current image $I_n$ will be lower than the high frequency data of the background image $B_n$. In the proposed algorithm, high frequency data in $I_n$ and $B_n$ are compared by using Fourier Transform.

After taking Fourier Transform of the images, a Gaussian windowing function is used to discriminate the higher frequency values from lower frequencies. Let $E_{HF}(I_n)$ be the sum of high frequency values of $I_n$ which is calculated as follows:

$$E_{HF}(I_n) = \sum_{x,y} G(x,y) * F\{I_n(x,y)\} \quad (4)$$

where $G(.)$ is the Gaussian windowing function which will be used to eliminate low frequency values and $F\{.\}$ indicates Discrete Fourier Transform.

Similarly, sum of high frequency values of $B_n$ will be called $E_{HF}(B_n)$ and calculated as follows:

$$\mathrm{E_{HF}}(B_n) = \sum_{x,y} G(x,y) * F\{B_n(x,y)\} \quad (5)$$

After these two functions, all pixels of $I_n$ and $B_n$ are summed to find $E_{HF}(I_n)$ and $E_{HF}(B_n)$. A camera lens is said to be defocused if;

$$E_{HF}(I_n) \leq \mathrm{T}h_1 E_{HF}(B_n) \quad (6)$$

where $0<Th_1<1$ is a threshold, the detection sensitivity increases when $Th_1$ is closer to 1. In this method, $Th_1$ is updated by taking into account the level of detail in the background image before applying (6). If the camera watches a scene with large uniform areas having little detail, the amount of high frequency data is expected to be low. Hence, defocusing camera view doesn't change the amount of high frequency data too much. In this situation, the threshold is set to a number which is closer to 1 to increase the sensitivity. If the camera watches a scene which contains large amount of high frequency data, the method is expected to be more sensitive. In this case, some events may be misinterpreted as defocused. To reduce the number of false alarms, the threshold is set to a number which is closer to 0 to decrease the sensitivity. The threshold is updated according to the following equation:

$$Th_1 = \begin{cases} L_{Bound} \ ,if\ 1 - \left(E_{HF}(B_n)/Max_{HF}\right) \leq L_{Bound} \\ 1 - \dfrac{E_{HF}(B_n)}{Max_{HF}} \\ ,if\ \ U_{Bound} \geq 1 - \left(E_{HF}(B_n)/Max_{HF}\right) \geq L_{Bound} \\ U_{Bound} \ ,if\ 1 - \left(E_{HF}(B_n)/Max_{HF}\right) \geq U_{Bound} \end{cases} \quad (7)$$

where $E_{HF}(B_n)$ shows the high frequency content of $B_n$ and calculated as in (5). We use $B_n$ to find the level of detail in the scene because $B_n$ is more stable than $I_n$. $Max_{HF}$ is an experimentally determined value which depicts the maximum number that $E_{HF}(B_n)$ can take. $L_{Bound}$ and $U_{Bound}$ indicate lower and upper bounds of the $Th_1$ respectively. In our implementation, we set $L_{Bound}$ to 0.4 and $U_{Bound}$ to 0.8.

Even though this method is useful for detecting the reduced visibility, there are cases in typical surveillance scenarios where the amount of high frequency changes significantly which could potentially generate false alarms. When new objects enter to the scene or objects leave the scene, total amount of edges and hence the high frequency content in $I_n$ changes. Fig. 1 shows an example showing the change in high frequency values when new objects enter or leave the scene.

To handle such cases, the algorithm is modified to ignore regularly changing parts of the images. Changing pixels are found and marked using (8) and they are kept in moving state until they are observed to be non-moving for a while.

$$M_n(x,y) = \begin{cases} M_n(x,y) + \beta|I_n(x,y) - B_n(x,y)| \\ \quad ,if\ (x,y)\ is\ moving \\ M_n(x,y) - \gamma|I_n(x,y) - B_n(x,y) + 1| \\ \quad ,if\ (x,y)\ is\ non\ moving \end{cases} \quad (8)$$

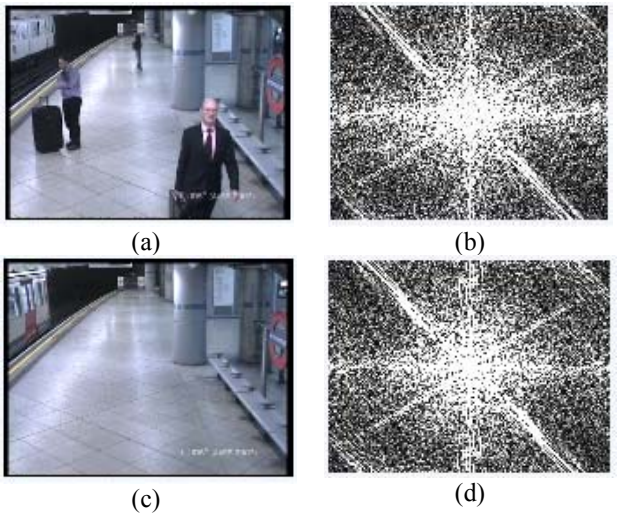


**Figure 1:** (a): An image which contains more high frequency component, (b): Magnitude of the image (a) after FFT, (c): An image which contains less high frequency values, (d): Magnitude of the image (b) after taking FFT.

In this equation, $M_n$ is the image which keeps track of the changing pixels related to $n^{th}$ frame. Initial values of its pixels are set to 0. $\beta$ and $\gamma$ are constants and $\beta$ is selected to be greater than $\gamma$ to make a pixel gracefully non-moving when no motion is observed.

After finding the changing pixels, $I_n$ and $B_n$ divided into 8x8 pixel blocks. Each block is checked for moving pixels, if the block contains any moving pixels, the pixels in this block are excluded from the equation calculating the high frequency content in both $I_n$ and $B_n$. Fig. 2 shows an example of exclusion of moving areas.

**Figure 2:** (a): Image showing changing pixels, $M_n$ where changing pixels are shown white (b): Corresponding image which shows the ignored blocks. Black rectangles show the ignored areas.

Amount of high frequency data in $I_n$ and $B_n$ are calculated using the following equations which excludes the moving blocks. A block is said to be moving if $M_n(x,y)\neq 0$ for any pixel in the block:

$$E_{HF}(I_n) = \sum_{i=0}^{k} \sum_{x,y} G(x,y) * F\{I_n(x,y)\} \qquad (9)$$

$$E_{HF}(B_n) = \sum_{i=0}^{k} \sum_{x,y} G(x,y) * F\{B_n(x,y)\} \qquad (10)$$

where $k$ is the number of blocks not having any moving pixels. Once $E_{HF}(I_n)$ and $E_{HF}(B_n)$ are found, they are compared using the equation 6. $Th_1$ is updated in the same way to equation 7.

## IV. Detection of Moved Camera

Turning a camera to make it point in a different direction is also a type of tampering. When a camera is moved to a different direction, the background image $B_n$ starts to be updated to reflect the changed view. In the proposed algorithm we use another image which holds a delayed background image and represented as $B_{n-k}$ where $k \in Z^+$. $B_{n-k}$ is compared with $B_n$ to find if a camera is moved to point towards a different direction. A proportion value denoted with $P$ is calculated by comparing each pixel on $B_n$ to corresponding pixel on $B_{n-k}$.

$$P = \begin{cases} P+1 \\ ,if\ B_n(x,y) \neq B_{n-k}(x,y) \\ P, if\ B_n(x,y) = B_{n-k}(x,y) \end{cases} \qquad (11)$$

Camera is said to be moved to different direction if $P>Th_2K$ where $0<Th_2<1$ is threshold value which increases sensitivity when it is closer to 0 and $K$ is the total number of pixels. Background is adaptive and updates slowly, hence eliminating the false alarms due to sudden light flickers such as changes due to moving clouds and reflection of sun light from luminous objects.

Fig. 3 shows the background and delayed background images when the camera is turned different direction. As can be seen in this figure, the current background image starts to be updated while the delayed background still has the background image for the previous location.

**Figure 3:** (a): Estimated background image $B_n$, which starts to be updated when camera is turned to different direction, (b): Delayed background image $B_{n-k}$.

Similar to the detection of defocused camera view method, the threshold $Th_2$ is updated adaptively in relation to the amount of high frequency component in the background image.

## V. Detection of Covered Camera View

When a camera view is covered with an object, histogram of $I_n$ is expected to have higher values in a specific range in the histogram compared to the $B_n$. Because most of the scene is occupied by the covering object, significant part of $I_n$ is expected to have the color of the covering object or become darker.

In [1], an algorithm which is used to detect covered camera view is proposed. The algorithm calculates the histograms of $I_n$ and $B_n$. In this algorithm there are two steps. If both of the steps are satisfied, camera view is said to be covered. In this study we changed the first step. Let *max(H(A))* represent the bin number of which value is the maximum value in histogram of image *A*. In the first step the values of maximum bin number and its neighbors of the histogram of $I_n$ and $B_n$ found and compared to check if $I_n$ has a higher peak than $B_n$. In the second step, histogram of the absolute difference $|I_n\text{-}B_n|$ is checked to see if most of the values accumulate near the black end.

32-bin histogram of an image will be called $H_i(.)$ where $1 \le i \le 32$. Both (12) and (13) are checked to find whether a camera view is covered or not.

$$\left(H_{max(H(I_n))-1}(I_n) + H_{max(H(I_n))}(I_n) + H_{max(H(I_n))+1}(I_n)\right) > Th_3\left(H_{max(H(I_n))-1}(B_n) + H_{max(H(I_n))}(B_n) + H_{max(H(I_n))+1}(B_n)\right) \quad (12)$$

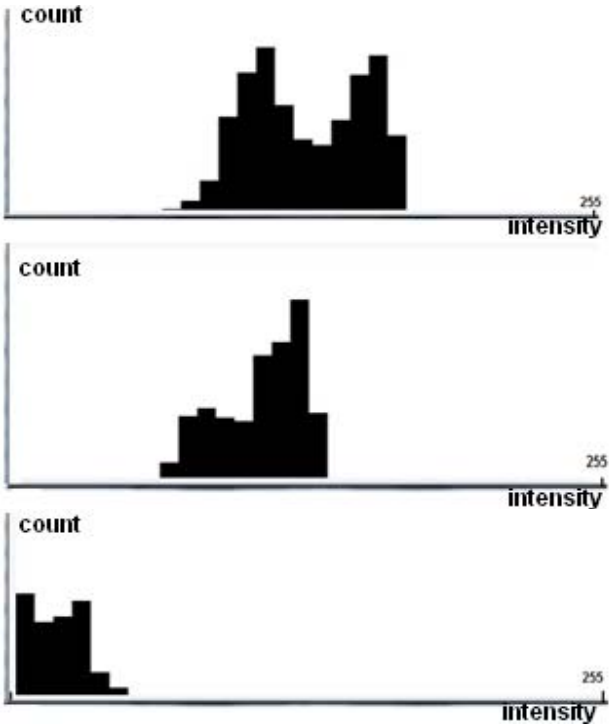
in the above equation, if a bin value is smaller than 1 or greater than 32 their corresponding value is thought as 1 or 32 respectively. The second step of the algorithm is given below:

$$\sum_{i=1}^{32} H_i(|I_n - B_n|) > Th_4 \sum_{i=1}^{k} H_i(|I_n - B_n|) \quad (13)$$

$Th_3 > 1$ and $Th_4 > 1$ are thresholds which can be increased for higher sensitivity of the algorithm. Where $0 \le k < 32$ and when it closer to the lower bound, sensitivity will be higher, typically $k$=3 is found to be generating satisfactory results. Fig. 4 shows the histograms of $I_n$, $B_n$ and $|I_n\text{-}B_n|$ image when a camera view is covered with an object.

## VI. Experimental Results

As there are no datasets available for testing camera tampering detection methods, we captured a range of images to reflect real-life camera tampering scenarios to test the proposed methods and compare with the methods in the literature. Also to test the false alarms, we used some video sequences from the *i-LIDS dataset* [5] which contain typical




surveillance scenarios without any camera tampering.

Tampering types which are explained in section 1 are separately tested. Experimental results will be explained according to these types. Methods explained in [1-3] are firstly implemented and tested with the same set of test videos. In [3], we reduced the number of calculations by keeping the dissimilarity measures in memory and only calculating the dissimilarity for images entering the two buffers. The algorithms have been implemented using C++ language and *OpenCV* library [6,7].

### A. Defocused Camera View

The algorithms were tested on total of 17 videos having 35 defocusing events. The videos were captured in different environments and they contain a variety of test scenarios. Table 1 shows the results of defocused camera view tests.

As seen from the results, [3] gives the highest number of missed events. When we evaluated the test videos in detail we observed that, if the scene contains more red and green objects the sensitivity of this algorithm increases. Since it uses red and green values 2-dimensional histograms and these histograms are used for image dissimilarity, red and green objects in the scene affect the results positively.

TABLE I. DEFOCUSED CAMERA TEST RESULTS FOR DIFFERENT ALGORITHMS

| Algorithms | Number of false alarms | Number of missed events | Percentage of missed events |
|---|---|---|---|
| Aksay et al. [1] | 5 | 7 | 20% |
| Gil-Jiménez et al. [2] | 1 | 13 | 37,1% |
| Ribnick et al. [3] | 0 | 14 | 40% |
| Proposed Method (not ignoring moving blocks) | 27 | 6 | 17,1% |
| Proposed Method (ignoring moving blocks) | 0 | 6 | 17,1% |

Our first method which doesn't ignore the moving parts on the images gives a high number of false alarms. Because moving objects on the scene changes the high frequency content in $I_n$, false alarms may arise. Ignoring the moving blocks reduces the false alarm rate to 0 while not affecting the true alarm rate.

Another important thing to note is the level of detail in the scene. Lack of salient edges makes the detection difficult for all of the methods.

### B. Moved Camera

We used a total of 12 video sequences containing 12 moved camera scenarios. The results are summarized in Table 2.

TABLE II. MOVED CAMERA TEST RESULTS FOR DIFFERENT ALGORITHMS

| Algorithms | Number of false alarms | Number of missed events | Percentage of missed events |
|---|---|---|---|
| Gil-Jiménez et al. [2] | 3 | 4 | 33% |
| Ribnick et al. [3] | 0 | 5 | 41.7% |
| Proposed Method | 0 | 1 | 8.3% |

As explained in the first section, [1] doesn't provide a method for detecting moved camera and hence we weren't able to include in this experiment. Because moving objects in the scene changes the result of Zero-mean Normal Cross Correlation (ZNCC) in [2], calculated correlation between the background and the current image is lower and this results in false alarms when there is high activity.

### C. Covered Camera View

We used a total of 25 video sequences having a 40 covered camera view cases. The results are summarized in Table 3.

TABLE III. COVERED CAMERA TEST RESULTS FOR DIFFERENT ALGORITHMS

| Algorithms | Number of false alarms | Number of missed events | Percentage of missed events |
|---|---|---|---|
| Aksay et al. [1] | 2 | 7 | 17,5% |
| Gil-Jiménez et al.[2] | 14 | 5 | 12,5% |
| Gil-Jiménez et al.[2] *(block based)* | 28 | 0 | 0% |
| Ribnick et al. [3] | 1 | 3 | 7,5% |
| Proposed Method | 0 | 2 | 5% |

In [2], an entropy value is calculated and low entropy values are used to infer covered camera view. It also has an alternative method for partial camera occlusions. In this method, image is divided into blocks and entropy is calculated for each block separately. Even though this increases the true alarm detection rate, as the entropy of individual blocks are affected significantly by moving objects, it generates considerably more false alarms and has the highest false alarm rate.

We tested these 3 detection types with a variety of camera tampering scenarios as well as 4 typical video surveillance footages not containing any tampering events totaling approximately 14 minutes from *i-LIDS dataset* [5]. Our results show that the proposed methods have very low false alarm rates and more favorable true alarm rates in moved and covered camera cases compared to the other algorithms. In the defocused camera case, results of [1] has 3% higher true alarm detection rate. However it generates significant number of false alarms while our method doesn't generate any false alarms.

The test videos don't include sudden illumination changes which could potentially cause false alarms. Even though not a likely scenario -considering the placement of real life CCTV cameras-, large objects moving in front of the camera and covering most of the view could increase the number of false alarms.

## VII. CONCLUSION

In this paper, we proposed camera tampering detection methods to detect occluded camera view, defocus and camera displacement. These methods are designed to be adaptive to changing conditions and to work in cases where there is high degree of motion. The low computational complexity of the algorithms makes them suitable for real-time operation in multi-camera systems as well as embedded systems. Experimental results are presented which shows high true alarm detection rate with very low false alarm rate compared to the other methods in the literature. The proposed methods also identify the type of tampering type.

## REFERENCES


[1] A. Aksay, A. Temizel and A.E. Cetin, "Camera Tamper Detection Using Wavelet Analysis for Video Surveliance", *IEEE International Conference on Video and Signal Based Surveillance*, September 2007.

[2] P. Gil-Jimenez, R. Lopez-Sastre, P. Siegmann, J. Acevedo-Rodriguez, and S. Maldonado-Bascon, "Automatic Control of Video Surveillance Camera Sabotage", *IWINAC 2007*, 2007.

[3] E. Ribnick, S. Atev, O. Masoud, N. Papanikolopoulos, and R. Voyles, "Real-Time Detection of Camera Tampering", *IEEE International Conference on Video and Signal Based Surveliance*, November 2006.

[4] R.T. Collins, A.J. Lipton, T. Kanade, H. Fujiyoshi, D. Duggins, Y. Tsin, D. Tolliver, N. Enomoto, O. Hasegawa, P. Burt, L. Wixson, "A system for video surveillance and monitoring:VSAM final report", *Technical Report*, Carnegie Mellon University, May 1998

[5] i-LIDS dataset for AVSS 2007[Cited: 13 March 2009], http://www.elec.qmul.ac.uk/staffinfo/andrea/avss2007_d.html

[6] *OpenCV Documentation and FAQs.* [Online] [Cited: 13 March 2009.] http://opencvlibrary.sourceforge.net/.

[7] G. Bradski and K. Adrian., *Learning OpenCV,* O'Reilly, September 2008.