Real-time Multi-Camera Video Analytics System on GPU

Puren Guler, Deniz Emeksiz, Alptekin Temizel, Mustafa Teke, Tugba Taskaya Temizel

Graduate School of Informatics, Middle East Technical University Ankara, Turkey

e170986@metu.edu.tr, e171000@metu.edu.tr, atemizel@metu.edu.tr, mustafa.teke@gmail.com, ttemizel@metu.edu.tr

Abstract – In this paper, parallel implementation of a real-time intelligent video surveillance system on Graphics Processing Unit (GPU) is described. The system is based on background subtraction and composed of motion detection, Camera Sabotage Detection (Moved Camera, Out-of-Focus Camera and Covered Camera Detection), Abandoned Object Detection and Object Tracking algorithms. As the algorithms have different characteristics, their GPU implementations have different speed-up rates. Test results show that when all the algorithms run concurrently, parallelization in GPU makes the system up to 21.88 times faster than the CPU counterpart, enabling real-time analysis of higher number of cameras.

Keywords – Video Surveillance, Video Analytics, Real-time, CUDA, GPU.

Abbreviations – CPU: Central Processing Unit; GPU: Graphics Processing Unit; CCD: Covered Camera Detection; MCD: Moved Camera Detection, OOFCD: Out-of-Focus Camera Detection; VMD: Video Motion Detection; GMM: Gaussian Mixture Model; IAGMM: Improved Adaptive Gaussian Mixture Model; VSAM: Video Surveillance and Monitoring; AOD: Abandoned Object Detection; OT: Object Tracking

I. INTRODUCTION

Keeping the security of public places has become increasingly more important in the recent years. Video surveillance systems take a significant role in security management of sites as well as towns and cities. In traditional video surveillance systems, there should be an operator actively watching the videos captured by the security cameras. The operator is expected to focus on all the cameras incessantly not to miss potential sabotage attempts, to track individuals for any potential suspicious behavior and to detect potentially dangerous abandoned objects. However, this is not a realistic expectation considering the high number of cameras. To overcome this problem, automated systems that can detect camera tampering, abandoned objects and track individuals in real time to raise an alarm to inform the operators have been put into use. These large-scale video surveillance systems should have the following properties to be of practical use:

- They should have low false alarm - high true alarm rates,
- The computational cost should be low to allow concurrent running of different automated analysis and detection algorithms on all the required cameras.

Video surveillance algorithms typically run on embedded systems –running algorithms on a few cameras- or on centralized locations where a number of servers are utilized for processing. Since these systems should work in real-time and high number of cameras needs to be analyzed, the computational cost of the algorithms tends to be high, prohibiting large-scale deployment.

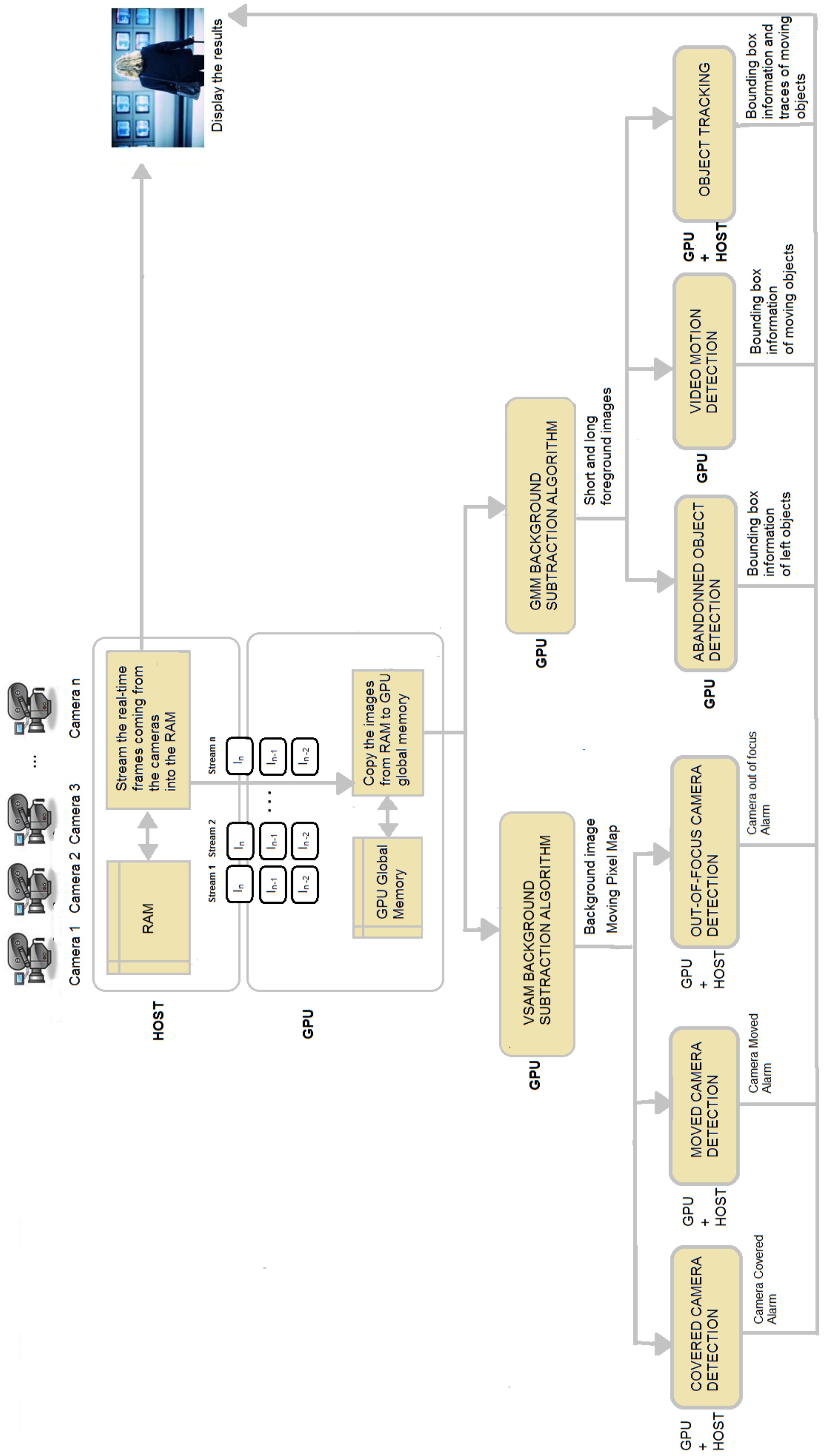
There are a number of video processing systems implemented on Graphics Processing Unit (GPU) leveraging the computational performance gain of these systems. In the method of [15], the authors develop a fast system that performs multiple image processing tasks by parallelizing a number of stereo vision algorithms. In the method of [18], a multi-cue face tracking system is proposed where different cues are combined under a particle filter tracking system and because of the high number of the particles; they implement a parallel system with GPU to achieve better performance. In the study of [16], a fast face tracking system boosted on GPU is proposed. A computer vision library implemented in GPU is presented in [2]. In this library, various well known computer vision algorithms are implemented in GPU such as computing image pyramids. However in the literature, - to best of our knowledge- there is no complete GPU based system implementing the whole set of video analytics algorithms used for real-time video surveillance.

Video surveillance systems typically consist of static and PTZ (Pan-Tilt-Zoom) cameras. The PTZ cameras are either controlled manually by the operators or used in patrol mode whereby the camera position is changed at regular intervals to predetermined positions. Hence in these systems, static camera assumption can be made thus allowing use of background subtraction based methods. Any camera motion not initiated by the operator or the system itself can be assumed a camera tampering event.

In this paper, we propose a GPU based automated video analysis system for video surveillance allowing processing of up to 21.88 times higher number of cameras compared to a Central Processing Unit (CPU) based implementation. The system is implemented using CUDA and composed of the following components which are typically needed in a real-time video analytics system:

- Video Motion Detection (VMD)
- Camera sabotage detection:
  - Moved Camera Detection (MCD)
  - Out-of-Focus Camera Detection (OOFCD)
  - Covered Camera Detection (CCD)
- Abandoned Object Detection (AOD)
- Object Tracking (OT)

The overall diagram of the system is shown in Figure 1. Images from different cameras are captured by the system in real-time and placed into the CPU memory. Then these images are transferred into the GPU global memory asynchronously. The aforementioned algorithms are run on the GPU using the images in GPU memory. The asynchronous transfer allows simultaneous processing and transfer, i.e. the current image is processed in the GPU while the next image is transferred from CPU memory into the GPU memory.



Camera 1
Camera 2
Camera 3
...
Camera n
HOST
RAM
Stream the real-time frames coming from the cameras into the RAM
Display the results
GPU
Stream 1
Stream 2
Stream n
I_n
I_{n-1}
I_{n-2}
GPU Global Memory
Copy the images from RAM to GPU global memory
GPU
VSAM BACKGROUND SUBTRACTION ALGORITHM
Background image
Moving Pixel Map
GPU
GMM BACKGROUND SUBTRACTION ALGORITHM
Short and long foreground images
GPU + HOST
COVERED CAMERA DETECTION
Camera Covered Alarm
GPU + HOST
MOVED CAMERA DETECTION
Camera Moved Alarm
GPU + HOST
OUT-OF-FOCUS CAMERA DETECTION
Camera out of focus Alarm
GPU
ABANDONNED OBJECT DETECTION
Bounding box information of left objects
GPU
VIDEO MOTION DETECTION
Bounding box information of moving objects
GPU + HOST
OBJECT TRACKING
Bounding box information and traces of moving objects

Figure 1. Overall system diagram.

The camera streams can be processed independently and no synchronization is required between the different camera streams. In the CPU, parallelism is achieved at camera level; a separate thread is assigned to each camera and each thread processes the aforementioned video analytics algorithms in a serial fashion. All the cameras have the same thread priority and context switch is handled by the Windows OS scheduler. In the GPU version, images are processed in the order of their arrival, regardless of which camera they belong to. The optimization in the GPU is twofold; (i) parallelization of the algorithms and benefiting from the parallel architecture on the GPU, (ii) generating a more efficient data flow and preventing replication of computation by intertwining the various algorithms and sharing/reusing the computed data.

In section II, a literature survey regarding these algorithms and GPU implementations is presented. Implementation of VSAM background subtraction and camera tamper detection algorithms on the GPU are explained in detail in section III. In section IV, V and VI Improved Adaptive Gaussian Mixture Model (IAGMM), abandoned object detection and object tracking-video motion detection algorithms are described with their GPU implementations respectively. In section VII, the performance analysis of the implemented algorithms in GPU and their comparison with the CPU counterparts are presented. Concluding remarks are given in the final section.

II. RELATED WORK

In automated video surveillance systems, the fundamental step is estimating a reliable background image and updating this background image to reflect any changes in the environment. In pixel based approaches, each pixel is modeled separately. While this method is able to obtain detailed shape information of moving objects, the segmentation is very sensitive to changing background. In block based approaches, the image is divided into blocks and the features of these blocks are used for background modeling. By this way, the problem of sensitivity to changing background is reduced, since blocks are less sensitive to changes in the background than individual pixels. On the other hand, it cannot obtain detailed shape of moving objects. In order to benefit from the advantages of these two methods, a hierarchical method combining these two approaches was proposed by the authors of [6]. In the method of [19], they use artificial neural network algorithm for background subtraction. Their aim is to develop an algorithm that enables obtaining a background through a self- learning manner. Then, based on the learnt model, moving objects can be detected. In [3] seven commonly used methods in background subtraction algorithms are compared. The authors of [17] try to overcome the drawbacks of Gaussian Mixture Model which are requirements of large memory space and "slow convergence". They develop an online learning method for improving Gaussian Mixture Model (GMM). According to their approach, Expectation Maximization method is combined with a recursive filter where each Gaussian has a learning parameter that gets closer to basic recursive learning after many observations.

Although, above methods are successful in extracting a background with acceptable accuracy, they are not able to overcome challenges like non-static background, different whether conditions, camera noise and extracting the full shape of the foreground objects with their inner pixels. On the other hand, Improved Adaptive Gaussian Mixture Model (IAGMM) [34] is a robust method and also it is computationally not very complex.

Camera sabotage detection (tampering) algorithms aim to detect deliberate attempts by a person to make captured video unusable and can be analyzed in 3 categories: detection of “Out-of-Focus Camera”, “Moved Camera” and “Covered Camera”. In the method of [27], the first step for performing camera sabotage detection is estimating a background image. For background estimation, the method described in Video Surveillance and Monitoring (VSAM) report of [7] is used. According to VSAM, background is estimated by adaptively updating previous background images. Background estimation allows comparing the estimated background with the past background information to detect any suspicious event. To detect the covered camera view, the histogram of brightness values of the current image pixels and the estimated background pixels are compared. If the camera is moved, the current and the previous background images are expected to be different. So, a delayed background image is also maintained as a reference and the current background image is compared against the delayed background image. If the camera is out-of-focus, the amount of high frequency data of the current image should be lower than that of the background image and detection is done using this information.

In another approach, two pools are kept, one of them, named short-term pool, holds new frames and the other, long-term pool, holds older frames. The camera tamper detection is done by comparing the frames in these two separate pools [26]. In [12], they tackle the camera sabotage problem by keeping a background image containing edges of the objects. For covered camera detection, entropies of the pixels in the background image are calculated. If the camera is covered, the entropy is expected to be lower than usual. If the camera gets defocused, the edges are expected to disappear causing the pixel count in the background image to get lower. For the detection of camera movement, a block-matching algorithm with 2 parameters is used. The first parameter gets the optimal value of the axes of the background and current image while the second parameter is the replacement of current and background images. Among these methods, we adopt the method of [27], as it is proven to handle the sabotage cases with high success rates.

There are various algorithms proposed for abandoned object detection algorithms in the literature. A method using background subtraction to detect objects and ground-truth homography to handle occlusions is described in [1]. Abandoned objects are detected when objects move away from each other. In the method of [13], object tracking, detection of drop-off events and stationary object detection methods are used in conjunction for abandoned object detection. A tracking based method, where the abandoned object is detected by observing the object split is given in the method of [11]. Color and shape information of static foreground objects are fused for unattended object detection by the method of [28]. Method described in [33], which is based on the method of [25] makes use of two foreground images, one long-term and one short-term. If the object’s pixels are in the long-term foreground, but not in the short-term background, then this object is accepted as an abandoned object. In order to detect left objects in the scene, long-term and short terms foregrounds’ pixels are subtracted from each other. In this work we adopt this algorithm due to its robustness and lower computational requirement. While the processing power of GPU allows high computational requirement algorithms to be run, a sufficiently robust algorithm with lower computational requirement allows supporting higher number of cameras and more preferable.

Another algorithm that has been parallelized within this project is object tracking. Object tracking is required in video surveillance systems to detect cases such as intrusions, counter-flows and also used as input to suspicious behavior detection algorithms. In this work, we adopt an object correspondence based approach as it provides satisfactory results when the camera is stationary [10]. This approach also allows the object tracking algorithm to be designed together with the motion detection algorithm, since initial steps required for the object tracking can also be used for motion detection, eliminating the need for running a separate algorithm.

In motion detection, the purpose is detecting the moving objects in the scene and tracking of these objects is not required. For motion detection, we utilize a number of steps of the object tracking algorithm. Connected component analysis is applied on the foreground objects' pixels extracted using IAGMM algorithm in order to eliminate the noise and to connect the separate parts of an object. These operations are the post-processing operations in object tracking algorithm used to obtain accurate shapes of the tracked objects.

In the proposed system, for abandoned object detection, object tracking and video motion detection algorithms, IAGMM algorithm is used for background estimation, since for these algorithms, a more accurate background image that can extract the foreground objects with inner pixels is required. For camera tamper detection algorithms, an algorithm with less accuracy is sufficient, hence background subtraction method proposed in VSAM, which has lower computational complexity, has been adopted for these algorithms. Whenever possible, the same estimated background image is reused in different algorithms for optimization purposes.

III. CAMERA TAMPER DETECTION

This section describes the camera tamper detection algorithms, namely covered camera, moved camera and out of camera detection. The section starts with description of the background subtraction algorithm as it is the basis of these algorithms.

Background Subtraction

Background subtraction is used to detect the non-moving parts in a video sequence and it is the primary step for the following camera sabotage algorithms. For the camera sabotage detection algorithms, as it is not necessary to segment the objects, a basic adaptive background subtraction method is sufficient. Hence an easier to implement and computationally less demanding algorithm is more preferable. For this purpose, we decided to use Video Surveillance and Monitoring (VSAM) algorithm which is specified in VSAM Final Report of [7]. The algorithm is adaptive and updates the background with each new frame to account for any changes in the background. The steps of the algorithm are shown in Figure 2.

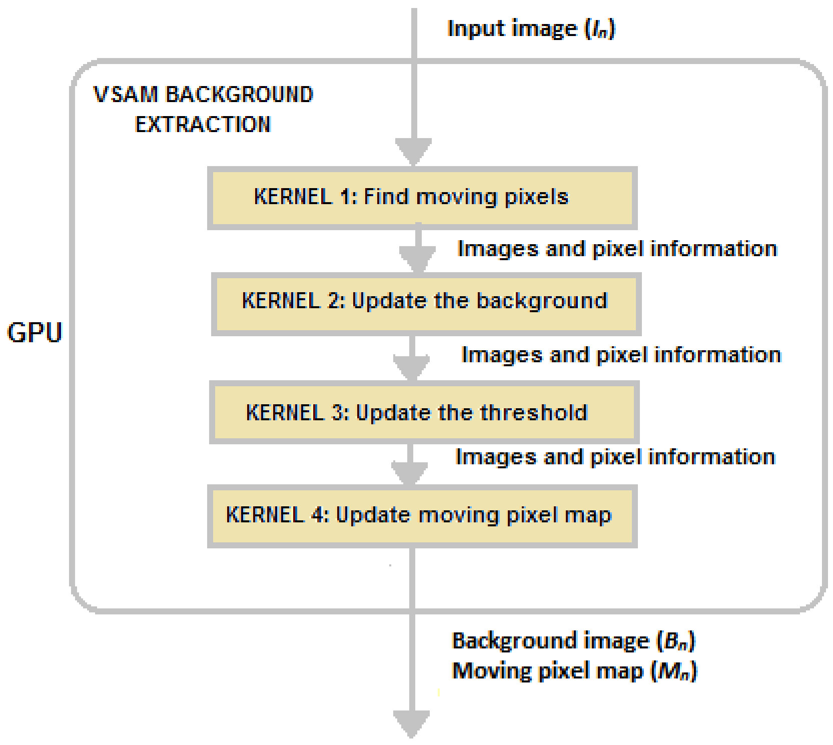

Input image (I_n)
VSAM BACKGROUND EXTRACTION
GPU
KERNEL 1: Find moving pixels
Images and pixel information
KERNEL 2: Update the background
Images and pixel information
KERNEL 3: Update the threshold
Images and pixel information
KERNEL 4: Update moving pixel map
Background image (B_n)
Moving pixel map (M_n)

Figure 2. VSAM background subtraction algorithm.

To integrate background subtraction into the system, we have started with the optimized implementation developed in our lab [30]. In this implementation, the inputs of the kernel are integer pointers of current frame, previous frame and frame before previous frame. Outputs are integer pointers of array of adaptive thresholds values, background image, array that indicates the blocks that contain moving pixels, moving pixel array and foreground array. For optimization of memory access time, in this kernel, the 8-bit char pointers of frames are converted to 32-bit integer pointers, so that in each thread 4 char pixels are loaded at once. In the original implementation, moving pixels were extracted by comparing the current and previous two frames which is similar to foreground extraction. However, in our system, more precise moving object detection is necessary in order to disregard the parts that change frequently. In this study, this algorithm is modified using the moving pixel extraction algorithm described in the method of [30] and parallelized. Due to the embarrassingly parallel nature of this algorithm, each four pixel group is processed in a thread and equation 1 is applied in a loop for each pixel in each thread.

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

In this equation$M_n(x,y)$, $I_n(x,y)$ and $B_n(x,y)$ are *n*th moving pixel map, input frame and background frame respectively. As a result of this operation, image pixels corresponding to moving objects are marked.

Another change from the implementation of [30] is the method of marking the blocks that contain moving pixels in the frame. As described in the method of [30] in camera out-of-focus detection algorithm, the frame is split into 8x8 blocks and the blocks that contain moving pixels are marked so that these blocks can be excluded from the summation of high frequency content. To maintain a list of moving blocks, a 2-D look up table with size of *(frame width/8) x (frame height/8)* is created. This process is illustrated in Figure 3. The input frame that is divided into 8x8 blocks is shown in Figure 3 (a). Figure 3 (b) shows the look-up table that keeps track of 8x8 blocks that contains at least one moving pixel. If in the frame's 8x8 block there is at least one moving pixel, then the corresponding index for the same block in the look-up table is set to 1. Later, while the frame is processed, first the look-up table is checked and if according to this table, there are blocks that have moving pixels, these blocks of the frame are excluded from the high frequency calculation since they give unreliable information about high frequency content in the scene.

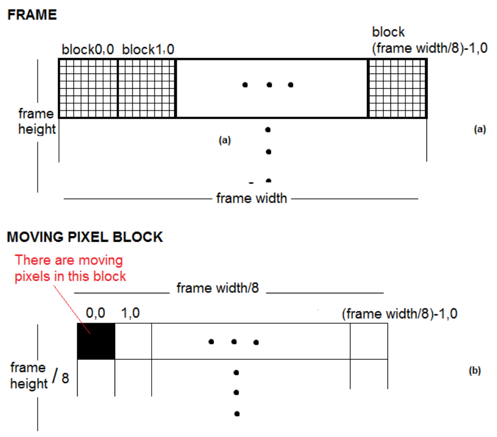


Figure 3. If there is a moving pixel in an 8x8 block (a), that block is marked as moving (b).

Covered Camera Detection

For detection of covered camera, current frame (I_n)'s histogram and background image (B_n)'s histogram are calculated. If the view of the camera is obstructed by an object, values in a particular range in I_n's histogram should be higher than the values in B_n's histogram. For example, if the covering object is black, the black pixel value should be higher in histogram of I_n than in the histogram of B_n because of there is no black object being in the background image. Camera is said to be covered if both the equations 2 and 3 are satisfied [27]. These equations require calculation of histogram for I_n, B_n and $|I_n - B_n|$.

$$\begin{aligned}&\left(H_{\max(H(I_n))-1}(I_n) + H_{\max(H(I_n))}(I_n) + H_{\max(H(I_n))+1}(I_n)\right)\\ &> Th_1\left(H_{\max(H(I_n))-1}(B_n) + H_{\max(H(I_n))}(B_n) + H_{\max(H(I_n))+1}(B_n)\right)\end{aligned} \quad 2$$

$$\sum_{i=1}^{32} H_i(|I_n - B_n|) > Th_2 \sum_{i=1}^{k} H_i(|I_n - B_n|) \quad 3$$

In these equations, $Th_1>1$, $Th_2>1$ and $0 \le k < 32$ are thresholds which are selected according to the desired sensitivity of the algorithm.

Figure 4 shows the steps of the covered camera detection algorithm.

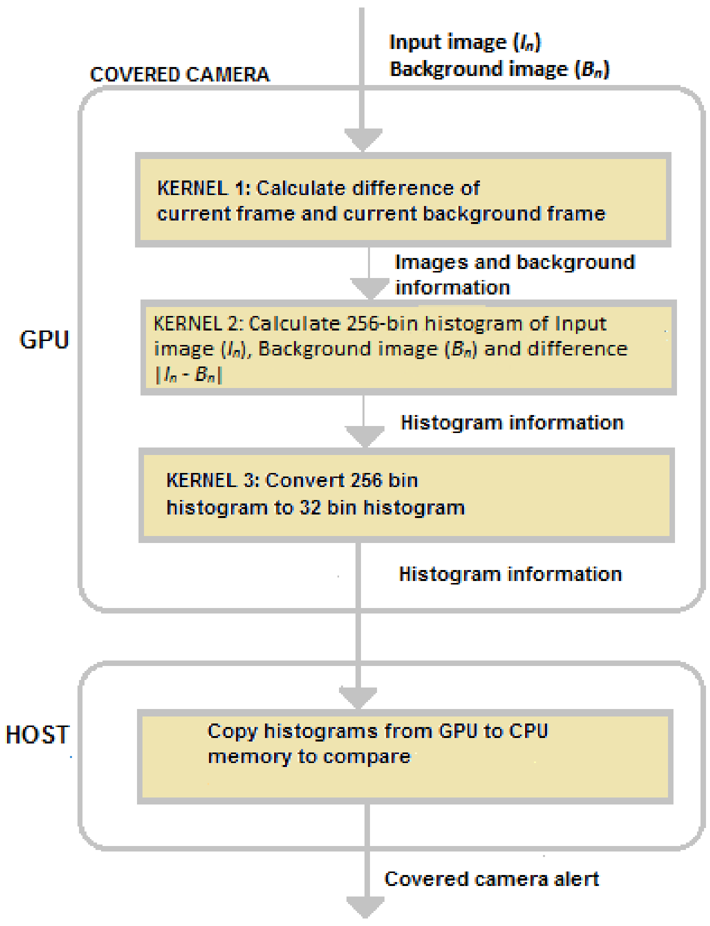

COVERED CAMERA
Input image (I_n)
Background image (B_n)
GPU
KERNEL 1: Calculate difference of current frame and current background frame
Images and background information
KERNEL 2: Calculate 256-bin histogram of Input image (I_n), Background image (B_n) and difference $|I_n - B_n|$
Histogram information
KERNEL 3: Convert 256 bin histogram to 32 bin histogram
Histogram information
HOST
Copy histograms from GPU to CPU memory to compare
Covered camera alert

Figure 4. Covered camera detection algorithm.

For these operations, firstly the difference image $|I_n - B_n|$ needs to be calculated. Calculation of difference image in GPU benefits from the parallel architecture as each pixel pair can be calculated independently. After obtaining the difference image, histograms are generated and equations 2 and 3 are applied to determine if the camera is covered. The histogram calculation is a high-cost operation because it iterates through all the pixels of the image. By parallelizing the histogram generation algorithm, covered camera detection can be speeded up.

According to the test results, histogram calculation process gets slower when number of bins is fewer; compared to 32, 64 or 128-bin histograms, 256-bin histogram is more efficient to calculate. In covered camera detection, 32-bin histograms are needed, so after generating 256-bin histograms, 32-bin histograms are generated by summing up the values corresponding to the bins in 256-bin histograms. For example, to calculate 1st bin in 32-bin histogram, values of 1st to 8th bins in 256-bin histogram are summed up.

Moved Camera Detection

In this algorithm, the purpose is to detect movement of the camera. It is expected that the background image starts to be updated with a change in the camera direction, resulting in a difference between the current and the delayed background. For this purpose, pixel by pixel difference of current and delayed background frames is calculated. Then, non-moving pixels are counted to obtain a proportion value of non-changing pixels which is then compared with a threshold. The threshold is adaptive and changes with the content of high frequency information in the image.

For the threshold calculation, amount of high frequency information of the background image needs to be computed. Firstly, the FFT of the image is calculated and then in order to eliminate low frequency information, the image is convolved with a Gaussian kernel. For this purpose, FFT-based 2D convolution implementation of [24] is used. The resulting high frequency values of current background image are summed using a sum reduction kernel to benefit from parallelization. The moving areas detected using equation 1 and stored in moving pixel map M_n. These areas are masked out and excluded from the summation as the moving objects could increase the total amount of high frequency information. The result of this operation is used to update the threshold as explained in [27].

Equation 4 is used for calculation of proportion of moved pixels [27].

$$P = \begin{cases} P + 1, if\ B_n(x,y) \neq\ B_{n-k}(x,y) \\ P, if\ B_n(x,y) =\ B_{n-k}(x,y) \end{cases} \qquad 4$$

Where P, B_n and B_{n-k} are proportion of moved pixels, *n*th background frame and *k* frame delayed background frame respectively. For this calculation, firstly a separate thread is assigned to each pixel to calculate the binary difference of B_n and B_{n-k}. This binary image, which has value of 1 at locations where B_n and B_{n-k} are different and 0 otherwise, is passed to a sum reduction kernel to calculate *P*. Camera is said to be moved if $P>Th_3K$ where $0< Th_3<1$ is threshold value which increases sensitivity when it is closer to 0 and *K* is the total number of pixels.

For sum reduction, the final optimized kernel in reduction implementation by CUDA is used [14]. The implementation is changed according to requirements of the Fermi architecture. Usually, threads in the same block do not need *__syncthreads()* for synchronization. But, in Fermi architecture, pointer with volatile qualifier is required for proper communication of threads in the same warp without *__syncthreads()* [21]. Thus, *volatile* qualifier should be used in order to write the values into the shared memory instead of registers.

The result provided by the sum reduction kernel is given to another kernel to calculate the total sum. The resulting kernel's dimension does not exceed 1024 in this work. Thus, another sum reduction kernel is created whose grid size does not exceed 1 and block size does not exceed 1024.

Figure 5 shows the steps of the moved camera detection algorithm.

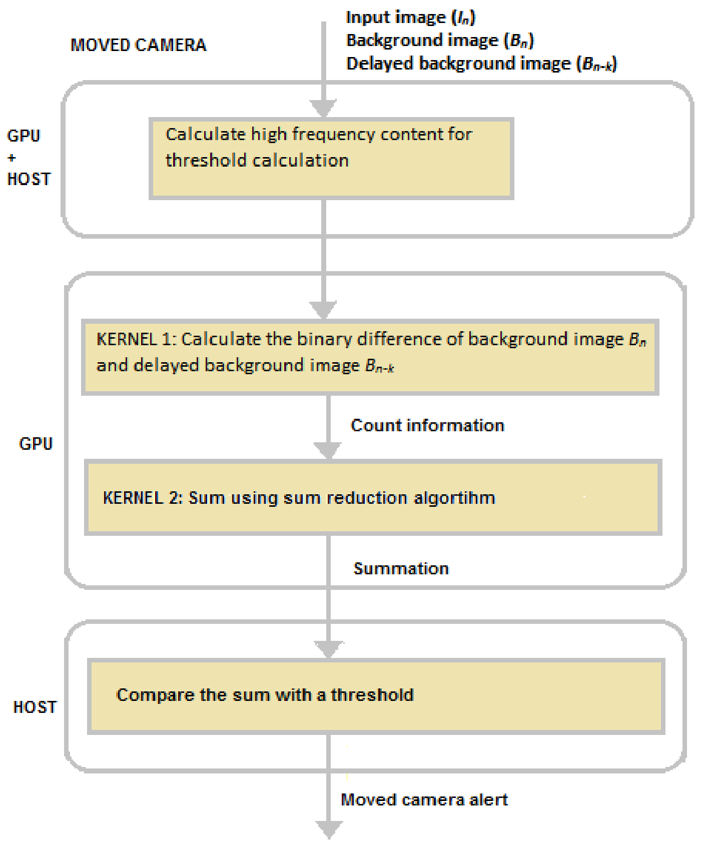

MOVED CAMERA
Input image (I_n)
Background image (B_n)
Delayed background image (B_{n-k})
GPU + HOST
Calculate high frequency content for threshold calculation
GPU
KERNEL 1: Calculate the binary difference of background image B_n and delayed background image B_{n-k}
Count information
KERNEL 2: Sum using sum reduction algortihm
Summation
HOST
Compare the sum with a threshold
Moved camera alert

Figure 5. Moved camera detection algorithm.

Out-of-focus Camera Detection

In order to detect defocusing in cameras, high frequency content of the current frame and the background frame are compared, since in the case of defocusing, high frequency content of the current frame is expected to be lower than that of the background frame [27]. In the original algorithm, for high frequency calculation, Discrete Fourier Transform (DFT) is used. In this study, we utilize Discrete Cosine Transform (DCT) implementation of NVIDIA GPU Computing SDK 4.0 [22] and modify the algorithm accordingly. The frames are first divided into 8x8 blocks and blocks that contain moving pixels (calculated previously and stored in M_n) are excluded from the DCT calculation for optimization purposes.

In the original implementation, memory is allocated inside the DCT function, requiring a new allocation for each frame. In our implementation, memory allocations are moved outside the function so that they are not done for each frame. Also, we parallelized *CopyByte2Float* and *AddFloatPlane* functions and combined them into a single kernel to eliminate passing of data between these kernels. These are functions that come with dct8x8 implementation of SDK [22]. *CopyByte2Float* function copy a byte array to float array. For proper conversion from byte to float -128.0f value is added to the float array in *AddFloatPlane* function.

Another alteration is using a custom 8x8 kernel instead of the quantization kernel used as default. We implemented a Gaussian kernel following DCT to obtain similar results to [27] where DFT is used instead. The aim of the window is eliminating low frequencies and retaining high frequency information to allow subsequent high frequency content calculation. Figure 7 shows the kernel used in this implementation which suppresses the low frequency content (at the top left of an 8x8 block) and retains high frequency content (at the bottom right of an 8x8 block).

After multiplying Gaussian window multiplication, high frequency content is summed inside a kernel. In this kernel, moving block information (*M_n*) is also used to ignore high frequency data in a block that has moving pixel(s) since moving pixels can be misleading in detecting defocusing. In each thread, if the value in the *M_n* is not 1 (which means there is no moving pixel, so the high frequency information for this block is reliable), the frequency value in this block is summed in a loop and put into the corresponding index of an array. Later, the output of the kernel is summed using sum reduction and the result of the summation becomes high frequency data for that frame. This operation is applied to both current frame and background frame and then the high frequency data that comes from these frames are compared to detect defocusing [27].

Figure 6 shows the steps of the out of focus camera detection algorithm.

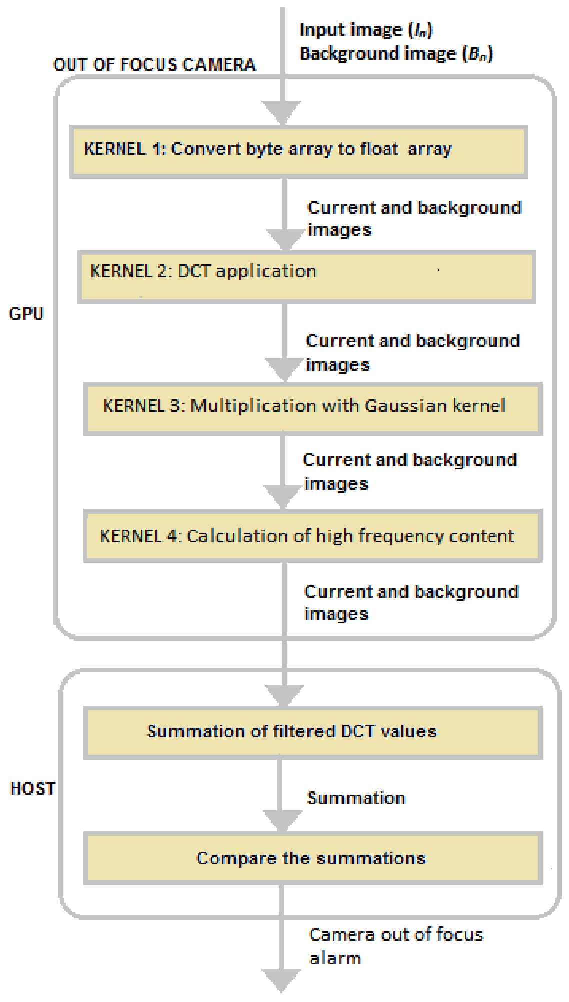

Input image (I_n)
Background image (B_n)
OUT OF FOCUS CAMERA
KERNEL 1: Convert byte array to float array
Current and background images
KERNEL 2: DCT application
GPU
Current and background images
KERNEL 3: Multiplication with Gaussian kernel
Current and background images
KERNEL 4: Calculation of high frequency content
Current and background images
Summation of filtered DCT values
HOST
Summation
Compare the summations
Camera out of focus alarm

Figure 6. Out-of-Focus detection algorithm.

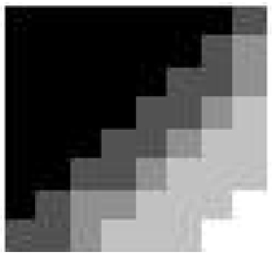


Figure 7. New kernel that is created using Gaussian kernel values and replaces the quantization kernel in GPU implementation.

IV. GAUSSIAN MIXTURE MODEL BACKGROUND SUBTRACTION

In the system, we use the Improved Adaptive Gaussian Mixture Model (IAGMM) Background Subtraction. It is a pixel-based method and through a Bayesian decision, it is decided that whether the pixel belongs to the foreground or background [34].

For GPU implementation of GMM, the implementation that is introduced in [31] is used. We haven't done any modifications to this algorithm for this work.

Since each pixel is processed independently in GMM, the parallelization is done by assigning each pixel to a thread. Global memory holds the output frames and background model and constant memory holds the configuration parameters. In this implementation, shared memory which is faster than global memory is not used due to its limited capacity for storing input and output data for all threads in the block. 4 bytes of dynamic shared memory is used to hold common parameters for the block.

V. ABANDONED OBJECT DETECTION

For the abandoned object detection, we adopt the algorithm proposed by the authors of [25]. Figure 8 shows the steps of this algorithm. In this method, GMM is used for background modeling and two background models, long-term and short-term, are used. These background models are initialized with the same parameters except the learning parameter. The short-term background model is set a higher learning rate and the long-term background model is et with a lower learning rate. By using these two foregrounds, an evidence image *E* is obtained using Equation 12. Evidence image is used for deciding whether a pixel is an abandoned object's pixel.

$$E(\mathrm{x,y}) = \begin{Bmatrix} E(x,y)+1 & F_L(x,y)=1 \wedge F_S(x,y)=0 \\ E(x,y)-k & F_L(x,y)\neq 1 \vee F_S(x,y)\neq 0 \\ max_e & E(x,y)>max_e \\ 0 & E(x,y)<0 \end{Bmatrix} \qquad 12$$

Where *E*(*x*,*y*) is the evidence image pixel value at (*x*,*y*), *k* is a decay constant and max_e is the maximum value of a pixel in the evidence image. According to the equation 12, if a pixel belongs to a foreground object in the long-term background model and not in the short-term background model, the pixel value at the same coordinate is increased in the evidence image. The pixel is said to belong to an abandoned object if it reaches max_e and evidence pixel value is not allowed to exceed this maximum limit. If the pixel does not belong to a foreground object in the long term background model and belongs to a foreground object in the short term background model, this pixel is said to belong to a moving foreground object and the respective evidence pixel value is decreased by *k*. The evidence pixel value is not allowed to take any value lower than 0.

In GPU implementation, for background modeling, GPU implementation of GMM that is described previously is used. For detecting abandoned objects, a separate kernel which gets long-term, short-term foregrounds is implemented. Output of this kernel is a binary image with abandoned objects' pixels. In this kernel, in each thread, a char pixel of the image is processed. In each thread, equatuon 12 is applied to each pixel, generating the evidence image . Then, detected abandoned object pixels are marked on the output image. Finally, noise reduction is done using connected components analysis and dilation operation is applied to mark the abandoned objects.

Connected component labeling(CCL) implementation is based on Stava and Benes' implementation [32] which requires input images to have sizes of powers of 2. This implementation is modified to work with other images sizes for this work. In segment analysis part, we implemented component boundary calculation as a post processing step along with calculation of component sizes.

Using the results of CCL region analysis, sizes and boundaries of components are determined. Sizes of the components are used to eliminate regions. In Figure 10, regions which have sizes smaller than 15 pixels(Region 0, 2 and 4) do not have bounding boxes. While *AtomicMin()* and *AtomicMax()* operations are used to determine the boundaries of connected components, *AtomicAdd()* operation is used for calculation of region sizes.

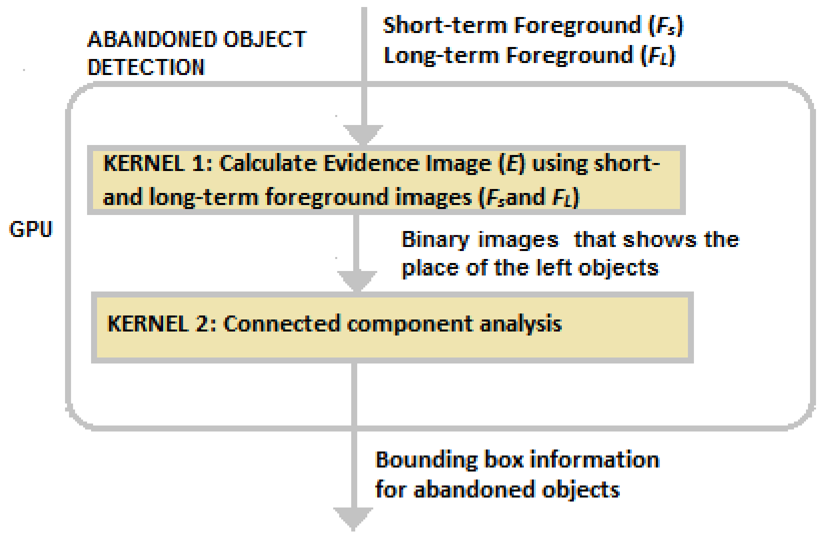

ABANDONED OBJECT DETECTION
Short-term Foreground (F_s)
Long-term Foreground (F_L)
GPU
KERNEL 1: Calculate Evidence Image (E) using short- and long-term foreground images ($F_s$and F_L)
Binary images that shows the place of the left objects
KERNEL 2: Connected component analysis
Bounding box information for abandoned objects

Figure 8. Abandoned object detection algorithm.

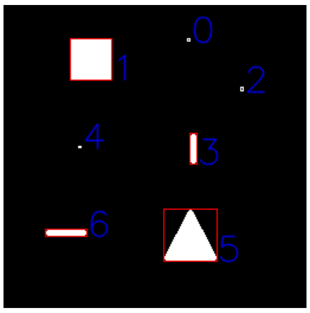
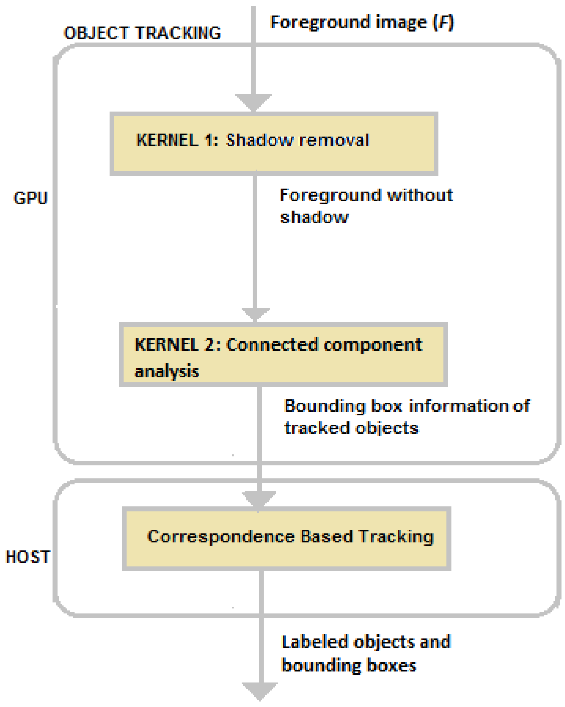


Figure 9. Connected Component Labeling demo results.

VI. OBJECT TRACKING

The object tracking algorithm has been designed as a fully automatic algorithm and object initiation is handled using the extracted foreground information. This algorithm starts with background subtraction using IAGMM. Since shadows may decrease the tracking accuracy, shadow removal is applied to the foreground frame. For this, the method that is described by [8] is used. According to this work, the hue and saturation of shadow pixels do not change significantly in the background while saturation of the foreground decreases. Thus, by comparing saturation, hue and value information of pixels that are classified as foreground, shadows are detected and they are considered as background. After that, noise is removed from the foreground frame. This operation is done using connected component algorithm where bounding box of the object, the number of pixels of each connected component and area of the object are calculated. Then using these information density of the connected components are calculated. A density threshold is defined and connected components that have lower densities than this threshold are eliminated. Detected bounding boxes are compared according to method in [4] to find the object correspondence between frames. The matching is done using the size and distance criteria. To detect occlusion and split the occlusion method proposed in [4] is used. Steps of this algorithm is shown in Figure 10.

Finding the correspondence between the objects requires a synchronization barrier as it requires all the object feature data which makes efficient parallelization in GPU difficult. So this operation is executed on the CPU.



OBJECT TRACKING
Foreground image (*F*)
GPU
KERNEL 1: Shadow removal
Foreground without shadow
KERNEL 2: Connected component analysis
Bounding box information of tracked objects
HOST
Correspondence Based Tracking
Labeled objects and bounding boxes

Figure 10. Object tracking algorithm.

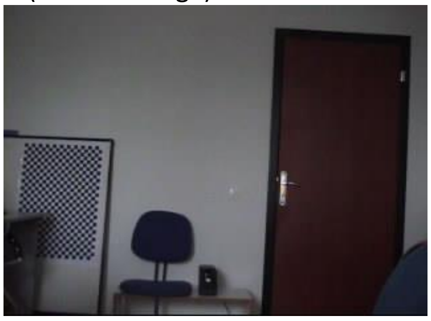
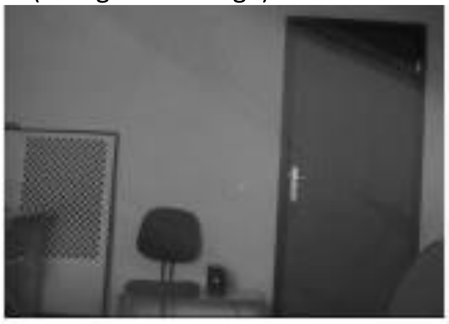
Video Motion Detection

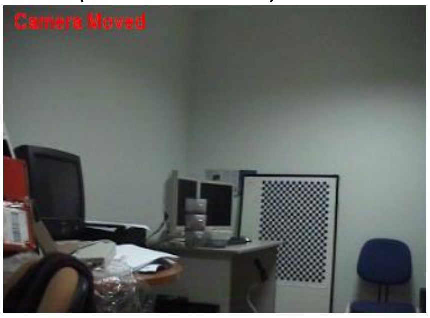
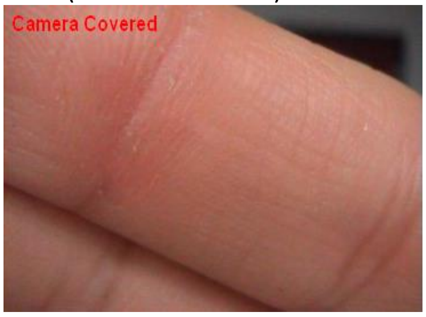
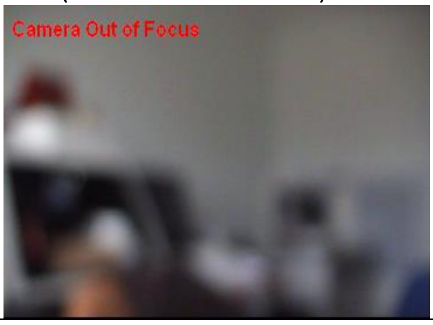
Video Motion Detection algorithm can be considered as a simpler version of object tracking whereby detected objects are not matched in consecutive frames. Hence optimization can be achieved by reusing the same intermediate outputs. In this method, moving objects are detected through foreground images extracted using GMM background. Moreover, since the only purpose is detecting moving objects on the frame, there is no need to apply pre-processesing such as dilation operation. Connected component algorithm, which restricts the detected objects according to the designated size, is sufficient for noise removal process. The output is the bounding box information of moving objects.

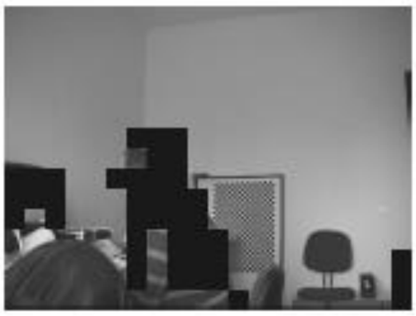
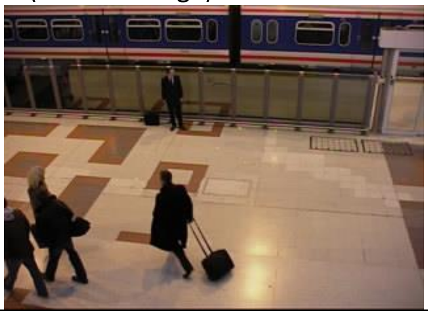
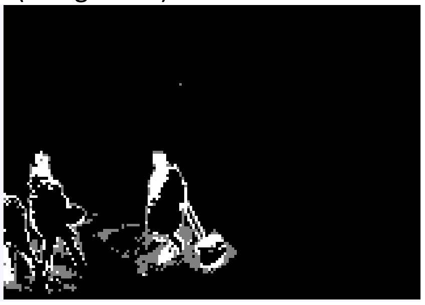
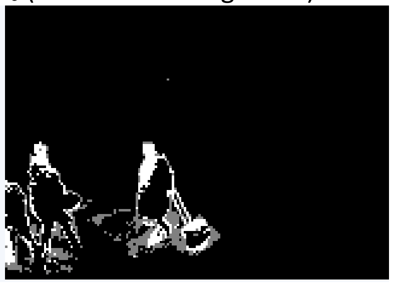
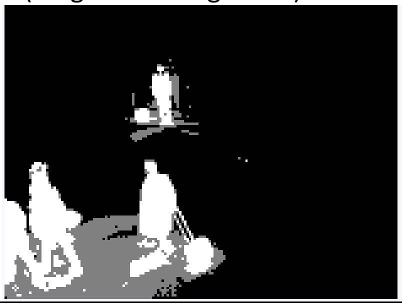
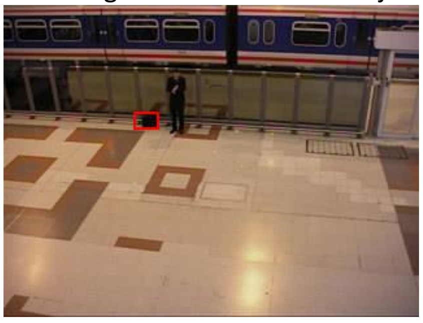
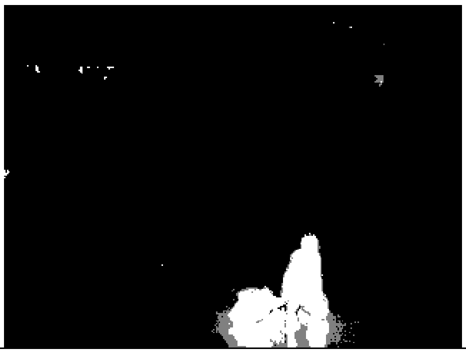
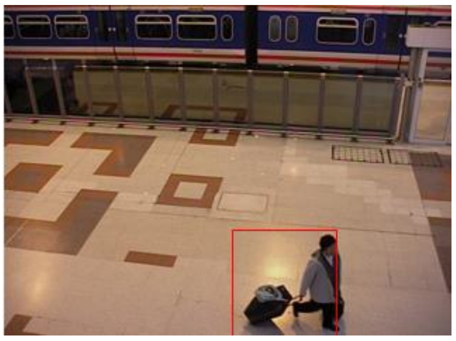
VII. EXPERIMENTAL RESULTS

Experiments were performed on a PC having Intel Core i7 CPU and 3.5 GB usable RAM. The GPU algorithms were tested with NVIDIA Tesla C2075. In time measurement tables, time measurements of CPU and Tesla C2075 are compared. Tests were run 10 times on both platforms and averaged to obtain final results. For calculating the run times of the individual algorithms, elapsed time from the launch of the kernel until its termination is measured. Similarly on the CPU side, the elapsed time is measured from the time the function is called until it terminates. In the calculation of total number of cameras supported, the system is simulated with camera streams. The supported camera is assumed the highest number of cameras at which the system still satisfies the real time constraint. *x*-axis in tables are illustrated in milliseconds in logarithmic scale. Also, gained speed-ups of each graphics board are specified. Speed-up is calculated by dividing CPU time with GPU time. In figures below, the comparison of measured times for CPU and Tesla C2075 in different image sizes are illustrated. These tables and figures are given for each algorithm and the whole system when all algorithms are run. In the end, while processing the same algorithm, maximum numbers of cameras which a computer can hold are specified for CPU and Tesla C2075. CPU execution time increases linearly with the size of the image while GPU is more efficient with larger image sizes as better parallelization and hence higher occupation can be achieved.

For Camera Tamper Detection and VSAM algorithms, the video named CameraCovered4_ByHand_25.avi in our dataset [5] is used. For tracking algorithms (Object Tracking, Video Motion Detection and Abandoned Object Detection), the video S1-T1-C, 3th Scene in [23] is used (Table 2). Sample results of each algorithm with the corresponding input images are shown in Figure 11. In this figure, on the left column, the inputs are shown. The outputs of the algorithms are shown on the right column. For the VSAM algorithm, the input is I_n which is the current image received from the camera, the algorithm updates the background and produces an updated background image B_n. For the moved camera detection algorithm, the inputs are background image and delayed background image, which are generated by the VSAM algorithm. In the figure current background image B_n and a background image delayed by 20 frames (B_{n-20}) are shown. The output of this algorithm is "camera moved" alarm. Camera covered detection algorithm gets the current image I_n and background image B_n as input and generates "camera covered" alarm as output. Out-of-focus camera detection algorithm gets the current image I_n and the moving pixel map generated by the background subtraction algorithm as input and generates "out-of-focus camera" alarm as output. GMM background subtraction algorithm has the same input and output structure with the VSAM background subtraction algorithm, i.e. gets I_n as input and generates updated background B_n as output. Abandoned object detection algorithm requires both a short term and long term background image generated by GMM algorithm and its output is a bounding box marking the location of the detected abandoned object. VMD algorithm takes the long term foreground image processed with shadow elimination algorithm as input and its output is bounding boxes marking the location of moving objects. The final algorithm, object tracking also uses the same shadow eliminated long term foreground image and its output is bounding boxes corresponding to moving objects and their trace information (i.e. their temporal position information).

	Input	Output
VSAM	I_n (current image) 	B_n (background image) 

MCD	B_n (background image) B_{n-20}(delayed background image)	Camera Tampering (Camera Moved) Alarm 
CCD	I_n (current image) B_n (background image)	Camera Tampering (Camera Covered) Alarm 
OOFCD	M_n (moving pixel map)	Camera Tampering (Camera Out of Focus) Alarm 

	I_n with moving area mask 	
GMM	I_n (current image) 	F (foreground) 
AOD	F_s (short-term foreground)  F_L(long-term foreground) 	Bounding box of abandoned object 
VMD	F_L(long-term foreground) with shadow elimination 	Bounding box of moving objects 

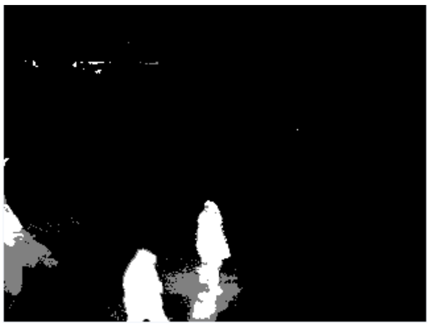
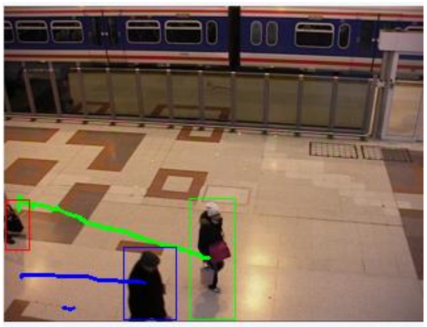
OT	F_L(long foreground) with shadow elimination 	Trace and bounding box of moving object 

Figure 11 - Sample input and output images for each algorithm. Dark gray areas in the foreground images mark the detected shadows.

Table 1 – Videos used in testing.

	Number of Frames	Duration	Dataset
CameraCovered4_ByHand_25.avi	850	00:00:34	Camera Sabotage Discovery Test Videos [5]
S1-T1-C, 3th Scene	2370	00:03:57	PETS2006 Dataset [23]

Table 2 shows the measured time for all the algorithms both on the CPU and GPU for different image sizes. The results show that for all the algorithms, increasing image size results in higher speed up in GPU. The speed up rates vary significantly for different algorithms, for 1024x768 images, while the best speed up rate of 101.87 times is achieved for out-of-focus camera detection, the least speed up is for covered camera detection which is 3.63 times. Results for different algorithms and overall system are discussed in detail in the remainder of this section.

For background subtraction, even in small images which have a size of 160x120, GPU is more than 4 times faster than CPU. As the images get bigger, CPU gets really slow. When the frame size gets 4 times of the previous size, CPU implementation gets 16 times slower and when the frame size reaches 640x480, the time to process a frame gets 8.25 seconds. But in GPU implementation, even in 1024x768 images, the time to process a frame is 0.34 milliseconds and speed-up reaches 75.

For covered camera detection, GPU is 2.12 and 3.63 times faster than CPU for 160x120 and 1024x768 images respectively. This algorithm does not present satisfactory speed up rates in GPU as some parts the algorithm are run on the CPU due to their non-parallel nature and the atomicAdd() function used in histogram calculation. The speed of atomicAdd() function, which is used in histogram calculation, differs according to the architecture of graphics boards and it is known that the newer architectures –such as the NVIDIA Kepler architecture- have substantially improved atomic operations. According to the test results the most of the time is spent in histogram calculation.

Table 2 - Time measurements per frame (ms) and speed-up values for the various algorithms on CPU and GPU for different frame sizes.

Resolution /Algorithm	160x120			320x240			640x480			1024x768		
	GPU (ms)	**CPU (ms)**	**Speed-up**	**GPU (ms)**	**CPU (ms)**	**Speed-up**	**GPU (ms)**	**CPU (ms)**	**Speed-up**	**GPU (ms)**	**CPU (ms)**	**Speed-up**
Background Subtraction	0.12	0.56	**4.67**	0.15	2.07	**13.8**	0.20	8.25	**41.25**	0.34	25.50	**75.00**
Covered Camera Detection	0.83	1.76	**2.12**	1.22	3.45	**2.83**	3.03	10.18	**3.36**	7.65	27.77	**3.63**
Moved Camera Detection	1.28	3.19	**2.49**	1.69	13.27	**7.85**	2.87	62.03	**21.61**	4.94	177.64	**35.96**
Out-of-Focus Camera Detection	1.01	8.52	**8.44**	1.19	34.17	**28.71**	2.10	150.91	**71.86**	4.24	431.91	**101.87**
GMM Background Update Algorithm	0.07	1.14	**16.29**	0.12	4.54	**37.83**	0.37	17.5	**47.30**	0.90	44.80	**49.78**
Video Motion Detection	1.20	1.20	**1.00**	1.47	4.82	**3.28**	3.12	18.26	**5.85**	5.52	47.25	**8.56**
Object Tracking	2.15	2.95	**1.37**	3.12	10.01	**3.21**	7.13	36.64	**5.14**	16.10	94.45	**5.87**
Abandoned Object Detection	1.26	2.41	**1.91**	1.56	9.35	**5.99**	3.47	38.78	**11.18**	6.40	104.19	**16.28**

For moved camera detection, while the GPU is 2.49 times faster than CPU at 160x120 resolution, the speed-up reaches to 35.96 when the resolution is 1024x768.

The most time consuming parts of the out-of-focus camera detection algorithm are the Fast Fourier Transfor (FFT) calculation and Gaussian window multiplication of FFT values followed by summing up high frequency content for each frame. Due the the efficiency of parallel FFT implementation on GPU and suitability of subsequent window multiplication and summing up operations, satisfactory speed-up values are achieved. For out-of-focus camera detection algorithm, at 160x120 resolution, the GPU has speed-up of 8.44. This increases with the increasing image resolution and speed-up of 101.87 is achieved at 1024x768 resolution.

For GMM background update algorithm, OpenCV CUDA implementation is used [31]. While there is 16.29 times speed up for 160x120 images, it reaches 49.78 for 1024x768 resolution.

The most time consuming parts of the VMD algorithm are GMM algorithm and connected component labelling algorithm. For VMD, while there is no speed up for 160x120 resolution, the speed up of 8.56 is achieved for 1024x768. While the GMM algorithm has high speed-ups, overall VMD speed up is adversely affected due to the connected component labelling algorithm.

Object tracking algorithm is mostly dependent on the GMM algorithm, connected component labelling and object correspondence matching. Due to the serial nature of object correspondence matching, this operation is done on the CPU. Similar to the VMD, object tracking performance is also adversely affected by the connected component labelling.

Abandoned Object Detection (AOD) algorithm uses two foreground images (obtained using short and long-term backgrounds). Hence, this process takes significant time on CPU due to the process of updating two background models. GPU version benefits from the faster GMM implementation.

Total Measured Time

Total measured time is the time elapsed when all 6 detection algorithms (Covered Camera Detection, Moved Camera Detection, Out-of-Focus Camera Detection, Video Motion Detection, Abandoned Object Detection and Object Tracking) run concurrently. For this measurement, Camera Tamper Detection (CCD, MCD, OOFCD) and VSAM algorithms are run once a second (once every 25 frames) and GMM algorithm is run for each frame to keep an updated reference background. Object Tracking algorithm is run at 5 frames/second which is sufficient to obtain trail of a moving person for most video surveillance scenarios. As can be seen in Figure 12, tor 160x120 images, GPU is 4.44 times faster than CPU. As the image resolution gets higher, the speed-up increases and speed-up of 21.82 is achieved at 1024x768 resolution. Figure 13 shows the total number of cameras that can be supported in real-time when all the algorithms run concurrently for all the cameras. In the CPU case, each camera is processed by a separate thread and when there is higher number of cameras than the number of threads that can be processed concurrently by the processor, there will be context switches. The system has been simulated with camera streams fed to the corresponding threads so the number of cameras supported takes the context switch and other operating system overheads. In the GPU version, images are processed in the order of their arrival regardless of their camera number. In GPU version, parallelization is done at image level. With GPU, in the smallest image size (160x120) 56.42 cameras can be processed at the same time compared to 12.7 cameras with CPU (Figure 13). At 1024x768 resolution, the GPU is able to process 7.1 cameras in real-time while CPU fails to process any cameras in real time.

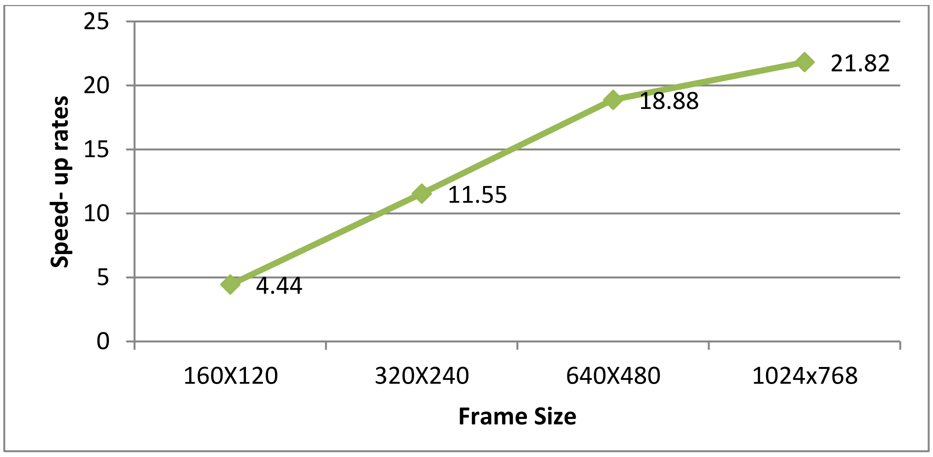


Figure 12. Speed up comparison between GPU and CPU implementations of overall system.

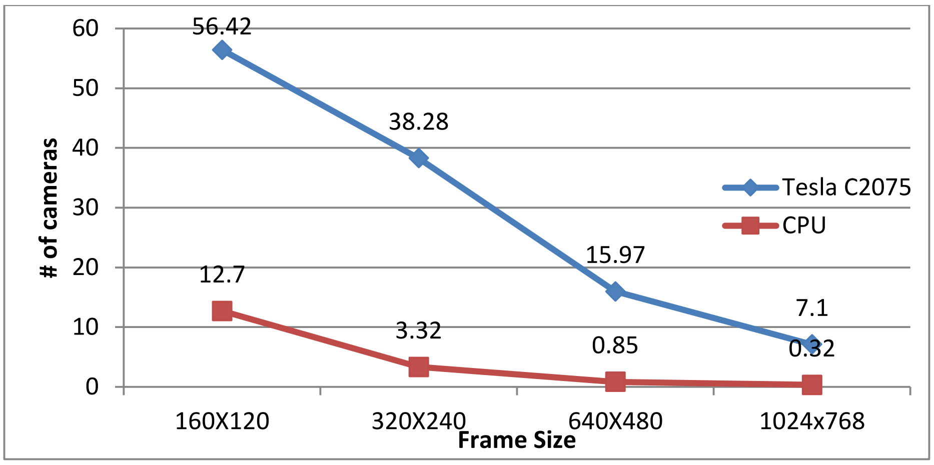


Figure 13. Total number of supported cameras on CPU and GPU in real-time when all the algorithms are run concurrently.

VIII. CONCLUSION

In this study, implementation of a real-time multi camera video analytics system on GPU is described. The system is comprised of camera tamper detection, motion detection, abandoned object detection, and object tracking algorithms. These components are based on background estimation and assumes that the cameras are static and the camera field of view should include some static areas for reliable background estimation.

As the system does not bring any heavy load to the CPU, the remaining CPU resources can be used for other purposes such as system I/O and integration, user interface, video recording as well as running video analytics.

Higher speed-up values could be achieved with larger images sizes in GPU as better parallelization and higher occupation of GPU resources can be achieved, while the speed-up of the overall system is 4.44 at 160x120 resolution, this increases to 11.55, 18.88 and 21.82 for 320x240, 640x480 and 1024x768 resolution respectively. With the ever increasing image resolution used in the systems, it is expected that the benefits of the GPU will be more pronounced in the future.

As the cameras are processed independent of each other, the system is scalable and with the GPUs having higher computational throughput, the number of cameras that can be processed will also be increased without any development effort. This independency also enables multiple GPU configurations in a single computer where higher number of cameras can be supported on a single computer.

For camera tamper detection algorithms, the optimization results are satisfactory. However, for the algorithms that use CCL which are object tracking, abandoned object detection and video motion detection, the obtained optimization is lower than expected. Therefore, CCL implementation could be improved for better performance gain in these algorithms.

ACKNOWLEDGEMENT

This research was funded by Ministry of Science, Industry and Technology SAN-TEZ program grant number 00542.STZ.2010-1. We also would like to thank NVIDIA for their donation of Tesla C2075 GPU boards.

REFERENCES

1. Auvinet, E., Grossmann, E., Rougier, C., Dahmane, M. and Meunier, J.: Left-luggage detection using homographies and simple heuristics. Proceedings of IEEE Int. Workshop on Performance Evaluation of Tracking and Surveillance, pp. 51–58 (2006)

2. Babenko, P. and Shah, M. MinGPU: MinGPU: a minimum GPU library for computer vision. Journal of Real-Time Image Processing. 3(4), 255–268 (2008)

3. Benezeth, Y., Jodoin, P.-M., Emile, B., Laurent, H., & Rosenberger C.: Comparative study of background subtraction algorithms. J. Electron. Imaging, 19(3) (2010)

4. Beyan, C., Temizel, A.: Adaptive Mean-Shift for Automated Multi Object Tracking. IET Computer Vision. 6(1), 1–12 (2012) doi:10.1049/iet-cvi.2011.0054

5. Camera sabotage discovery test video database of Graduate School of Informatics, Middle East Technical University (2009): ftp://ftp.vrcv.ii.metu.edu.tr/Datasets/Camera%20Covered, ftp://ftp.vrcv.ii.metu.edu.tr/Datasets/Camera%20Moved, ftp://ftp.vrcv.ii.metu.edu.tr/Datasets/Camera%20Out%20of%20Focus

6. Chen, Y.-T., Chen, C.-S., Huang, C.-R., & Hung, Y.-P.: Efficient hierarchical method for background subtraction. Pattern Recognition, 40(10), 2706–2715 (2007)

7. Collins, R. T., Lipton, A., Kanade, T., Fujiyoshi, H., Duggins, D., Tsin, Y., et al.: A System for Video Surveillance and Monitoring: VSAM Final Report. Carnegie Mellon University. (2000).

8. Cucchiara, R., Grana, C., Piccardi, M. and Prati, A.: Detecting objects, shadows and ghosts in video streams by exploiting colour and motion information. Proceedings of ICIAP, pp. 360–365 (2001)

9. Dedeoglu, Y.: Moving Object Detection, Tracking and Classification for Smart Video Surveillance. Master's thesis, Bilkent University, Department of Computer Engineering, Turkey.

10. Emeksiz D. and Temizel A.:"A Continuous Object Tracking System with Stationary and Moving Camera Modes, Proceedings of SPIE Security+Defense, Electro-Optical and Infrared Systems: Technology and Applications, , Sept. 2012.

11. S. Ferrando, G. Gera, M. Massa, and C. Regazzoni: A new method for real time abandoned object detection and owner tracking. Proceedings of IEEE Int. Conf. on Image Processing, pp. 3329–3332 (2006).

12. Gil- Jiménez, P., López-Sastre, R., Siegmann, P., Acevedo-Rodríguez, J., & Maldonado-Bascón, S.. Automatic Control of Video Surveillance Camera Sabotage. IWINAC '07 Proceedings of the 2nd international work-conference on Nature Inspired Problem-Solving Methods in Knowledge Engineering: Interplay Between Natural and Artificial Computation, Part II. (2007)

13. S. Guler and M. K. Farrow, “Abandoned object detection in crowded places,” in Proc. of IEEE Int. Workshop on Performance Evaluation of Tracking and Surveillance, pp. 99–106 (2006).

14. Harris, M: Optimizing Parallel Reduction in CUDA (2008). http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/reduction/doc/reduction.pdf

15. Hosseini, F., Fijany, A., Safari, S., & Fontaine, J.-G.: Fast implementation of dense stereo vision algorithms on a highly parallel SIMD architecture. Journal of Real-Time Image Processing (2011)

16. Ishii, I., Ichida, T., Gu, Q., & Takaki, T.: 500-fps face tracking system. Journal of Real-Time Image Processing (2012)

17. Lee, D.-S.: Effective Gaussian mixture learning for video background subtraction. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 27(5), 827–832 (2005)

18. Liu, K.-Y., Li, Y.-H., Li, S., Tang, L., & Wang, L.: A new parallel particle filter face tracking method based on heterogeneous system. Journal of Real-Time Image Processing (2011)

19. Maddalena, L., & Petrosino, A.: A Self-Organizing Approach to Background Subtraction for Visual Surveillance Applications. Image Processing, IEEE Transactions on, 17(7), 1168–1177 (2008)

20. NVIDIA Corporation: CUDA CUFFT Library. http://developer.download.nvidia.com/compute/cuda/3_1/toolkit/docs/CUFFT_Library_3.1.pdf

21. NVIDIA Corporation: Fermi Compatibility Guide for CUDA Applications. http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/Fermi_Compatibility_Guide.pdf

22. Obukhov, A., & Kharlamov, A.: Discrete Cosine Transform for 8x8 Blocks with CUDA (2008). http://developer.download.nvidia.com/compute/DevZone/C/html/C/src/dct8x8/doc/dct8x8.pdf

23. Performance Evaluation of Tracking and Surveillance (PETS) 2006 Benchmark Data: http://pets2006.net/

24. Podlozhnyuk, V.: FFT-based 2D convolution (2007). http://developer.download.nvidia.com/compute/cuda/2_2/sdk/website/projects/convolutionFFT2D/doc/convolutionFFT2D.pdf

25. Porikli F., Ivanov Y. and Haga T.: Robust Abandoned Object Detection Using Dual Foregrounds. EURASIP Journal on Advances in Signal Processing (2008) doi:10.1155/2008/197875

26. Ribnick, E., Atev, S., Masoud, O., Papanikolopoulos, N., & Volyes, R.: Real-Time Detection of Camera Tampering. AVSS '06 Proceedings of the IEEE International Conference on Video and Signal Based Surveillance (2006)

27. Saglam, A., & Temizel, A. Real-Time Adaptive Camera Tamper Detection for Video Surveillance. 2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance AVSS '09, pp. 430–435 (2009).

28. SanMiguel, J. C. and Martinez, J. M.: Robust unattended and stolen object detection by fusing simple algorithms. Proceedings of IEEE Int. Conf. on Advanced Video and Signal Based Surveillance, pp. 18–25 (2008).

29. Teke, M.: Satellite Image Processing on GPU Technical Report (2011). http://www.ii.metu.edu.tr/coursewebsites/quda/mteke/Satellite_Image_Proc_GPU_Paper.pdf

30. Temizel, A., Halıcı, T., Loğoğlu, B., Taşkaya Temizel, T., Ömrüuzun, F., & Karaman, E.: Experiences on Image and Video Processing with CUDA and OpenCL. GPU Computing Gems 1, NVIDIA. Elsevier (2011)

31. Vu, P., Phong, V, Vu, T. H., Le Hoai, B.: GPU Implementation of Extended Gaussian Mixture Model for Background Subtraction. International Conference on Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2010 IEEE RIVF, pp. 1–4 (2010)

32. Stava O., and Benes B., Connected Component Labeling in CUDA, Chapter in GPU Computing Gems (2010)

33. Yigit, A.: Thermal and Visible Band Image Fusion for Abandoned Object Detection, Middle East Technical University (2010)

34. Zivkovic, Z.: Improved adaptive Gaussian mixture model for background subtraction. Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004, 2, pp. 28–31 (2004)