International Journal of Scientific & Engineering Research, Volume 6, Issue 8, August -2015
ISSN **2229-5518**

GPU-based Real-Time Multiple Moving Objects Tracking using Integrated Spatial Region Graph for Video Surveillance

Samy S. A. Ghoniemy

Abstract— This paper presents the integration of a proposed enhanced multi-object color tracking, Partitioned Region Matching (PRM), and Spatial Region Graph (adjacency graph) for real time multi-object tracking. The problem of real-time object tracking is addressed by employing feature-based tracking technique that focuses on the integration of color feature tracking in regions of interest, and motion estimator which directly exploits computation of the region-level motion vectors through Partitioned Region Matching (PRM) that is based on the presence of gradients and semantically identify them according to their energy and other motion parameters. The preprocessed information are then converted to a spatial region graph (SRG) which is used as a starting point of a Markov Random Field (MRF) process, where regions are merged according to their semantics. The execution of the proposed system using GPU (NVIDIA GeForce GT 740M) showed that the processing time is enhancement by an order of 64% compared with its execution using CPU, which enabled an efficient onboard processing as well as centralized real-time processing of surveillance data, images and videos. The proposed method maps perfectly onto GPU architecture and has been implemented using NVIDIA CUDA. Experimental results on GPU for a sequence of frames, each of 460x480 pixels, showed that the implementation on GPU is 64 times faster than on CPU and confirmed the ability to process approximately 62 frames/s satisfying the necessary requirements for the correct subsequent tracking and reaching real time performance that demonstrates the suitability of the proposed system for real-time video surveillance.

Index Terms— GPU-based real-time tracking, Moving object tracking, Multi-object color tracking, Partitioned Region Matching, Remote video surveillance, Spatial region graph, GPU-CUDA.

1 Introduction

Nowadays, object tracking is a mature discipline aiming to define techniques and systems for processing videos from cameras placed in a specific environment. Tracking an object in video has a variety of real world applications; these include autonomous aerial reconnaissance, remote surveillance, and advanced real time collision avoidance systems.

Computer vision image processing algorithms were used to allow the computer to understand the contents of the image of multidimensional data and track specific color combination [1], [2], [3] to avoid the target mismatching. These research openings allowed extracting specific information from the image for a specific purpose such as controlling industry robot or an autonomous vehicle [4], [5]. Computer vision systems use digital gray-scale or color image data [6], [7], [8], [9], [10] to solve a specific task. Other computer vision systems use two or more images from a stereo camera pair, a video sequences, or a 3D volume [11], [12] to solve the same problem. Most of these computer vision systems are pre-programmed; and they are not completely suitable for real-time tracking systems. The reason for this is, because most of the computer vision systems study the tracking problem as a purely technical image processing problem depending on the software developments.

Tracking moving objects in the real time is a complex problem and many extensive studies conducted over the last few years tried to enhance these approaches. However, impressive tracking systems have been developed for some specific applications [13], [14], [15]; they showed a lack of the navigation enhancement for these moving objects and failed to integrate with a complete remotely controlled environment.

HAAR like features are image features used for object detection and recognition. It is one of the fastest and most accurate object detection algorithms. The term "HAAR-like features" origins from the calculation of Viola and Jones [16] which works with HAAR wavelet transform method, a window of the target size is moved over the input image, and for each subsection of the image the HAAR-like feature is calculated. This difference is then compared to a learned threshold that separates non-objects from objects. The discrimination between objects and non-objects is achieved by the learning phase. A set of both positive and negative images is fed to the HAAR trainer, from which the classifier is extracted. After the training phase, the classifier could be applied to a region of interest on the captured frame from the video. The classifier outputs "1" if the region contains an object and "0" otherwise. To search for an object in the whole frame, sliding window technique is applied, whereas a window (of fixed size as that used during training) is moved across the image [17].

Agrawal proposed a design, which will have a great impact on video surveillance systems, whereas a segmentation is used after background subtraction and background estimation, this was used to reduce noise and locate a moving target in a frame. For multiple moving objects detection in poor lighting conditions, wavelet-based contrast change detector was inte-

• *E-mail: samy.ghoniemy@bue.edu.eg*

grated with locally adaptive thresholding scheme for initial frames. For later frames latest change detector mechanism was used. This has significant results for surveillance systems, where the camera is mounted in a stable location, but for handheld devices [1].

The main problems in this research are the robustness of the tracking algorithm especially for fast moving objects and system integration. The gab between the capabilities of the vision system and the embedded processing system still a challenging subject, and many proposed systems in this prospective have been published, such as in [18], [19]. In these systems, motion planning is integrated with the tracking system in order to improve the overall system capabilities. The main drawbacks of these systems are their burden in the integration with the highly sensitive vision systems. Robust and fast image processing techniques, and embedded MCU based systems for robot motion control still require a noticeable enhanced navigational multi-object color tracking algorithms.

The organization of this paper starts with an introduction in Section 1. In Section 2, the theory of multi-object motion and color tracking is introduced then hybrid background removal, partitioned region matching and spatial region graph for multi object motion tracking are deeply discussed, also experimental results and analysis of this part are presented. In Section 3 a modified multi-object tracking algorithm using multi-core processor and GPU together with its experimental results are presented. Finally Section 4 includes the summary and some conclusions.

2 Multi Object Motion and Color Tracking Using Single Processor

The process of detecting objects is to analyze the information in the image, to create state measurements that enable to track moving objects. There are two methods that are useful for motion detection. The first technique is based on adaptive threshold , while the second is the color detection using hue segmentation [15]. After the detection the problem is to track and recognize these moving objects and preserve their locations even if any object stops its motion. In this section a multi-object color tracking model will be introduced, the model integrates the motion and color tracking taking the advantages of the two methods to optimize the processing time

2.1 Multi-Object Motion Tracking

The proposed multi-object motion tracking started by implementing the simplest frame difference approach with a modified dynamic (global) threshold technique in which the threshold is extracted from the histogram of the current frame to be suitable for lighting conditions changing and to overcome the limitations of previously published techniques that were based on static threshold assumption [15]. The modified algorithm is implemented and tested under different conditions to verify its robustness and its validity in detecting objects without false motion or missing motion detection.

This algorithm is further modified by introducing the novel "Sequential Connected Component Algorithm with Feedback". In this algorithm, a search is conducted for all points that represent an object using frame difference in order to detect the area in which motion takes place and then feedback is used to update previous locations. The proposed algorithm was implemented and enhanced using multithreading in which the system automatically creates a thread for each detected motion area and whose computations can then be done separately. In this proposed algorithm, the motion tracking computations are done by taking a ROI of W×H pixels around each centroid according to the experimental environment (Robot Speed, Camera Height) as given in the following equation,

$$ROI: \{W, H >= V / DT\} \quad (1)$$

where DT is the time difference between two frames.

Experiments showed that the proposed system is able to identify the locations of all objects with no false alarms or missing motion areas as shown in Fig. 1.

The time enhancement was measured during many experiments and it was found that for a few number of objects, the proposed system using multi-threading takes around 19 to 59% of the time using the traditional full frame difference technique. Fig. 2 shows the calculated processing time according to number of objects via ROI vs. the full frame processing time.

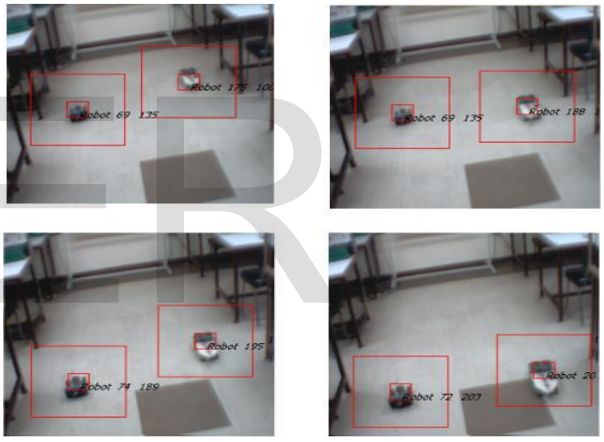

Fig. 1. Experimental results of a multi object motion tracking algorithm.

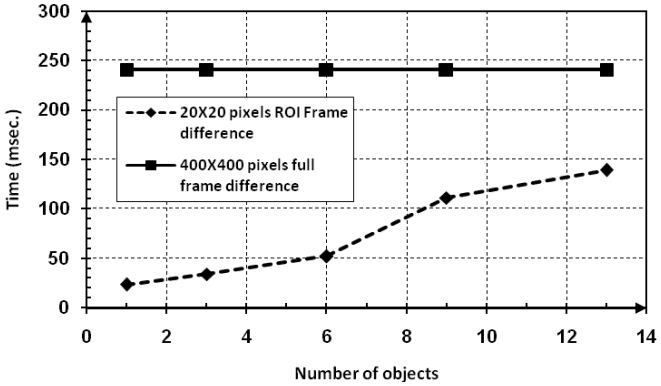

Fig. 2. Calculated processing time according to number of objects using multithreading.

Table 1 shows the computation time to track multiple objects using the proposed algorithm (Average overhead computation time is 150 msec for (Frame header + Fill Matrix + Global threshold calculation + Labeling)) and using traditional techniques for different numbers of objects. The average computation time is reduced by approximately 60%.

INTERNATIONAL JOURNAL OF SCIENTIFIC & ENGINEERING RESEARCH, VOLUME 6, ISSUE 8, AUGUST-2015
ISSN **2229-5518**

TABLE 1

COMPUTED TRACKING TIME USING THE PROPOSED ALGORITHM VS. USING THE TRADITIONAL ALGORITHM.

No. Of Objects / Algorithm	1	3	5	7	10	13
Proposed Multi-Threading Motion Tracking	29	34	41	48	52	57
Traditional Motion Tracking	308	312	320	329	338	345

The average error between the actual and the measured trajectories after applying the multi-threading motion tracking algorithm of two tracked objects at certain locations in specified frames is measured and drawn as shown in Fig. 3.

The average error is found to be less that 0.07% under different environmental conditions and the algorithm is stable towards stopped objects.

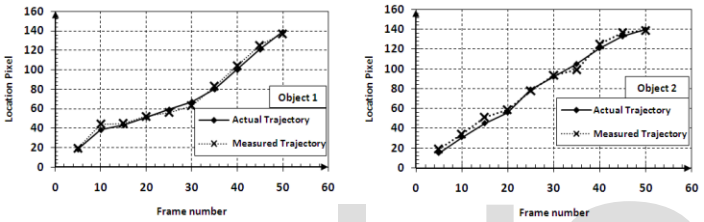

Fig. 3. Experimental results of the integrated multi object motion tracking algorithm.

The main drawback that emerged after testing many situations is that the system gets confused if the edges of two or more objects touch each other or objects areas overlap. To solve this problem a modified color tracking algorithm is proposed and discussed in Section 2.2.

2.2 Multi-Object Color Tracking

The main goal of artificial vision applications is to simulate the human vision skills such as recognizing object movements and track them in complex environments. A commonly applied technique to identify moving objects is background removal (subtraction) as it was discussed in Section 2.1. In general, background removal algorithm pass throw four steps which are pre-processing, background modeling, foreground detection and data validation [20]. For the purpose of designing an accurate background removal algorithm features such as; size (pixel, block, or cluster) and type (color, edge, stereo, motion and texture) must be chosen carefully with respect to the application and it should be able to accurately detect shapes (i.e. not affected by shadows, highlight, etc.), be reliable in different light conditions and movements of background objects, flexible to different scenarios (indoor and outdoor) and computationally efficient. On the other side, as a fact, the color of an object in video streams can differ quite significantly from what is seen using naked eye. Many environmental factors can affect the color of an object in the video exposure of the camera, such as white balance setting, ambient lighting, and reflection of the spot lights.

For these purposes, many significant research efforts had been published to address this problem such as in [21], in which Median Filter (MF) and approximated median filter (AMF) were used. The results in this publication showed that although MF is fast and simple, it needs high memory requirements, while AMF is considered a good technique only for indoor applications and it suffers from slow adaptation when there is a large change in background. Other research groups such as in [22], [23] proposed statistical models. In [22], statistical models were designed based on color and shape features where researchers in [23] used statistical models of gradients and color to classify pixels and keeps six values per pixel for gradient/color information, and region mapping. The algorithm works by grouping the foreground pixels in regions based on their color, and then checks their boundaries for foreground gradients, and keeps those whose boundary overlaps with detected foreground gradients.

As these algorithms fail to deal with complex outdoor scenes, the need for other statistical methods like W4 appeared. W4 system uses a bimodal background model keeping three values per pixel: minimum intensity, maximum intensity, and maximum intensity difference between consecutive frames [24]. Initially the background is constructed using MF and is then updated based on the change map, the major drawback of this algorithm is the high memory requirements.

A very popular technique in surveillance systems for outdoor scenes is Stauffer Mixture of Gaussian (MoG) [25], which is adaptive, online, and can handle multimodal backgrounds. MoG maintains a probability density function (PDF) for each pixel and thus has intermediate to high memory requirements.

One of Several methods to deal with multi-modal distributed background pixels is Wallflower [26], which employs a linear Wiener filter to learn and predict background variations. Although Wallflower works well for periodically changing pixel, it is less effective when background pixels change dramatically or the movements of those background pixels are less periodically.

All these algorithms faced many problems such as similar color appearing in foreground and background areas, change of lighting condition and noise. To overcome these problems and to obtain a robust segmentation algorithm, a hybrid background removal algorithm will be discussed below.

2.2.1 Hybrid background removal algorithm

In this section, a proposed hybrid background removal algorithm that preserves the accurate boundary of moving object is discussed. Fig. 4 illustrates the overall structure of the proposed algorithm which consisite of four sequential stages.

In the first stage background model is trained in RGB color space from certain number of frames; the background model ($\boldsymbol{B}_i$) is modeled by calculating the mean and standard deviation (μ, σ) for each pixel. In the second stage, the current frame (I_i) is subtracted from the background model ($\boldsymbol{B}_i$), then a predetermined threshold is used to detect the foreground binary mask (Fs_i). At the same time a single 3D Gaussian distribution is used to detect the foreground binary mask (Fg_i) and an ANDing is performed to combine FS_i and Fg_i leading to the common foreground binary mask (FC_i). Next a simple connected component method is used to get the boundaries of the region of interest (ROI) from the common foreground binary mask (FC_i) results from the previous stage then the edges of

INTERNATIONAL JOURNAL OF SCIENTIFIC & ENGINEERING RESEARCH, VOLUME 6, ISSUE 8, AUGUST-2015
ISSN **2229-5518**

the moving objects is detected using the frame difference (FD) algorithm while its area is filled using simple line segmentation method and shadowes are removed. The last stage is to validate region pixels using *Fs*, *Fg* and the output from line segmentation, then the background model ($\boldsymbol{B_i}$) is updated recursively using Gaussian running average filter.

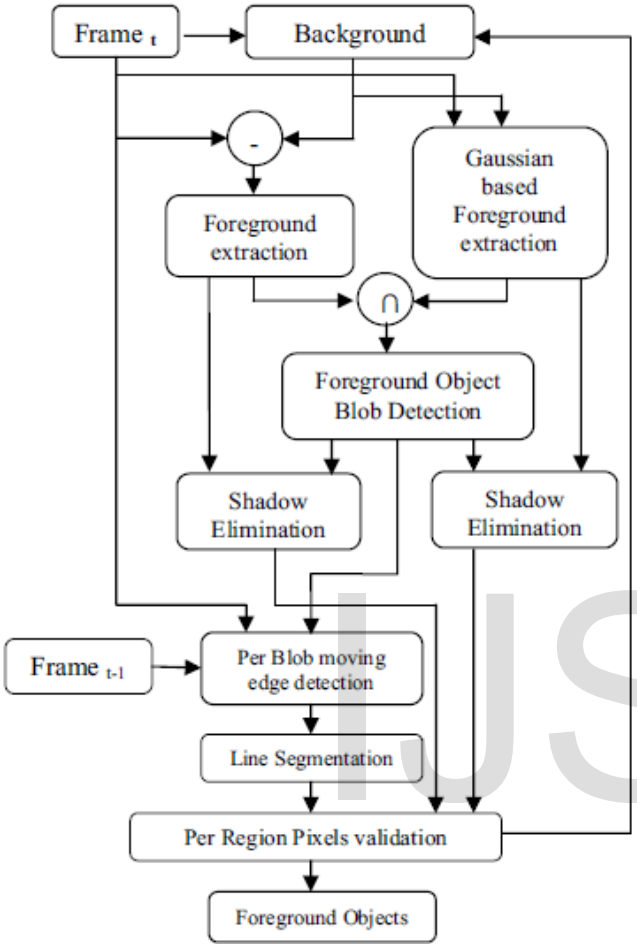

Fig. 4. Block diagram of the proposed adaptive background removal algorithm.

The proposed technique was evaluated using the Wallflower sequences, 120x160 color images [27] and compared to some commonly used detection techniques. To evaluate the performance against each sequence, False Positive (FP), False Negative (FN), and total error (TE) are used. FP is the number of wrongly marked background pixels as foreground; FN is the number of wrongly marked foreground pixels as background; TE is the sum of FP and FN for each sequence. For the overall performance evaluation, TE_T is the sum of total error for all image sequences. Fig. 5 shows the results obtained by using the proposed technique compared with several other state-of-the-art methods. From Fig. 5, it is concluded that the proposed technique achieves the most accurate overall performance of total error (TE) for image sequence of moved object among the competitive methods even though there is no elimination for the foreground pixels whose 4-connected foreground pixels number less than 8 as other methods did.

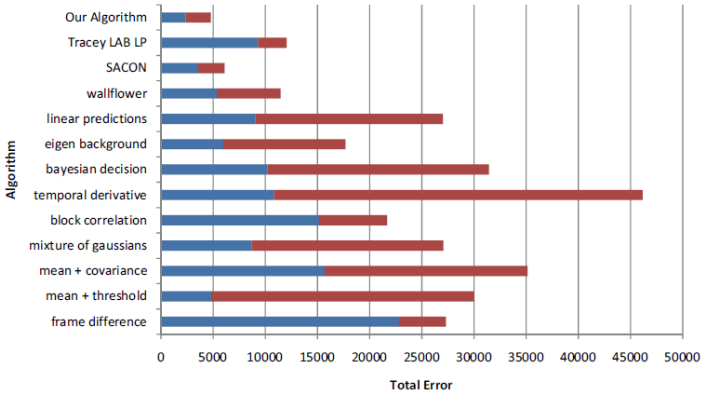

Fig. 5. Overall performance obtained by using the proposed technique compared with several other state-of-the-art methods.

2.2.2 Enhanced hybrid background removal algorithm

In practice, the performance of this technique is largely degraded because the color of an object in video streams can differ quite significantly from what is seen using naked eye. Practically, many environmental factors can affect the color of an object in the video exposure of the camera, such as white balance setting, ambient lighting, and reflection of the spot lights. Theoretically, an object of red color should have the RGB value of (255, 0, 0), in practice, one can never find such value although the object is really red in color when seeing it with naked eyes. Testing the RGB values produced by the camera attached to server showed that they do not give enough information to identify all objects in the region of interest. So, it is more reasonable to convert the representation of colors into suitable color space in order to catch all color specifications. Many color spaces like CMY, HSV, XYZ, and YCbCr are analyzed, from them the HSV gave the best results and found to be closer to human perception. The HSV color space provides us with a component containing the color wavelengths called Hue (H) which defines the set of the color, Saturation (S) which defines how pure the color will be, and Value (V) which defines the brightness of the color as shown in Fig. 6.

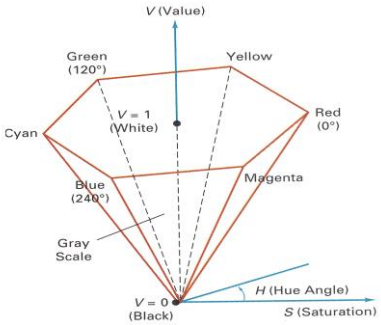

Fig. 6. HSV Color Model.

In this model, segmentation by hue is achieved by setting all pixels that have a hue in a specific range around the hue of the background to belong to the background. Pixels with a different hue are set to be objects. The range of hues which will be classified as background is selected by examination of the hue component's histogram, where the background will have a distinct peak. The main advantage of this method is that effects of lighting and shadows can be reduced since the intensity and saturation information is separated from the hue

INTERNATIONAL JOURNAL OF SCIENTIFIC & ENGINEERING RESEARCH, VOLUME 6, ISSUE 8, AUGUST-2015
ISSN **2229-5518**

component in the HSV color space.

Testing the traditional multi-object color tracking technique showed that there exist two main problems. The first problem is that it requires a long processing time for full frame color transformation which is not suitable for real time applications. Therefore, more modifications are proposed to minimize the processing time in which color transformation is applied only for regions around the extracted centroids of targets.

Fig. 7 illustrates the flowchart of the integrated motion and modified color tracking algorithms after applying the color space transformation for the regions of interests.

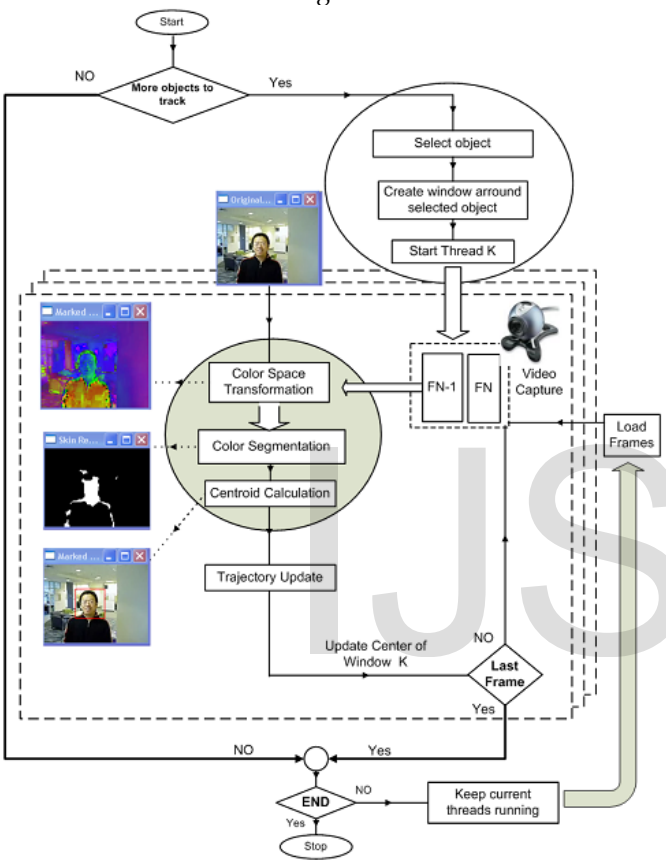

Fig. 7. Flowchart of enhanced motion and color tracking algorithms.

In this model, segmentation by hue is achieved by setting all pixels that have a hue in a specific range around the hue of the background to belong to the background. Pixels with a different hue are set to be objects. The range of hues which will be classified as background is selected by examination of the hue component's histogram, where the background will have a distinct peak. The main advantage of this method is that effects of lighting and shadows can be reduced since the intensity and saturation information is separated from the hue component in the HSV color space. A disadvantage of this model is that it is usable only when there are enough colors in the image.

The proposed model enhancement allows color transformation of areas corresponding to theoretically more than 50 objects/frame at a frame size of 400×400 pixels while the traditional algorithm just makes color transformation of the given frame. Since the average number of objects in the frame is 8 objects to keep the details of objects, then many tasks can be done during the saved time.

Fig. 8 shows the theoretical number of objects at different times at frame of 400x x400 pixels.

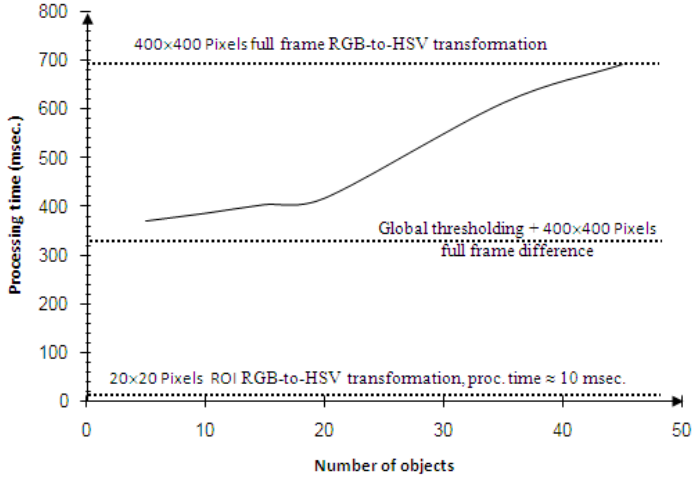

Figure 8: Procesing time enhancement using multi-threading techniques.

The second problem appeared when more than one color exist and more than one object have the same color leading to different areas with the same label after color segmentation.

The labeling here is therefore nested in order to cover all objects of different colors. For each color i, assign label L_i, scan full frame, and the objects of that color are labeled L_{i0}, L_{i1} … L_{iN}. So, the more the colors, the more processing time needed. The proposed model implements one level labeling which reduces the time needed to that of a single full frame scan by integrating motion tracking with color tracking.

Fig. 9 presents some of the experimental results using the modified multi-object color tracking algorithm. This figure shows that no false alarms or missing objects during the experiment for 120 frames.

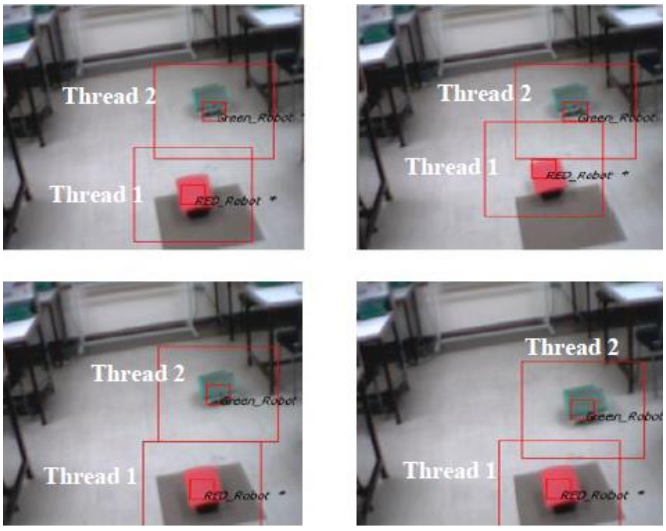

Fig. 9. Experimental results of the enhanced integrated multi object motion and color tracking algorithm.

The average time enhancement using the multi-threading technique after many experimental measures was found to be 54% of the traditional multi-object color tracking when the

number of objects is less than 15. Table 2 compares the processing time for different cases using multi-threading and traditional color tracking algorithms.

TABLE 2
PROCESSING TIME USING MULTI-THREADING COMPARED WITH TRADITIONAL MULTIOBJECT COLOR TRACKING TECHNIQUES.

No. Of Objects / Algorithm	1	3	5	7	10	13
Multi-Threading Color Tracking	339	353	369	385	409	430
Traditional Color Tracking	693	710	726	736	747	762

Testing this technique showed that the required processing time and memory requirements are nonlinearly increasing functions with respect to the number of objects and even gets worth in case of different object shapes/geometries such as in tracking multiple persons. This indeed not suitable for real time applications such as in case of survelance applications. Therefore, more modifications will be proposed in the following section to minimize the processing time and memory utilization for real time survelance applications.

2.3 Partitioned Region Matching and Spatial Region Graph (SRG) for Multi Object Motion Tracking

In this proposed enhanced approach each frame of the sequence is segmented into regions by using color with the assumption that each region contains only one object as discussed in Section 2.2. In this section this assumption is more modified by assuming that each region may contain a part of one object. This algorithm consists of two major phases.

Phase 1- color-based region segmentation and SRG creation, for this task the following tasks are performed:

1) Color segmentation is implemented based on the region growing algorithm using the proposed algorithm in Section 2.2.2, then;
2) A Spatial Region Graph (SRG) is created, in which nodes with attributes (area size, the coordinate of the bounding-box and the centroid's position) represent regions, whereas arcs (edges) represent topological adjacencies (predicates).

Phase 2- object motion estimation and regions merging, for this task the following tasks are performed:

1. With the assumption that camera and objects are slowly moving, the motion estimation algorithm computed with the adoption of the translational motion model in which the region-level motion vectors (velocity and reliability) are directly computed for each region using Partitioned Region Matching that is based on classical Sum of Absolute Differences (SAD) over each region, and are then stored in the Spatial Region Graph (SRG) as an update, then;
2. Regions with similar motion are merged according with an energy function influenced by the motion vectors using Markov Random Field (MRF) process [28]. As proposed in [28], the energy function is composed by three terms: the first term (U1) represents the motion, the second term (U2) represents the geometrical regularization, and the last term (U3) represents the number of broken arcs (pairs of regions geometrically adjacent but with different labels). This energy function is given by; U(e, o) = U1(e, o) + U2(e) + U3(e), where e and o are the labels and the observation fields, respectively, then;
3. Background and objects identification starts. In this algorithm, the largest region after merging similar regions that exhibit similar motion (i.e., the Euclidean distance in the velocity space must be under a suitable threshold) is classified as background and its motion as the motion of the camera, while other regions are classified as moving objects and temporal continuity of motion field is supposed to perform a prediction of the next frame.
4. The final step is the prediction step, which is used to restart the region-level MRF process for the analysis of the next frame.
5. Repeat.

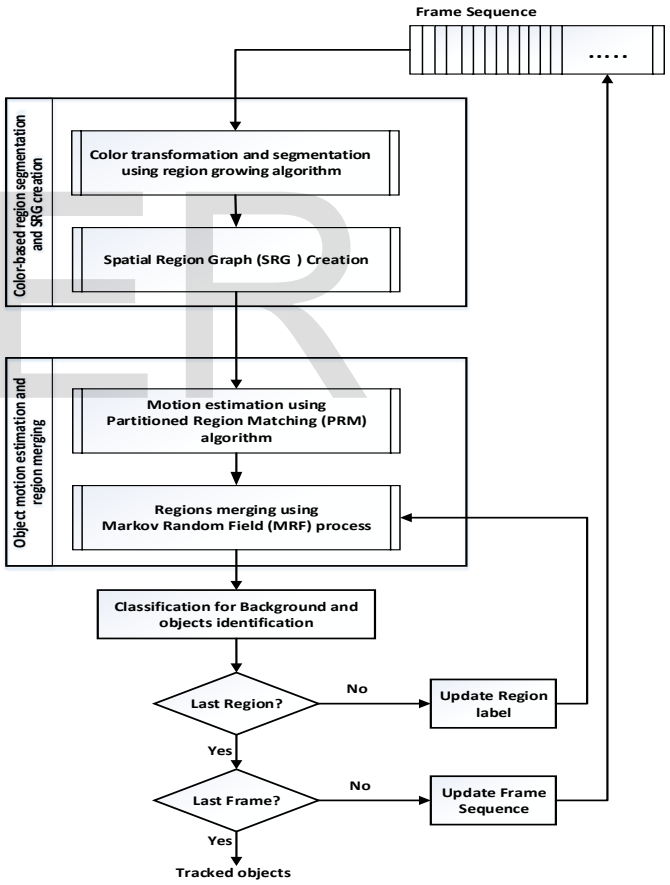

Fig. 10. Flowchart of multi object motion tracking using partitioned region matching (PRM) and spatial region graph (SRG).

2.3.1 Experimental results

In order to study the efficiency of the above algorithms, the proposed system has been implemented and evaluated on the Wallflower dataset sequences [26]. The video sequences composed of 50 frames, these sequences are:

Moved Object: A person enters into a conference room, makes a telephone call, and leaves. The background is evaluated using 50

INTERNATIONAL JOURNAL OF SCIENTIFIC & ENGINEERING RESEARCH, VOLUME 6, ISSUE 8, AUGUST-2015
ISSN **2229-5518**

frames after the person leaves the scene.

Time of Day: A darkened room that gradually becomes brighter over several minutes. A person enters into the room and sits down.

Light Switch: First as a training stage the camera sees a room with the lights both ON and OFF for long period. Then the room starts with the lights OFF. After a few minutes a person enters the room turns ON the light, and moves a chair. The foreground objects are both the person and the moved chair is considered foreground.

Waving Trees: A swaying tree and a person walks in front of it.

Camouflage: A rolling interference monitor with bars sits on a desk. A person walks into the room in front of the monitor.

Bootstrapping: The sequence shows several minutes of an overhead view of a busy cafeteria, where every frame contains people.

Foreground Aperture: A back view of a sleeping person, with a uniformly colored shirt, at his desk, viewed from the back. He slowly wakes up and begins to move.

To evaluate the performance against each sequence, FP, FN, and TE are used. For the overall performance evaluation, TE_T, the sum of total error for all image sequences is used. The proposed evaluation tests are; (1) ground-truth tests (i.e. tests with manually segmented frames as ground-truth) have been performed to evaluate the efficacy, (2) a time analysis test has been carried out to study the capability of the proposed system to satisfy real-time requirements. The proposed system is executed on a Dell Latitude E5530 laptop with a 2.5 GHz i5 processor, 8 GB RAM, and Windows 7 64-bit.

1- Performance evaluation

The FP, FN, and TE errors obtained for the foreground aperture and waving trees sequences using the proposed system and other 12 published systems are presented in Figures 11-13.

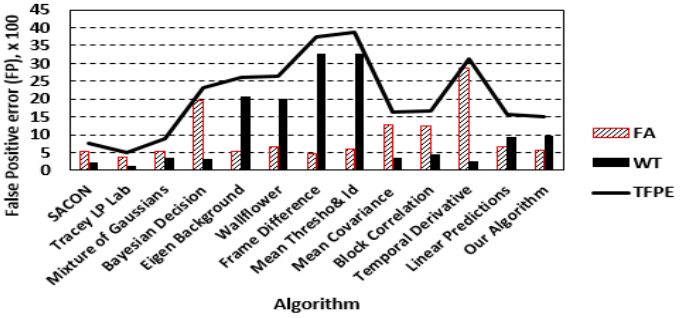

Fig. 11. False Positive (FP) error points of still objects or of the background wrongly marked as belonging to moving objects.

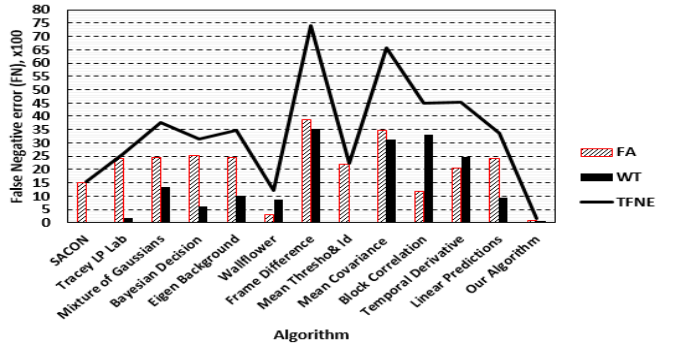

Fig. 12. False Negative (FN) error points or moving objects' points that are not detected or wrongly marked as belonging to background.

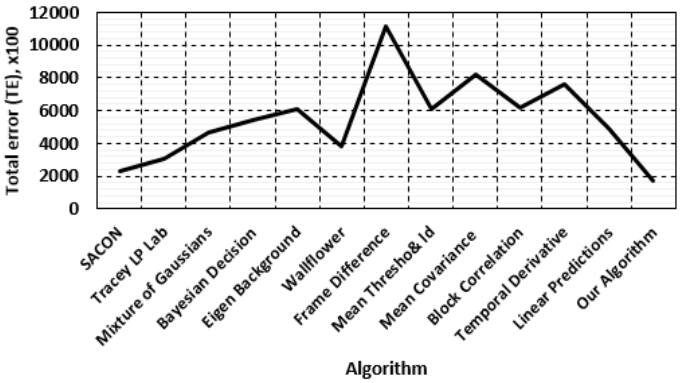

Fig. 13. Total error (TE), the sum of total error for all image sequences.

Taking all together, Figures 11-13 show that the proposed system gives satisfactory results in terms of accuracy of segmentation and motion detection.

2- Computational performance evaluation

Computational time analysis has been carried out using different frame sizes such as 100x100, 120x160, 200x200, 400x400, 640x480, 900x450, and 1024x768 to present a detailed time analysis that shows the cost of each phase and their possible dependencies on application parameters. Fig. 14 illustrates the computational time of color segmentation, spatial region graph (SRG), object motion estimation using PRM and regions with similar motion merging using broken arcs-based MRF process for one 200x200 frame versus different number of regions at maximum region shift of 20 pixels.

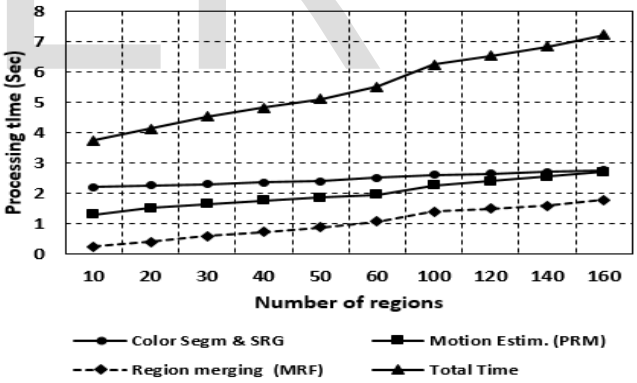

Fig. 14. Computational time for one 200x200 frame versus different number of regions at maximum region shift of 20 pixels.

Taking all together, Fig. 11–to- Fig. 14, it is concluded that the necessary computational time for the color segmentation and SRG phase approximately does not depend on the number of regions, instead largely depends on the frame size, taking into account that SRG processing time is affected only by the number of nodes representing the adjacent regions and their connecting arcs. It is also concluded that the computational time for motion estimation depends not only on the frame size but also on the number of regions and the size of region shift. Finally, the processing time required for MRF increases nonlinearly with the number of regions which in turn affects the PRM especially for larger size of region shift.

INTERNATIONAL JOURNAL OF SCIENTIFIC & ENGINEERING RESEARCH, VOLUME 6, ISSUE 8, AUGUST-2015
ISSN **2229-5518**

3 MULTI OBJECT TRACKING USING MULTI-CORE PROCESSOR AND GPU

When looking at the execution time of the whole algorithms proosed in Section 2, it was concluded that the processing time of the color segmentation, SRG creation and motion estimation is largely depends on the frame size, the number of regions and the size of region shift. To reduce the number of computations required, all operations are restricted to a region-of-interest (ROI) or window. Yet, region growing still the most consuming processing power. Therefore it will be reasonable to implement this feature on the graphics processing unit (GPU) since the programmable GPU is specialized for highly parallel computations, multi-threaded, and many-core processor with tremendous computational processing power and very high memory bandwidth.

Recently, many parallel programming interfaces have been proposed, among them, the compute unified device architecture (CUDA) [29], [30] has become one of the most practical parallel computing platform and programming model invented by NVIDIA [31], [32], [33]. CUDA enables dramatic increases in computing performance by harnessing the power of the GPU, utilizing two other types of memory access, shared memory and global memory, on top of the texture cache units as well as the utilization of its execution model which involves kernels, threads, blocks and grids.

From the programmer's point of view, every program consists of two parts, a host code (called host) run on CPU, and a kernel, a piece of code (called device) run in parallel on GPU. CUDA's extensions to the C programming language allow programmer to define how many threads performing a kernel will be executed on a device. The number of threads can greatly (up to 10,000×) exceed the number of processing units which execute the program. This paradigm allows programmers to consider GPU as an abstract device providing a huge number of virtual resources. Therefore, not only it provides a good scalability, it also allows to hide a global memory latency by very fast thread switching. The threads executed on GPU form a 1D, 2-D, or 3-D structure, called a block. The blocks are arranged in 1D or 2-D layout, called a grid. Thus, each thread can be exclusively defined by its coordinates within a block, and by coordinates of its block within a grid. Each block of threads is executed on a single streaming multiprocessors (SMs); therefore, all threads of a single block can share data in a shared memory. Block's threads are executed in warps of 32 threads each, as shown in Fig. 15 [34]. A memory hierarchy is introduced in a number of levels as shown in Fig. 16. In the lowest level, there are small, fast, non-shared registers. Then, each SM contains a shared memory. CUDA threads may access data from multiple memory spaces during their execution as illustrated by Fig. 17.

The proposed implementation uses a 2D grid although it is based on a 1D grid. The conversion is carried out by each kernel. The kernel's job is to scan for the features on a single ROI, keeping in mind that the algorithm yields different window sizes (i.e. scales) which requires pre-scaled features and window information such as the x and y position of the window and the window size. The features set consists of different stages cascaded together. Each window will pass through all stages till the end.

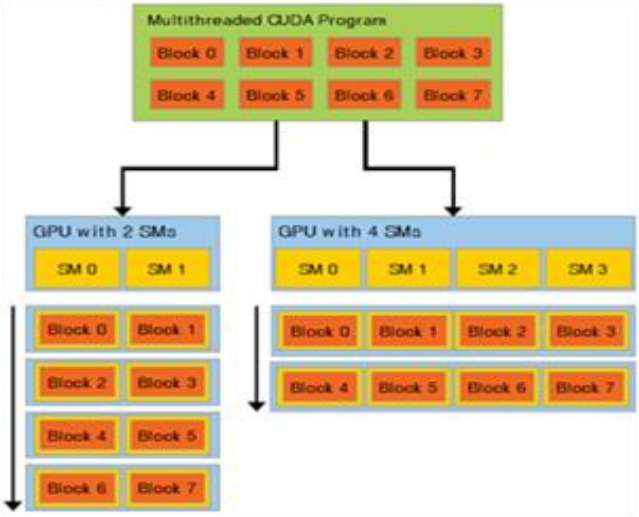

Fig. 15. GPU Automatic Scalability [34].

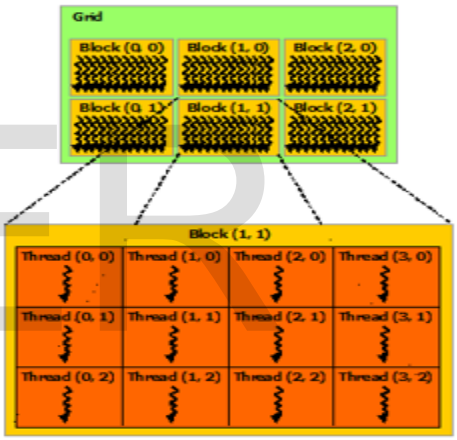

Fig. 16. Threads hierarchy in CUDA programming model [34].

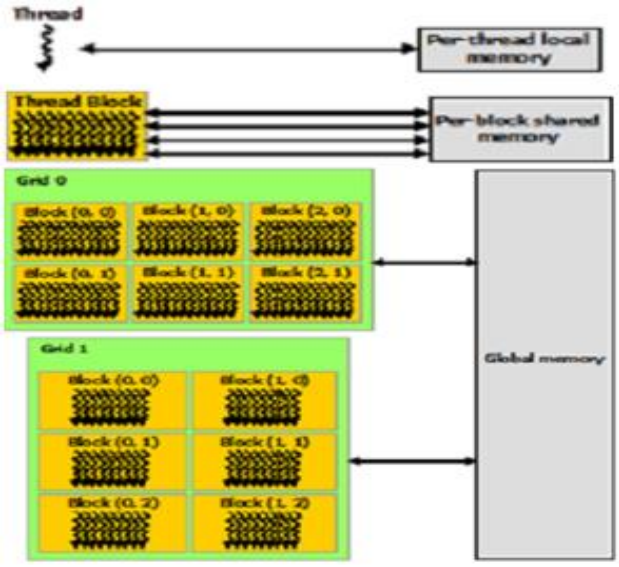

Fig. 17. Memory hierarchy in CUDA programming model [34].

The features set consists of different stages cascaded together. Each window will pass through all stages till the end. This set of features is done sequentially, meaning that each kernel will not need to scale the features nor scale the image itself, in other words the kernel's job is to evaluate features and stages for its specific window. To avoid memory latency and maximum memory bandwidth, the data is sorted according to the scale following by the **X** and **Y** position of each window [35]. The number of kernels is equal to the number of search windows which varies according to the image size used. Fig. 18 shows the number of search windows vs the image size, it is noted that the number of search windows increases in a cubic manner, which helps in evaluating the proposed system.

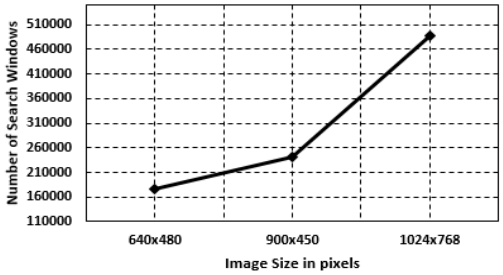

Fig. 18. Number of Search Windows vs. Image Size (pixels).

Since GPUs are based on PCI Express (PCIe), additional overhead costs are resulted from host-to-GPU data transfers and vice versa, which may cause application bottleneck at data transfer. Many researchers such as in [36] proved that this process consumes time and processing power, which derived researchers to search for other approachs to minimize the size of data allocated and shared from the host to each working item. For proper memory utilization and decreasing non-necessary data allocation, all criteria and data passed needs to be studied to determine the dependency and data size. These parameters are the original image, integral image, the scaled feature set, the window upper left and lower right corner (X and Y) positions, and window step size. Some of these parameters are indispensable such as the original and integral images, and the scaled feature. The rest of the parameters could be substituted by equations. Therefore, the goal is to find a model that includes scale, step size, window upper left and lower right corners (X and Y positions), a modified window ID, number of windows per row, number of rows and total number of windows then relate them to the Kernel ID.

3.1 Scale factor prediction model

In order to find an equation that satisfies the scale values, scatter plot as a diagnostic test to curve fitting is applied first. This helps to identify the model or set of models that would provide the least error (highest R Squared value). After testing 39 models it was found that the Ratio of Polynomials Fit model given by (2) and illustrated in Fig. 19 leaded to the highest least error values, $R^2 \approx 0.996$.

$$Scale = \left[\frac{1.08 - (9.24E-06) \times ID + (1.79E-11) \times ID^2}{1 - (1.05E-05) \times ID + (2.74E-11) \times ID^2} \right] \quad (2)$$

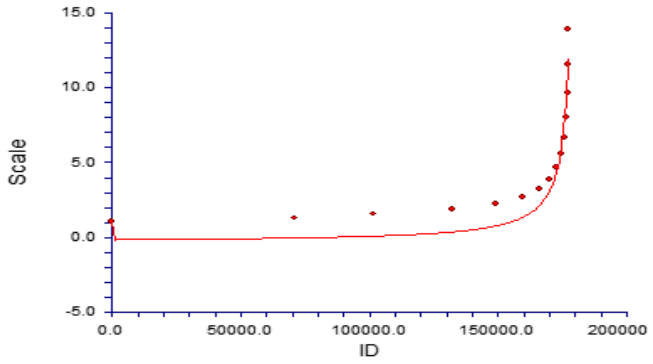

Fig. 19. PolyRatio (3, 3) Model for Scale Value estimation.

3.2 Optimization of the GPU code

Consulting the scatter plot, it is noted that the relation between the step value and the scale is a simple linear relation since it leaded to the highest square value of ≈ 0.999562. The main drawback in this model is its linear dependency on the step size, since as the scale increases the step size increases accordingly; which means higher possibility of undetected objects. For this purpose, a modified window model is proposed as follow.

The modification started by introducing two variables to evaluate both the ***X*** and *Y* start position of the window in the original image. After testing 39 models, it was found that no model will satisfy the whole data set, so the data set was divided into smaller data sets according to the scale. Referring to Fig. 19, it is noted that the scale remains constant for a period of time and number of windows per scale (W_{PS}) is given by (3),

$$W_{PS} = W_{PR} * NumOfRows \quad (3)$$

in which, W_{PR} represents the number of windows per each row and *NumOfRows* represents the number of rows in each image, and they are given by (4) and (5), respectively.

$$W_{PR} = I_{Width} - \frac{(W_{Width} * Scale)}{Step} \quad (4)$$

$$NumOfRows = I_{Height} - \frac{(W_{Width} * Scale)}{Step} \quad (5)$$

Where $\boldsymbol{W}_{Width}$ represents the window size, $\boldsymbol{I}_{Width}$ and $\boldsymbol{I}_{Height}$ represent the sample image width and height, respectively.

The modified window ID would be incremental for every sub-data set and is given by (6),

$$WID_{modified} = W_{range} + WID \quad (6)$$

where *WID* represents the Kernel ID currently running.

By applying this model we noticed that it gives extremely bad results on GPU, since branching was applied to detect the range for the data set, these conditions froze other working items due to the Single Instruction Multiple Data (SIMD) architecture of the GPU. This means that not all methods are suitable for a massive parallel processing because the GPU performance is limited by several conditions: a) the ratio of parallel to sequential fraction of the algorithm, b) the ratio of floating point operations to global memory accesses, c)

branching diversity, d) global synchronization requirements and e) data transfer overhead [37], [38].

Thus, the most desired goal is to split data into independent parts so that they can be processed in parallel. When a multiprocessor is given one or more thread blocks to execute, it partitions them into warps and each warp gets scheduled by a *warp scheduler* for execution. The way a block is partitioned into warps is always the same; each warp contains threads of consecutive, increasing thread IDs with the first warp containing thread 0. A warp executes one common instruction at a time, so full efficiency is realized when all 32 threads of a warp agree on their execution path. If threads of a warp diverge via a data-dependent conditional branch, the warp serially executes each branch path taken, disabling threads that are not on that path, and when all paths complete, the threads converge back to the same execution path. Branch divergence occurs only within a warp; different warps execute independently regardless of whether they are executing common or disjoint code paths [34]. The number of blocks and warps that can reside and be processed together on the multiprocessor for a given kernel depends on the amount of registers and shared memory used by the kernel and the amount of registers and shared memory available on the multiprocessor. There are also a maximum number of resident blocks and a maximum number of resident warps per multiprocessor. These limits as well the amount of registers and shared memory available on the multiprocessor are a function of the compute capability of the device.

Therefore, to enable the number of blocks and warps to be processed together on the multiprocessor for a given kernel, branch divergence is adopted and minimization of the branch condition true hit is applied as follow.

The proposed enhanced model starts by calculating the parameter ***y*** which indicates whether the current kernel is within the data set range or not, and is given by (7),

$$y = \frac{WID \ - RangeStart}{|WID - RangeStart|} = \begin{cases} -1 & \text{if } WID < \text{the range} \\ 0 & \text{if } WID = \text{the range} \\ 1 & \text{if } WID > \text{the range} \end{cases} \quad (7)$$

where ***RangeStart*** values represents the data set window ID range. Equation (7) is used to calculate the predictor, **z,** value and is given by (8),

$$z = f(y) = \left\lfloor \frac{y+1}{2} \right\rfloor = \begin{cases} 0 & if \ y < 0 \\ 1 & if \ y \geq 0 \end{cases} \quad (8)$$

The modified window ID could be estimated using (9),

$$WID_{Modified} = (WID - range) \times z \quad (9)$$

To calculate both **X** and **Y** positions it necessary to use the modified window ID, hence the **X** and **Y** positions will also be within a sub-data sets, this is done using (10), (12), respectively.

$$X = \left\| \left(\left| WID_{Modified} \right|, W_{PR} \right) \right| \times Step, i_{width} \quad (10)$$

where i_{Width} is given by (11),

$$i_{Width} = I_{Width} - (W_{Width} * Scale) \quad (11)$$

The equation for **Y** position calculations is also presented in a single formula given by (12),

$$Y = \left\lfloor \left(\frac{WID_{Modified}}{W_{PR}} \right) \times Step \right\rfloor \quad (12)$$

3.3 Experimental results

The algorithms presented in Section 2 have been implemented by following the described method in Section 3, and the experiments were conducted using the same image set on a Laptop HP Pavilion 15-N043se (Intel Core i7-4500U) with 4M Cache @1.8 GHz, 2.4 GHz, processor bus speed is 5 GT/s, 8 GB RAM, display card NVIDIA GeForce GT 740M (2 GB GDDR3 dedicated), and Windows 7 64-bit. The computational time analysis has been carried out using different frame sizes, namely 100x100, 120x160, 200x200, 400x400, 640x480, 900x450, and 1024x768, to present a detailed time analysis that shows the cost of each phase and their possible dependencies on application parameters. Fig. 20 illustrates the computational time of color segmentation, spatial region graph (SRG), object motion estimation using PRM and region with similar motion merging using broken arcs-based MRF process for one 640x480 frame versus different number of regions at dynamic number of regions and region shift in pixels. The illustrated results in Fig. 20 are calculated by negating the kernel compilation time, which only runs once on application startup.

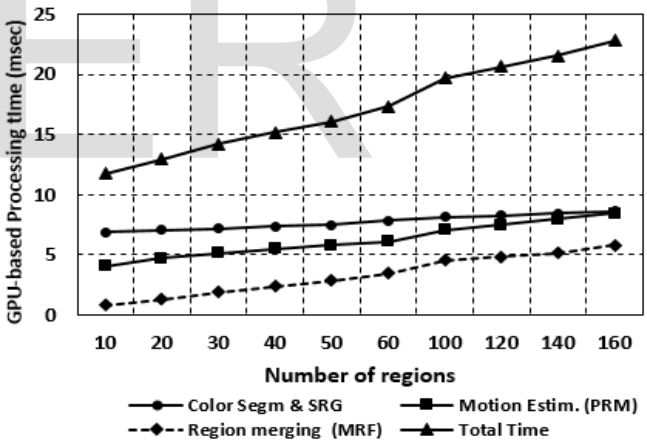

Fig. 20. GPU-based computational time for one 640x480 frame versus different number of regions at dynamic region shift.

This figure shows that using GPU and optimizing the implementation process have dramatically enhanced the accuracy, the computation time (the total processing time for 160 regions is reduced from 1.46 sec to 22.81 ms which means the processing time is enhancement by an order of 64% compared with its execution using CPU), and memory allocation.

Another metric usually employed to evaluate the computing capability of a real-time oriented system is the processing rate or rate. Rate measures how many frames are processed per second and is given by (13) [39].

$$Rate \ (\text{FPS}) = \frac{1000 \ (\text{ms})}{t_{total} \ (\text{ms})} \ (\text{FPS}) \quad (13)$$

in which t_{total} is the total processing time of all stages for a single frame.

Using GPU optimized implementation and applying (13) the computed processing rate for 640x480 frames versus different number of regions is illustrated in Fig. 21. When applying (13), t_{total} is substituted by the summation of processing time of color segmentation, SRG, object motion estimation, and region merging.

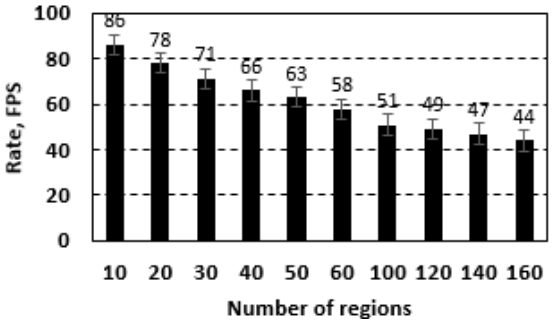

Fig. 21. GPU-based processing rate for 640x480 frames versus different number of region.

Fig. 21 shows that using GPU optimized implementation and applying (13), the processing rate varies from 86 to 44 frames per second when the number of regions varies from 10 to 160, respectively and the average rate is 62 frames per sec satisfying the necessary requirements for the correct subsequent tracking stage (a minimum rate between 8 and 15 fps) and reaching real time performance for surveillance applications.

4 Conclusion

This paper proposed a multi-object visual color tracking algorithm using multi-threading in real-time. Experimental results show that the proposed integrated system is stable towards stopped objects (using a constructed memory module), and partially occluded objects can still be detected, recognized and tracked using an added color feature. These results also show that the proposed system supports overlapped regions of interest (ROIs) and reduced the average computation time by approximately 54% when multithreading is applied. This proposal was further enhanced by including a hybrid background removal technique. The proposed enhancement is applied to different practical environments and proved to be effective in detecting and validating the foreground objects even under sudden illumination. The modified system is evaluated with Wallflower benchmarks and the experimental results are compared with popular background removal techniques and showed that the system offers comparable, better detection accuracy and proved to be robust with different background scenes.

The problem of real-time object tracking is also addressed by employing feature-based tracking technique that focuses on the integration of color feature tracking in regions of interest, spatial region graph, and motion estimator which directly exploits computation of the region-level motion vectors through partitioned region matching and Markov random field process, where regions are merged. The proposed method maps perfectly onto GPU architecture and has been implemented using NVIDIA CUDA. Experimental results on GPU for a sequence of frames, each of 460x480 pixels, showed that the implementation on GPU leaded to the total processing time for 160 regions is reduced from 1.46 sec to 22.81 ms which means the processing time is 64 times faster than on CPU and the processing rate varies from 86 to 44 frames per second when the number of regions varies from 10 to 160, respectively and the average rate is 62 frames per sec satisfying the necessary requirements for the correct subsequent tracking stage (a minimum rate between 8 and 15 fps) and reaching real time performance. These results demonstrate the suitability of the proposed algorithm for real-time video surveillance.

References

[1] D. Agrawal and N. Meena, "Performance Comparison of Moving Object Detection Techniques in Video Surveillance System," The International Journal of Engineering And …, pp. 240–242, 2013.

[2] E. Weng, R. Khan, S. Adruce, and O. Bee, "Objects Tracking from Natural Features in Mobile Augmented Reality," Procedia-Social and …, vol. 97, pp. 753–760, Nov. 2013.

[3] Ramshetty K Sure and Savitha Patil ," Android Based Autonomous Coloured Line Follower Robot", International Journel of Research and Technology, Vol. 3, May 2014.

[4] Z. Macarthur, "Compliant Formation Control of an Autonomous Multiple Vehicle System," PhD thesis, University Of Florida, May 2006.

[5] L. Edwards, Open Source Robotics and Process Control Cookbook, 2nd Ed., Printce Hall, April 2005.

[6] G. D. Hager and K. Toyama, The XVision system: A General Purpose Substrate for Real-Time Vision Applications,' Computer Vision and Image Understanding, vol. 69, pp. 23-27, January 1998.

[7] C. Wren, A. Azerbaijani, T. Darrell, and A. Pentland, "Real-Time tracking of the human body," IEEE Trans. on PAMI, vol. 19, no. 7, pp. 780-785, 1997.

[8] S. Asaad, A low-cost, real-time, intelligent vision sensor, Master's thesis, Vanderbilt University, Electrical Eng., 1995.

[9] M. Borg et al., "Evaluation of object tracking for aircraft activity surveillance," (Beijin), Oct. 2005.

[10] J. Shi and C. Tomasi, "Good Features to Track," IEEE Conf. on Computer Vision and pattern Recognition, pp. 593-600, 1994.

[11] Y. Bar-Shalom and X. Li, "Multitarget-Multisensor Tracking: Principles and Techniques," YBS Publishing, 1995.

[12] J. Black and T. Ellis, "Multi Camera Measurement and Correspondence," J. of the International Meas. Confederation, vol. 35, no. 1, pp. 61-71.

[13] Y. Cai, N. Freitas, and J. Little, Robust visual tracking for multiple targets, Master's thesis, University of British Columbia, September 2005.

[14] M. Swain and M. Stricker, "Promising directions in active vision," Int. J. of Comp. Vision, vol. 11, no. 2, pp. 109-126, 1993.

[15] Tarek Said, Samy Ghoniemy and Omar Karam, "Real-Time Multi-object Detection and Tracking for Autonomous Robots in Uncontrolled Environments," *IEEE 7th International Conference on computer engineering and systems (ICCES 2012), Cairo, Egypt, November* 2012.

[16] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, vol. 1, 2001.

[17] O. Community, "OpenCV Reference Manual," October, pp. 1–1104, 2010.

[18] Freksa, R. Moratz, and T. Barkowsky, "Navigation with schematic maps," Intelligent Autonomous Systems, University of Hamburg, April 2005.

[19] J. Aguilera and et al., "Visual surveillance for airport monitoring applications," 11th Comp. Vision Winter Workshop, 2006.

[20] Wessam Elhefnawy, Gamal Selim, Samy Ghoniemy, "Robust Hybrid Foreground Detection Adaptive Background," IEEE Int. Conf. on information science and applications (ICISA 2010), April 21-23, 2010, Seoul, Korea.
[21] M. Piccardi, "Background subtraction techniques: a review," IEEE Conf. on Systems, Man and Cybernetics, vol. 4, pp. 3099-3104, October 2004.
[22] C. Wren, A. Azrbayejani, T. Darrell, A.P. Pentland, "Pfinder: real-time tracking of the human body," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 19, no.7, pp. 780-785, 1997.
[23] M. Shah and O. Javed, K. Shafique, "Automated visual surveillance in realistic scenarios". IEEE Multimedia, vol. 14, no. 1, pp. 30-39, 2007.
[24] I. Haritaoglu, D. Harwood, L.S. Davis, "W4: real-time surveillance of people and their activities," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 22, no. 8, pp.809–830, August 2000.
[25] C. Stauffer and W.E. Grimson, "Adaptive background mixture models for real time tracking," IEEE Proc. CVPR, pp.246-252, June 1999.
[26] K. Toyama, J. Krumm, B. Brumitt, B. Meyers, "Wallflower: principles and practice of background maintenance," IEEE Conf. Computer Vision, vol. 1, pp. 255-261, 1999.
[27] K. Toyama, J. Krumm, B. Brumitt, B. Meyers, "Wallflower: principles and practice of background maintenance," IEEE Conf. Computer Vision, vol. 1, pp. 255-261, 1999
[28] M. Gelgon and P. Bouthemy, "A region-level motion-based graph representation and labeling for tracking a spatial image partition," Pattern Recognition, vol. 33, pp. 725–740, 2000.
[29] David B. Kirk and Wen-mei W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*, Elsevier Science, pp. 14-16, 2012.
[30] Luebke, D., "CUDA: Scalable parallel programming for high-performance scientific computing," *Biomedical Imaging: From Nano to Macro, 2008. ISBI 2008. 5th IEEE International Symposium on*, pp.836-838, 14-17 May 2008.
[31] CUDA zone, *nVidia,* URL: *http://www.nvidia.com/object/cuda_home.html*, last visited, August 2015.
[32] NVIDIA CUDA Programming Guide, August 2015. http://developer.download.nvidia.com
[33] J. Sanders and E. Kandrot , "CUDA by example: an introduction to general-purpose GPU programming," Addison-Wesley Professional, 2010.
[34] CUDA Programming Guide v7.0, March 2015, http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf
[35] R. Uribe-Paredes et al., "Similarity search implementations for multi-core and many-core processors," High Performance Computing and Simulation (HPCS), 2011 International Conference on, pp. 656-663, 2011.
[36] M. Daga, A. M. Aji, and W. Feng, "On the Efficacy of a Fused CPU+GPU Processor (or APU) for Parallel Computing," 2011 Symp. Appl. Accel. High- Performance Compute. pp. 141–149, 2011.
[37] In Kyu Park, Nitin Singhal, Man Hee Lee, Sungdae Cho, and Chris Kim. Design and Performance Evaluation of Image Processing Algorithms on GPUs. IEEE Transactions on Parallel and Distributed Systems, 2010.
[38] Victor W. Lee et al., "Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU," SIGARCH Comput. Archit. News, 38(3):451–460, 2010.
[39] Montañés Laborda, MA, Torres Moreno, EF, Herrero Jaraba, JE and Martínez del Rincón, J, 'Real-time GPU color-based segmentation of football players' *Journal of Real-Time Image Processing*, vol 7, no. 4, pp. 267-279, 2012.