Real-Time Stopped Object Detection by Neural Dual Background Modeling

Giorgio Gemignani[1], Lucia Maddalena[2], and Alfredo Petrosino[1]

[1] DSA - University of Naples Parthenope
Centro Direzionale, Isola C/4, 80143 Naples, Italy
[2] ICAR - National Research Council
Via P. Castellino 111, 80131 Naples, Italy

Abstract. Moving object detection is a relevant step for many computer vision applications, and specifically for real-time color video surveillance systems, where processing time is a challenging issue. We adopt a dual background approach for detecting moving objects and discriminating those that have stopped, based on a neural model capable of learning from past experience and efficiently detecting such objects against scene variations. We propose a GPGPU approach allowing real-time results, by using a mapping of neurons on a 2D flat grid on NVIDIA CUDA. Several experiments show parallel perfomance and how our approach outperforms with respect to OpenMP implementation.

Keywords: Video Surveillance, Stopped Object Detection, Neural Model, GPGPU.

1 Introduction

Moving object detection in videos is the first relevant step of information extraction in many computer vision applications. The usual approach is through *background subtraction*, that consists in maintaining an up-to-date model of the scene background and detecting moving objects as those that deviate from such a model. Among the objects detected as extraneous to the background, in visual surveillance, specific attention is given to *stopped objects*, i.e. temporally static image regions indicating objects that do not constitute the original background, but were brought into the scene at a subsequent time. Examples are given by abandoned luggage or illegally parked vehicles [2,3,8]. Approaches proposed in literature are either *tracking-based*, i.e. results are obtained based on the analysis of object trajectories through an application-dependent event detection phase (e.g. most of the papers in [2,3]), or *non tracking-based*, i.e. objects are classified without the aid of tracking modules (e.g. [1,4,7]).

We adopt a non tracking-based approach to stopped object detection in image sequences taken by stationary cameras. Based on the use of a double background strategy [7], we construct separate long- and short-term neural self-organizing backgrounds, based on the approach reported in [5]. The long-term background is the usual background model, holding a model for the scene background, while

M.R. Guarracino et al. (Eds.): Euro-Par 2010 Workshops, LNCS 6586, pp. 357–364, 2011.

the short-term model contains temporarily static background elements. Using such models an evidence score is inferred for each pixel as belonging or not to a stationary foreground object. The model, named by us Dual Background SOBS Algorithm, possesses a significant amount of data-level parallelism, suitable for a Single Instruction Multiple Data (SIMD) architecture that allows massively parallel processing. Following the approach of General-Purpose computing on Graphics Processing Units (GPGPU), we decided to exploit the CUDA (Compute Unified Device Architecture) technology on NVIDIA graphical chipsets [6], by organizing both short- and long-term models as neural networks mapped onto 2D flat grids. Section 2 reports the Dual Background SOBS Algorithm, while Sections 3 and 4 respectively report the adopted parallelization technique and performance evaluation.

2 Stopped Object Detection

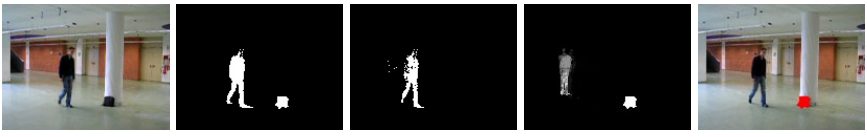
2.1 Dual Background Approach

In order to automatically detect stopped objects in digital color sequences $I_t, t = 1, \ldots, .$ taken by stationary cameras, we adopt a dual background strategy based on the approach proposed in [7]. We construct two separate models (whose specific structure is described in §2.2): a *long-term model* B^L, that models the scene background without moving objects, and a *short-term model* B^S, that contains temporarily static background elements, including moving objects that have been excluded by B^L. By comparing each sequence frame with these models, we obtain two binary masks: the long-term foreground mask F^L, that is true only for objects (both moving and stopped) that are extraneous to the background, and the short-term foreground mask F^S, that is true only for moving objects. An evidence score is inferred at each pixel $\mathbf{x}$ by applying a set of hypotheses on the foreground masks and, to provide temporal consistency, the resulting evidence scores $E_t(\mathbf{x})$ for each pixel $\mathbf{x}$ of sequence frame I_t are aggregated in time as

$$E_t(\mathbf{x}) = \begin{cases} \min(\tau, E_{t-1}(\mathbf{x}) + \Delta t) \text{ if } F^L(\mathbf{x}) \wedge \ !F^S(\mathbf{x}) \\ \max(0, E_{t-1}(\mathbf{x}) - k) \quad \text{if } !F^L(\mathbf{x}) \vee F^S(\mathbf{x}) \end{cases} \tag{1}$$

where $\wedge$, $\vee$ and ! indicate logical *and*, *or* and *not* operations, respectively. If $\mathbf{x}$ is not modeled by B^L but it is modeled by B^S, than it must belong to a stopped object, and correspondingly $E_t(\mathbf{x})$ is incremented by the factor Δt. The maximum value τ for $E_t(\mathbf{x})$ corresponds to the *stationarity threshold*, i.e. the minimum number of consecutive frames after that a pixel assuming constant features is classified as stopped; the value for τ is chosen depending on the desired responsiveness of the system. Otherwise, if $\mathbf{x}$ is modeled by B^L or is not modeled by B^S, than it must belong either to the background or to a moving object, and correspondingly $E_t(\mathbf{x})$ is decremented by the factor k, that determines how fast the system should recognize that a stopped pixel has moved again. When thresholded by τ, the aggregated evidence score computed as in (1) provides a binary mask that is true only for pixels belonging to objects extraneous to the

background and that stay stopped for at least τ consecutive sequence frames. An example of the stopped object detection procedure is shown in Fig. 1. The sequence frame I_t of Fig. 1-(a) shows a bag that has been left on the floor more than τ frames ago, while a man is moving. The corresponding long-term foreground mask F^L shown in Fig. 1-(b) is true (white colored) only for pixels extraneous to the scene background (i.e. pixels that are not modeled by the long-term background B^L), while the short-term foreground mask F^S (Fig. 1-(c)) is true only for moving pixels, since the short-term background has included pixels belonging to the stopped object. The evidence score E_t, whose normalized version is shown as a gray-level image in Fig. 1-(d), has reached the maximum value τ only for pixels of the stopped object, while gets lower values for pixels that where recently moving. Thresholding the evidence score with respect to τ, the stopped object shown in red in Fig. 1-(e) is obtained.



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

Fig. 1. (a) original sequence frame; (b) F^L; (c) F^S; (d) E; (e) stopped object (in red)

2.2 Neural Self-Organizing Background Model

The background model constructed and maintained in SOBS algorithm [5], here adopted for both the long-term and the short-term backgrounds, is based on a self-organizing neural network arranged as a 2-D flat grid of neurons. Each neuron computes a function of the weighted linear combination of incoming inputs, with weights resembling the neural network learning, and can be therefore represented by a weight vector obtained collecting the weights related to incoming links. An incoming pattern is mapped to the neuron whose set of weight vectors is most similar to the pattern, and weight vectors in a neighborhood of such node are updated; such learning of the neuronal map allows to adapt the background model to scene modifications. Specifically, for each pixel $\mathbf{x}$ we build a neuronal map consisting of $n \times n$ weight vectors $b_0^i(\mathbf{x}), i = 1, \ldots, n^2$ where each weight vector is a 3D vector initialized to the color components of the corresponding pixel of the first sequence frame I_0. The complete set of weight vectors for all pixels of an image I_0 with N rows and M columns is represented as a neuronal map B_0 with $n \times N$ rows and $n \times M$ columns, where adjacent blocks of $n \times n$ weight vectors correspond to adjacent pixels in image I_0. The value $n = 3$, suggested and justified in [5], is adopted for all experiments reported in §4. For each frame I_t, the color $I_t(\mathbf{x})$, at position $\mathbf{x}$ is compared to the weight vectors $b_{t-1}^1(\mathbf{x}), \ldots, b_{t-1}^{n^2}(\mathbf{x})$ related to it in the model B_{t-1}, to determine the weight vector $b_{t-1}^{BM}(\mathbf{x})$ that best matches it according to a metric $d(\cdot)$:

$$d(b_{t-1}^{BM}(\mathbf{x}), I_t(\mathbf{x})) = \min_{i=1,\ldots,n^2} d(b_{t-1}^i(\mathbf{x}), I_t(\mathbf{x})). \quad (2)$$

The best matching weight vector is used as the pixel's encoding approximation, and therefore $\mathbf{x}$ is detected as foreground if the distance in (2) exceeds a threshold ϵ; otherwise, it is classified as background. The adopted color space, the metric $d(\cdot)$ and the threshold ϵ can be suitably chosen as in [5].

Learning is able to adapt the background model B_{t-1} to scene modifications and is achieved by updating the best matching weight vector $b_{t-1}^{BM}(\mathbf{x})$, supposed at position $\mathbf{z}$ of B_{t-1}, and all other weight vectors in its neighborhood $N_{\mathbf{Z}}$ according to:

$$b_t(\mathbf{y}) = (1 - \alpha_t(\mathbf{y}, \mathbf{z}))b_{t-1}(\mathbf{y}) + \alpha_t(\mathbf{y}, \mathbf{z})I_t(\mathbf{x}), \quad \forall \mathbf{y} \in N_{\mathbf{Z}} \tag{3}$$

Here $\alpha(\mathbf{y}, \mathbf{z}) = \gamma_t \cdot G(\mathbf{y}\text{-}\mathbf{z})$, where γ_t represents the learning rate, that depends from scene variability, while $G(\cdot) = G(\cdot; 0, \sigma^2)$ are the values of a Gaussian filter with zero mean and σ^2 variance in $N_{\mathbf{Z}}$. For the purpose of the double background approach to stopped object detection, the long-term background model B_t^L is updated according to (3) in a *selective* way, i.e. only if $d(\cdot, \cdot) \leq \varepsilon$, otherwise, it remains unchanged. Such selectivity allows to adapt the background model to scene modifications without introducing the contribution of pixels not belonging to the background scene. Instead, the short-term background model B_t^S is always updated according to (3), with a learning rate γ_t^S higher than the learning rate γ_t^L for the long-term model, so that it can quickly include moving and temporarily static background elements that have been excluded by the long-term model.

2.3 The Algorithm

The stopped object detection procedure described in the previous section for each pixel $\mathbf{x}$ can be sketched as the following algorithm:

Dual Background SOBS Algorithm

Input: pixel $\mathbf{x}$ in sequence frame $I_t, t = 0, \ldots,$ LastFrame
Output: aggregated evidence score $E_t(\mathbf{x})$

1. $InitializeModels(B_0^L(\mathbf{x}), B_0^S(\mathbf{x}))$
2. **for** t=1, Kinit
3. $\quad CalibrateModels(B_t^L(\mathbf{x}), B_t^S(\mathbf{x}))$
4. **for** t=Kinit+1, LastFrame
5. $\quad (F^L(\mathbf{x}), F^S(\mathbf{x})) = UpdateModels(B_t^L(\mathbf{x}), B_t^S(\mathbf{x}), I_t(\mathbf{x}))$
6. $\quad E_t(\mathbf{x}) = ForegroundCompare(F^L(\mathbf{x}), F^S(\mathbf{x}))$

Steps 1-3 represent the *calibration phase*, that involves initial learning of the two neural networks modeling the long-term and the short-term backgrounds over Kinit initial frames, while steps 4-6 represent the *online phase*, that involves the updating of both neural network models and the computation of the evidence score based on (1), (2), and (3).

Most of the computation for each pixel can be done concurrently, independently from adjacent pixels. Communication is only needed for the update of the background models in the case that the neighborhood of best matching weight vector, computed according to (2), contains weight vectors of adjacent pixels.

A suitable parallelization strategy is therefore based on data decomposition, where each thread is responsible of data needed to perform the whole Dual Background SOBS Algorithm for a single pixel.

3 Parallelizing Stopped Object Detection

To afford the parallelization approach we should firstly point out some costraints of CUDA architectures. Parallel programs for CUDA are organized into three levels of abstraction: elementary sequences of instructions (*threads*) are clustered into different *blocks*, which are themselves grouped into *grids*. All threads execute the same sequence of instructions on different data. Each block is executed on one multiprocessor (consisting of 8 processors), which can alternate with other blocks in order to hide latencies due to not-cached memory accesses. Once a block is assigned to a multiprocessor, it is divided into 32 thread units (called *warps*), effectively scheduled such that the threads within a warp are executed in a somewhat lock-step way called *single-instruction multiple-thread*. Whenever the number of blocks is higher than the number of available multiprocessors, the remaining blocks are queued.

Given a sequence frame consisting of $M \times N$ pixels and fixed the number $th_x \times th_y$ of threads for each block, we define a grid G consisting of $(\frac{2\times M}{th_x}) \times (\frac{N}{th_y})$ blocks. The grid G is subdivided into two subgrids, G_S and G_L, each consisting of $(\frac{M}{th_x}) \times (\frac{N}{th_y})$ blocks, running simultaneously. The subgrid G_S processes the short-term background model B^S, while the subgrid G_L processes the long-term background model B^L.

The implementation is divided into four different *kernels* (program modules executed independently on different data), corresponding to steps 1, 3, 5, and 6 of the Dual Background SOBS Algorithm (cfr. §2.3): the initialization, the calibration, the model update, and the evidence computation, which are launched sequentially by the CPU. The only exchange of memory between CPU and GPU is the one for loading the input images into the global memory device and for sending back the resulting binary evidence image. Since global memory is not cached, it is particularly important to follow the right access pattern to get the maximum bandwidth. The device is capable of reading 4-byte, 8-byte, 16-byte words from memory into registers in a single instruction; for structures larger than 16 bytes the compiler generates several load instructions. Moreover, global memory bandwidth is most efficiently used when the simultaneous memory access by threads in half-warp can be coalesced into a single memory transaction of 32, 64, or 128 bytes, respectively, and this is achieved if all threads access 1-byte, 2-byte, or 4-byte words lying in the same segment, respectively.

Since usually images are stored into 1D arrays in row-major order, we reorganize the image data before transferring them to the global memory device in order to guarantee coalesced access during the parallel processing. As an example, in Figure 2-(a) we show a simple image consisting of $M \times N = 6 \times 6$ pixels, distributed on a grid of 2×2 blocks (identified by different colors), where each

block consists of 3×3 threads which are assigned one pixel each. The usual storing into a 1D array in row-major order (shown in Figure 2-(b)) is reorganized as in Figure 2-(c), so that each block accesses data as sequential chunks of the 1D array. This preprocessing phase is done sequentially and should be taken into account for time measurements, as described in the next section.

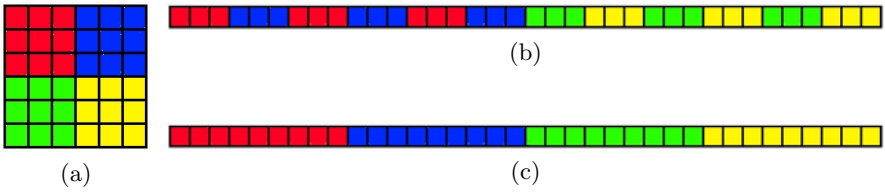


Fig. 2. (a) Image consisting of 6×6 pixels distributed on a grid of 2×2 blocks (identified by different colors), each consisting of 3×3 threads, having one pixel each; (b) usual image memorization pattern; (c) reorganized image memorization pattern

4 Experimental Results

In order to evaluate the performance of the parallel version of the Dual Background SOBS Algorithm we considered several color image sequences resized at a resolution of $M \times N = 720 \times 480$ pixels, belonging to the publicly available i-LIDS 2007 dataset (available at ftp://motinas.elec.qmul.ac.uk/pub/iLids/), that represent typical situations where stopped objects can be of great concern (abandoned luggage in train stations and parked vehicles in no parking zones). Experiments were performed on two Intel Core i7 CPUs at 2.67GHz equipped with two different NVIDIA GPUs, both based on the G80 architecture: a Tesla C1060 (in the following referred to as Tesla), with 30 multiprocessors and 4 GB of global memory, and a GeForce 8400GS (in the following referred to as GeForce), with 2 multiprocessors and 300 MB of global memory. Concerning memory into the NVIDIA G80 chipset, each thread accesses only 32 registers and each block has 16 KB of shared (cached) memory common to all its threads. Furthermore, as memory transfer between CPU and GPU is very time consuming, it is preferable to perform all the calculations on data stored in the global memory. The Tesla card contains 16384 registers, and thus only 512 threads per block should be active at a time; the number of threads per block has to be chosen between 64 and 512 in order to optimize the block distribution and avoid latency. The GeForce card contains 8192 registers, and thus only 256 threads per block should be active at a time; the number of threads per block has to be chosen between 64 and 256 in order to optimize the block distribution and avoid latency.

The serial implementation in such environment has an average execution time $T_{SEQ} = 431.68ms$ per frame; this accounts for only $2.3165fps$, much lower than the $24fps$ required for real-time processing of an image sequence. The parallel processing time T_{PAR} is given by the sum of a sequential time t_s (elapsed time to

acquire an image from the video and to re-arrange the image data as described in §3) and a parallel time t_p (elapsed time from the start of storage of the re-arranged image on global memory device to the end of the computation of the binary evidence image). To compare performance on Tesla and GeForce, we fix 8×8 threads per block (optimal configuration supported by GeForce to process the considered 720×480 image sequences) corresponding to a grid of 180×60 blocks, that simultaneously processes B^L and B^S. On GeForce T_{PAR} is $280.13ms$, consisting of $t_s = 3.73ms$ and $t_p = 276.40ms$, while on Tesla T_{PAR} is $20.55ms$, with $t_s = 3.73ms$ and $t_p = 16.82ms$. It turns out that achieved speedups as compared to serial time T_{SEQ} (1.6x for the GeForce and 20x for the Tesla) can be considered satisfactory, although the implementation with GeForce still does not allow for real-time execution. Indeed, the implementation on GeForce achieves $3.57fps$, while that on Tesla $48.64fps$. Other experiments on 720×480 image sequences have been carried out on the most performing Tesla GPU varying the number of block threads, in order to measure the overhead given by block scheduling as shown in Table 1. The number of active blocks per

Table 1. Parallel times T_{PAR} (in ms), Speedups, and fps on Tesla for image size 720×480 and different configurations of Block and Grid sizes

Block size	*Grid size*	T_{PAR}	*Speedup*	*fps*
8×4	180×120	22.63	19.07x	44.18
8×8	180×60	20.84	20.70x	47.97
16×8	90×60	20.55	21.00x	48.64
16×16	90×30	22.19	19.45x	45.05
20×8	72×60	20.86	20.68x	47.91

multiprocessor depends on how many registers per thread and how much shared memory per block are required for a kernel, since the multiprocessor registers and shared memory are split among all the threads of the active blocks. Allocating more threads per block is better for efficient time slicing, but the more threads per block the fewer registers are available per thread, which might prevent the kernel invocation from succeeding.

Further experiments have been done varying the size of the image sequences to assess the extent to which the requirement of real-time processing ($24fps$) is guaranteed. Results on 720×480 , 960×720 and 1200×960 image sequences, with an average execution serial time $T_{SEQ} = 431.68ms$, $T_{SEQ} = 862.94ms$ and $T_{SEQ} = 1430.30ms$ per frame, respectively, are reported in Table 2(a). Finally, we have designed an OpenMP implementation that parallelizes the algorithm on the 8 cores of our CPU through the definition of 8 independent threads working on data stored on shared memory. The parallel processing time (Table 2(b)) reduces 6 times: multicore architecture gets better performance with respect to GeForce 8400GS, even if the real time requirement is not yet satisfied.

Table 2. Parallel times (ms), speedups, and fps on (a) Tesla and (b) OpenMP

(a) Tesla

Image Size	*Block size*	T_{PAR}	*Speedup*	*fps*
720×480	16×8	20.55	21x	48.64
960×720	8×8	40.96	21x	24.9
1200×960	16×8	65.41	21.8x	15.2

(b) OpenMP implementation

Image Size	T_{PAR}	*Speedup*	*fps*
720×480	76.42	5.64x	13.08
960×720	152.08	5.67x	6.57
1200×960	254.25	5.62x	3.9

5 Conclusions

We have described a dual background approach to stopped object detection in digital image sequences based on self-organizing neural networks, proposing a data parallel algorithm, which is specifically suitable for SIMD architectures. Parallel performance of our CUDA implementation with two different GPUs has been analyzed, concluding that with the Tesla C1060 GPU significant speedups can be achieved as compared to our serial implementation, allowing for real-time stopped object detection. Further experiments have been carried out in order to check different configurations of the parallel implementation, varying the number of threads, the number of pixels per thread, the number of blocks per multiprocessor and the size of image sequences, also making comparisons with an OpenMP implementation.

References

1. Collins, R.T., et al.: A System for Video Surveillance and Monitoring, The Robotics Institute, Carnegie Mellon University, Tech. Rep. CMU-RI-TR-00-12 (2000)
2. Ferryman, J.M. (ed.): Proceedings of the 9th IEEE International Workshop on PETS, New York, June 18 (2006)
3. Ferryman, J.M. (ed.): Proceedings of the 10th IEEE International Workshop on PETS, Rio de Janeiro, Brazil, October 14 (2007)
4. Herrero-Jaraba, E., et al.: Detected Motion Classification with a Double-background and a Neighborhood-based Difference. Patt. Recogn. Lett. 24, 2079–2092 (2003)
5. Maddalena, L., Petrosino, A.: A Self-Organizing Approach to Background Subtraction for Visual Surveillance Applications. IEEE Transactions on Image Processing 17(7) (July 2008)
6. NVIDIA, CUDA guide, `http://www.nvidia.com/object/cuda_home_new.html`
7. Porikli, F., Ivanov, Y., Haga, T.: Robust Abandoned Object Detection Using Dual Foregrounds. EURASIP Journal on Advances in Signal Processing (2008)
8. Proc. of Fourth IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS 2007). IEEE Computer Society, Los Alamitos (2007)