NEURAL DUAL BACKGROUND MODELING FOR REAL-TIME STOPPED OBJECT DETECTION

Gemignani G., Maddalena L., Petrosino A.

Abstract

Stopped object detection is a relevant step for computer vision applications and mainly in real-time vision systems where processing time is a challenging issue.
We propose a dual background approach for detecting stopped objects based on a neural background model capable of learning from past experience and efficiently detecting stopped objects against light variations, shadows, etc.
In our approach neurons are organized as a 2D flat grid on CUDA, a SIMD technology for high-performance parallel computing on NVIDA GPUs. Achieved results show high detection accuracy and parallel efficiency.

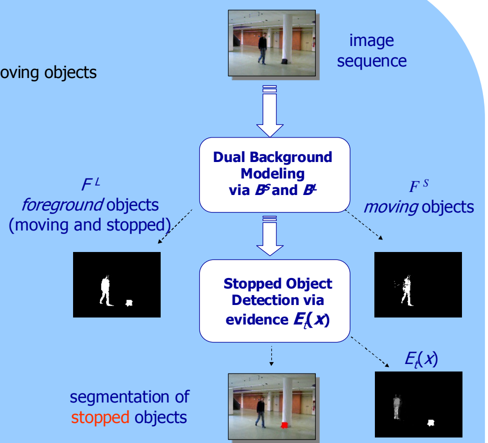
Dual Background Approach

1. **Construct 2 separate models**
   - Long-term model B^L: usual background model adopted for moving object detection, modeling the scene background without moving objects
   - Short-term model B^S: contains temporally static background elements, including moving objects that have been excluded by B^L
2. **Compare each sequence frame I_t with the 2 models and compute 2 foreground binary masks**
   - Long-term foreground mask F^L: contains stopped and moving objects
   - Short-term foreground mask F^S: contains only moving objects
3. **For each pixel compute an evidence score by applying a set of hypotheses on the foreground masks**

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

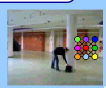
τ stationary threshold: minimum number of consecutive frames after which a pixel is classified as static

k decay factor: determines how fast the system should recognize that a stopped pixel has moved again



Neural Self Organizing Background Model

The background model constructed and maintained in **SOBS** algorithm, here adopted for both the long-term and the short-term backgrounds, is based on a self organizing neural network organized as a **2-D flat grid of neurons** [Maddalena & Petrosino, TIP'08]. 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.



1. For each pixel $\boldsymbol{x}$, build a neuronal map consisting of $n \times n$ weight vectors all represented in **HSV** color space
   – Each of the n^2 weight vectors $b_t^i(\boldsymbol{x})$ is a 3D vector initialized to the corresponding pixel components of first sequence frame I_0:

$$b_0^i(\boldsymbol{x}) = I_0(\boldsymbol{x}), \quad i = 1, \ldots, n^2$$

2. By subtracting the current Image I_t from the background model B_t at each subsequent time instant t, every pixel $\boldsymbol{x}$ of I_t is compared to current pixel weight vectors ($b_t^1(\boldsymbol{x})$, …, $b_t^L(\boldsymbol{x})$) to determine the weight vector $b_t^{BM}(\boldsymbol{x}) = B_t(\boldsymbol{z})$ that best matches it according to a metric $d(\cdot)$:

$$d(b_t^{BM}(\boldsymbol{x}), I_t(\boldsymbol{x})) = \min_{i=1,\ldots,n^2} d(b_t^i(\boldsymbol{x}), I_t(\boldsymbol{x}))$$

HSV colour space : $I(\boldsymbol{x}_i) = (h_i, s_i, v_i), I(\boldsymbol{x}_j) = (h_j, s_j, v_j) \quad d(I(\boldsymbol{x}_i), I(\boldsymbol{x}_j)) = \| (v_i s_i cos(h_i), v_i s_i sin(h_i), v_i) - (v_j s_j cos(h_j), v_j s_j sin(h_j), v_j) \|$

3. Weight vectors are updated in a neighborhood of best matching neuron (**adaptivity of the model**).
   Updating the model B_t in a neighborhood $N_{\boldsymbol{z}}$

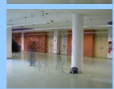
$$B_t(\boldsymbol{y}) = (1 - a_t(\boldsymbol{y}, \boldsymbol{z})) B_{t-1}(\boldsymbol{y}) + a_t(\boldsymbol{y}, \boldsymbol{z}) I_t(\boldsymbol{x}), \quad \forall \boldsymbol{y} \in N_{\boldsymbol{z}} \quad (1)$$

⇨ Reinforcement of center pixel's model and of the model of pixels adjacent to sample $\boldsymbol{x}$ (adjacent pixels move accordingly)

Gaussian weights

$$a_t(\boldsymbol{y}, \boldsymbol{z}) = \gamma_t G(\boldsymbol{y} - \boldsymbol{z}) \qquad \gamma_t = \frac{\beta_t}{\max\{G(\boldsymbol{y} - \boldsymbol{z})\}}, \quad \beta_t \in [0,1] \quad \text{s.t. } a_t(\boldsymbol{y}, \boldsymbol{z}) \in [0,1]$$

4. For the purpose of the double background approach to stopped object detection:
   - B_t^L is updated according to (1) in a ***selective* way**, only if $d(b_t^{BM}(\boldsymbol{x}), I_t(\boldsymbol{x})) < \varepsilon$ ⇨ Background model adapts to scene modifications without introducing the contribution of pixels not belonging to the background scene
   - B_t^S is updated according to (1) in a ***non selective way***, with $\gamma_t^S >> \gamma_t^L$ ⇨ Quick inclusion of moving and temporarily static background elements that have been excluded by the long-term model

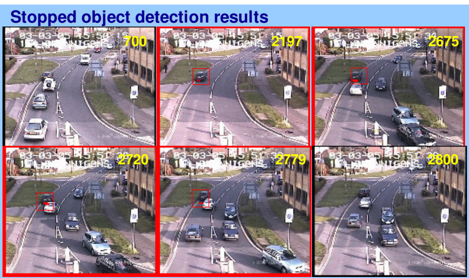


Dual Background SOBS Algorithm

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

```
InitializeModels(B_0^L(x), B_0^S(x))
for t=1, Kinit
  CalibrateModels(B_t^L(x), B_t^S(x))
for t=Kinit+1, LastFrame
  (F^L(x), F^S(x)) = UpdateModels(B_t^L(x), B_t^S(x), I_t(x))
  E_t(x) = ForegroundCompare(F^L(x), F^S(x))
```

Stopped object detection results



i-LIDS'07 dataset - stopped vehicles in no parking areas [ftp://motinas.elec.qmul.ac.uk/pub/iLids]:

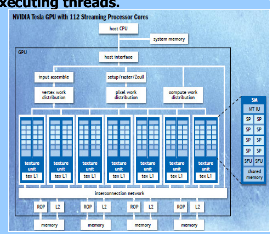
- **Stationary threshold**: τ=1500
- **Strong illumination variations** (clouds)

A [Boragno et al., Proc. AVSS 2007]
B [Guler et al., Proc. AVSS 2007]
C [Lee et al., Proc. AVSS 2007]
D [Porikli et al., EURASIP JASP 2008]
E [Venetianer et al., Proc. AVSS 2007]
DBSOBS [Gemignani,Maddalena & Petrosino, submitted to UCHCP 2010)]

Seq.	Event	GT	DBSOBS	ε_{DBSOBS}	A	ε_A	B	ε_B	C	ε_C	D	ε_D	E	ε_E
PV-easy	Start	02:48	02:44	4	02:48	0	02:46	2	02:52	4	02:51	3	02:52	4
"	End	03:15	03:20	5	03:19	4	03:18	3	03:19	4	03:13	2	03:16	1
PV-medium	Start	01:28	01:28	0	01:28	0	01:28	0	01:41	13	01:33	5	01:43	15
"	End	01:47	01:52	5	01:55	8	01:54	7	01:55	8	01:50	3	01:47	0
PV-hard	Start	02:12	02:12	0	02:12	0	02:13	1	02:08	4	02:13	1	02:19	7
"	End	02:33	02:34	1	02:36	3	02:36	3	02:37	4	02:32	1	02:34	1
Total err				**15**		**15**		**16**		**37**		**15**		**28**

Cuda Programming Model

The G-80 architecture is built around a scalable array of multithreaded **SMs** (Streaming Multiprocessors).
Current GPU implementations **range from 768 to 12,288 concurrently executing threads.**



- When a CUDA program on the host CPU invokes a kernel grid, the **CWD** (Compute Work Distribution) unit numbers the blocks of the grid and begins distributing them to **SMs** with available execution capacity
- The threads of a thread block execute concurrently on one **SM**. As thread blocks terminate, the **CWD** unit launches new blocks on the vacated multiprocessors.
- An **SM** consists of 8 scalar **SP** (Scalar Processor) cores, two **SFUs** (Special Function Units) for transcendental functions, an **MT IU** (Multi-Threaded Instruction Unit), and on-chip shared memory
- The **SM** creates, manages, and executes up to **768** concurrent threads in hardware with zero scheduling overhead

SIMT Architecture (Single Instruction Multiple Thread)

- The **SM** maps each thread to one **SP** scalar core, and each scalar thread executes independently with its own instruction address and register state
- The **SM SIMT** unit creates, manages, schedules, and executes threads in groups of **32 parallel threads,** called ***warps***
- At every instruction issue time, the **SIMT** unit selects a warp that is ready to execute and issues the next instruction to the active threads of the warp

Memory Model

Registers
Per thread
Data lifetime = thread lifetime
Local memory
Per thread off-chip memory (in device DRAM)
Data lifetime = thread lifetime
Shared memory
Per thread block on-chip memory
Data lifetime = block lifetime
Global (device) memory
Accessible by all threads as well as host (CPU)
Data lifetime = from allocation to deallocation
Host (CPU) memory
Not directly accessible by CUDA threads

Parallel Scheme

	No. of Task	Task	Computational complexity	CPU-GPU processing	GPU kernel Grid size
Off-Line	1	Init Models	$O(2 \times M \times N \times n^2)$	**GPU** processing Pixel-level parallelism	$G_1((2 \times M)/th_x, N/th_y)$
		For t=1, Kinit		**CPU** processing	
	2	Calibrate Models	$O(2 \times M \times N \times n^2)$	**GPU** processing Pixel-level parallelism	$G_1((2 \times M)/th_x, N/th_y)$
On-Line		For t=Kinit+1, Lastframe		**CPU** processing	
	3	Update Models	$O(2 \times M \times N \times n^2)$	**GPU** processing Pixel-level parallelism	$G_1((2 \times M)/th_x, N/th_y)$
	4	Foreground Compare	$O(2 \times M \times N)$	**GPU** processing Pixel-level parallelism	$G_2(M/th_x, N/th_y)$

Given a sequence image consisting of $M \times N$ pixels, and fixed the number of threads per block $th_x \times th_y$
We generate a grid of blocks $G_1((2 \times M)/th_x, N/th_y)$

We split G_1 into 2 subgrids G_S, G_L of blocks:
- G_S processes the short-term background model B_t^S
- G_L processes the long-term background model B_t^L

We generate a grid of blocks $G_2(M/th_x, N/th_y)$ to calculate evidence image E

Speedup

- Serial implementation:
  Intel Core i7 CPU at 2.67GHz 543*ms* per frame
- Parallel implementation:
  Tesla C1060 (**30 SMs**)

Time measurements for the on-line phase for *AB-Easy* sequence:

Number of threads x block	Grid size (in blocks)	Time	Speedup
8 x 4	160 x 120	12.91*ms*	45*x*
8 x 8	160 x 60	7.94*ms*	73*x*
16 x 16	80 x 30	7.99*ms*	72*x*
20 x 16	**40 x 30**	**7.1 *ms***	**82*x***

Increasing the block size and maintaining a large number of blocks we observe an improvement in performance