Color and motion-based particle filter target tracking in a network of overlapping cameras with multi-threading and GPGPU

Rastreo de objetivos por medio de filtros de partículas basados en color y movimiento en una red de cámaras con multi-hilo y GPGPU

Francisco Madrigal*, Jean-Bernard Hayet*

ABSTRACT

This paper describes an efficient implementation of multiple-target multiple-view tracking in video-surveillance sequences. It takes advantage of the capabilities of multiple core Central Processing Units (CPUs) and of graphical processing units under the Compute Unifie Device Arquitecture (CUDA) framework. The principle of our algorithm is 1) in each video sequence, to perform tracking on all persons to track by independent particle filters and 2) to fuse the tracking results of all sequences. Particle filters belong to the category of recursive Bayesian filters. They update a Monte-Carlo representation of the posterior distribution over the target position and velocity. For this purpose, they combine a probabilistic motion model, *i.e.* prior knowledge about how targets move (*e.g.* constant velocity) and a likelihood model associated to the observations on targets. At this first level of single video sequences, the multi-threading library Threading Buildings Blocks (TBB) has been used to parallelize the processing of the per-target independent particle filters. Afterwards at the higher level, we rely on General Purpose Programming on Graphical Processing Units (generally termed as GPGPU) through CUDA in order to fuse target-tracking data collected on multiple video sequences, by solving the data association problem. Tracking results are presented on various challenging tracking datasets.

RESUMEN

Este artículo describe una implementación eficiente de un algoritmo de seguimiento de múltiples objetivos en múltiples vistas en secuencias de video vigilancia. Aprovecha las capacidades de las Unidades Centrales de Procesamiento (CPUs, por sus siglas en inglés) de múltiples núcleos y de las unidades de procesamiento gráfico, bajo el entorno de desarrollo de Arquitectura Unificada de Dispositivos de Cómputo (CUDA, por sus siglas en inglés). El principio de nuestro algoritmo es: 1) aplicar el seguimiento visual en cada secuencia de video sobre todas las personas a seguir con filtros de partículas independientes y 2) fusionar los resultados de seguimiento de todas las secuencias. Los filtros de partículas pertenecen a la categoría de filtros Bayesianos recursivos. Actualizan una representación Monte-Carlo de la distribución posterior sobre la posición y la velocidad de los objetivos. Para este fin, combinan un modelo probabilístico de movimiento, es decir un conocimiento a priori de como se mueven los objetivos (por ej. velocidad constante) y un modelo de verosimilitud asociado con las observaciones de los objetivos. En este primer nivel de procesamiento de las secuencias de video simples, la librería multi-hilo (TBB, por sus siglas en inglés) es utilizada para paralelizar el procesamiento de los filtros de partículas asociados a cada objetivo. Luego, al nivel superior, utilizamos Programación de Propósito General con Unidades de Procesamiento Gráficas (conocido por su acrónimo en inglés GPGPU) a través de CUDA con el fin de fusionar los datos del seguimiento de objetivos colectados entre las diferentes secuencias de video, al resolver el problema de asociación de datos. Los resultados del seguimiento son presentados en algunas bases de datos desafiantes.

Recibido: 9 de agosto de 2012
Aceptado: 30 de enero de 2013

Keywords:
Particle filter, multi-view tracking, GPGPU, multi-threading.

Palabras clave:
Filtro de partículas, seguimiento en vistas múltiples, GPGPU, multi-hilo.

INTRODUCTION

One of the most recent and striking trends in the market of public safety has been the skyrocketing development of video-surveillance systems. Thousands of cameras have invaded most downtown areas in large cities with the primary motivation of using them as a strong dissuasive tool for potential criminals, and when possible, aiming to provide live monitoring tools and give forensic evidence to solve crimes. However, so far the re-

*Computer Science Group, Centro de Investigación en Matemáticas. Jalisco S/N. Valenciana, Guanajuato, Gto., México. Zip code 36240. Phone:+ 52-473-732-7155. Fax:+52-473-732-5749. E-mail: pacomd@cimat.mx, jbhayet@cimat.mx, mrivera@cimat.mx

sults have been quite disappointing, mostly because in many situations human agents are let on their own with dozens of video sequences to monitor. Hence research efforts for the development of next generation video-surveillance systems have focused on the automation of monitoring tasks, using, for example, automatically generated alerts when suspect events take place. One of the key elements for that purpose is the tracking system, *i.e.* a program in charge of detecting people in the observed scene and for preserving their identities along all video sequences. This is not a trivial task in real life situations. Because of the projective nature of a camera, occlusions among visible people are unavoidable which make the tracking difficult. Indeed the system has to maintain the presence of the occluded persons while they are not observable, and it must be careful not to invert the identities of people crossing each other. One of the most efficient strategies to overcome this problem is hardware, *i.e.* to rely on multiple cameras with overlapping view fields (Black & Ellis, 2002). With the help of various views on a same scene, most occlusion problems can be solved.

In Liem & Gavrila, (2009), 3D moving blobs are reconstructed by intersecting all visible 2D blobs in video streams. Another example is (Yao & Odobez, 2008), where particle filters allow estimating targets position in the ground level, based on observations from the principal axis of the detected blobs. In Berclaz, Fleuret, Turetken & Fua, (2011), an occupancy map-based method accumulates different motion detections made in the cameras video streams and an optimization scheme determines the most probable trajectories followed during the last frames.

This article proposes a software architecture for such a multi-camera tracking system, taking full advantage of recent advances on parallel computing, namely multiple-threading and General Purpose Programming on Graphical Processing Units (GPGPU). The former refers to the ability of single or multiple core systems to run several lightweight processes within a common process. The latter refers to the massively parallel use of graphical boards processing units in more general applications than mainly graphical processing, for which they have been initially designed.

An overview of the tracking strategy is given, and then the two main components of our system are described: 1) Local trackers running in separate threads for each video sequence; 2) Data association of local trackers estimates to global trackers. We present interesting results on public tracking datasets, study the effect of parallelizing local tracking and data association tasks, and finally develop conclusions over the entire approach.

Overview

The overview of our approach is summed in figure 1. Video sequences corresponding to all the cameras are processed separately and a set of trackers is run for each of them. One tracker is associated to each person and it continuously estimates the position of this person in the image. We will refer to each view with indices $j = 1 \ldots V$ where V is the number of views. Then trackers in view will be denoted by t_k^j for $k = 1 \ldots N_j$ (N_j is the number of trackers in view j). Note that these trackers estimate the position of targets in the image in pixel coordinates. We will suppose that we have the geometric knowledge of how the image points are mapped to the real world, which is supposed to be planar. It is well known that when considering a planar scene, the relation between the image coordinates and real world coordinates is through a homography (Du, Hayet, Verly & Piater, 2009). In our case these homographies are given after a phase of calibration.

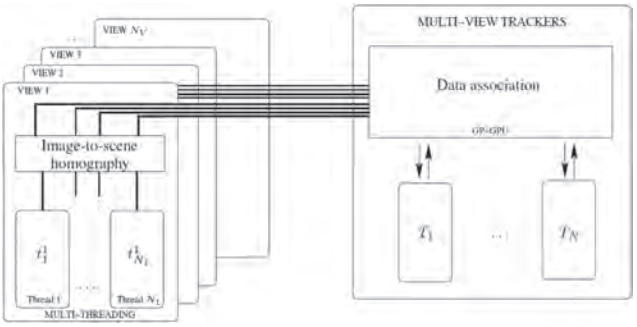

Figure 1. Overview of tracking architecture: Local trackers based on particle filters run independently on separate threads for each target and for each view. The estimates of all these *local* trackers are then treated as observations for *global* trackers running at a higher level. The association of local trackers estimates to global trackers is done through GPGPU.

All these trackers run according to a recursive Bayesian technique and are implemented as sequential Monte-Carlo filters also named particle filters (Du, Hayet, Verly & Piater, 2009). They will be describe in detail in the next section. Multi-threading is used at this step with the help of the Threading Building Blocks (TBB) library from Intel. For the next step, *i.e.* the fusion of the information collected in video sequences into estimates at a global level, the estimates that we get from local trackers are first converted into real-world coordinates (through the image-to-scene homography) for allowing the fusion to be done properly.

As it has been shown in figure 1, this fusion relies on trackers which are similar to the previous ones. They provide estimates of real world coordinates for each person under camera scrutiny. Again, these trackers are kept independent and are referred to

as T_k for $k = 1, \ldots, N$, where N is the number of persons globally tracked. Their observations come from projected estimates of local trackers. Therefore, the main problem is to associate each particle (weighted sample), from each filter, to only one observation from each camera (problem also known as data association). As this problem has a combinatorial nature and even if the computational burden can be made easier, it remains complex. Hence as explained in the upcoming section, we make use of General Purpose GPU for massively accelerating data association and performing updates of all particles in parallel.

Local trackers

As ibis mentioned above, local trackers t_k^j in view j,which are in charge of estimating the position in images of observed targets, rely on recursive Bayesian estimators. Let the state of the tracker – the 4 × 1 vector holding all the quantities of interest, namely the position in x and y and velocity in x and y – be $\mathbf{X}_{k,\tau}^j$. The additional index τ refers to the time instant. In this Bayesian scheme, it used prior information on the target motion and observations (image cues) extracted from the video sequence to derive the posterior distribution $p(\mathbf{X}_{k,\tau}^j \mid \mathbf{Y}_{1:\tau}^j)$, where $\mathbf{Y}_{1:\tau}^j$ is the collection of all observations (images) extracted in view j up to time τ (*i.e.* $\mathbf{Y}_{1:\tau}^j = \mathbf{Y}_{1,}^j \mathbf{Y}_{2,}^j, \ldots, \mathbf{Y}_{\tau}^j$).

It prior for this Bayesian inference problem is given by the assumption of a constant velocity linear model *i.e.* for target k in view j,

$$\mathbf{X}_{k,\tau+1}^j = \begin{bmatrix} \mathbf{I}_{2\times2} & \mathbf{I}_{2\times2} \\ 0 & \mathbf{I}_{2\times2} \end{bmatrix} \mathbf{X}_{k,\tau}^j + S_{x,y}^j(\bar{\mathbf{X}}_{k,\tau}^j)\nu, \tag{1}$$

where υ is a zero-mean Gaussian noise and $S_{x,y}^j(\bar{\mathbf{X}}_{k,\tau}^j)$ is a scale factor evaluated at the mean of $\mathbf{X}_{k,\tau}^j$, $\bar{\mathbf{X}}_{k,\tau}^j$. This factor scales the amount of noise and incorporates the geometric knowledge about a view j, if available. For example, it scales down the motions which are far from the camera and amplifies those close to it. This mapping is described with more details in Du, Hayet, Verly & Piater, (2009).

The second element in Bayesian inference is the inclusion of observations $\mathbf{Y}_{\tau}^j$. A somewhat classical probabilistic generative model of the appearance of the object we are tracking is used, derived from the one in Pérez, Vermaak & Blake, (2004) and based on a combination of color and motion histograms. Basically, these histograms are discrete representations of the probability distribution of the corresponding features. They are built in the following way: First, any newly acquired image is subtracted from the previous one in order to produce a difference image; the idea is that absolute values of this different image pixels are representative of motion. Second, each particle in the particle filter gives a target position candidate (*i.e.*, a rectangle in the image), in which we compute the color (HSV channels) and motion (absolute values of gray-value differences (Du, *et al.*, 2009) histograms relative to this area. Histograms are built by counting, for a given discretization of the value range of some interest variable, how many pixels are in each interval.

For both motion and color cues, likelihoods are evaluated through the Bhattacharya histogram distance D, between a reference histogram $\tilde{h}_k^j$ and a current histogram h_k^j corresponding to the target state $\mathbf{X}_{k,\tau}^j$ and extracted from the current image.

The Bhattacharya distance is defined between two histograms h and h' as:

$$D(h,h') = -\ln \sum_n \sqrt{h_n h'_n},$$

where n covers the support of both distributions. It defines the corresponding likelihood as:

$$P(\mathbf{Y}_{\tau}^j \mid \mathbf{X}_{k,\tau}^j) \propto \exp\left(-\sum_{c\in H,S,V} \frac{1}{2\sigma_c^2} D^2(h_k^{j,c}, \tilde{h}_k^{j,c})\right) \times \exp\left(-\frac{1}{2\sigma_m^2} D^2(h_k^{j,m}, \tilde{h}_k^{j,m})\right). \tag{2}$$

The first term represents the color-based likelihood where the exponent refers to the channel in the Hue, Saturation Value (HSV) color space (*i.e,* we use histograms $h_k^{j,c}$ for $c \in$ HSV). Note that σ^2 is the variance on the Bhattacharya distance and it is specific to each cue channel. Reference histograms are initialized in the first frame. The target is detected and updated each time the quality of tracking is well evaluated. Also note that we incorporated a bit of spatial information along with color distribution with two histograms per channel instead of one, which are defined on the upper half and on the lower half bounding boxes. The previous likelihood definition stays the same except in that histograms in each channel have a double dimension. Moreover, we incorporate an image motion model based on absolute differences between consecutive images. These differences are accumulated in a histogram $h_k^{j,m}$ and incorporated again in equation (2).

Target detection allows initializing the trackers based on the principle of blob detection. A background subtraction algorithm has been used (Hernandez–Lopez & Rivera, 2010) to generate a binary image with multiple blobs corresponding to zones in motion in the image (figure 2). These blobs may relate to people in

the scene or to other artifacts. Each blob is segmented and filtered with simple rules (based on its size and the ratio of its width/height), those that have the dimension of a person (according to $S^j_{x,y}$) remain. A blob generates a new tracker if and only if no other tracker is close to it.

To summarize the earlier, the algorithm has been running on a per thread basis, for each tracker, in each view.

(a)

Figure 2. Background subtraction: (a) Input image, frame 332 of PETS 2009. (b) Target detection results, six blobs formed from the pedestrians on the scene. Also, some clutter is detected.

Data association and global trackers

In the previous section, the different threads that are run to estimate individual target states in all video sequences are described. In this section we propose performing associations across the different views that are used. This idea is based on running independent particle filters, and associating them to physical entities evolving in the ground plane. For each view, we try to associate at most one observation per view to any particle on the ground. These observations are simply the projection of the mean estimate of each local tracker, on the ground plane represented in figure 3 by the colored ellipses, which depict the covariance matrices on these distributions. The mapping is done with the same homography which was mentioned before.

Figure 3. Data association: Projected estimates. The ellipses depict the variance of the projected single-target estimators onto the ground plane. Colors correspond to the cameras IDs (black for view 1 and gray for view 2). For each view we want to associate at most one of these "observations" to particles on the ground.

Algorithm 1 Thread description for local tracker t^j_k in view j

Initialize t^j_k when target is detected and form the corresponding set of particles.

while t^j_k is not lost **do**
Prediction. Predict target position by using the prior of equation 1 and sampling particles from the previous set of particles.
Correction. Update particle weights by multiplying them by the likelihood in equation (2).
Evaluate tracking quality. Evaluate the tracker quality based on the unnormalized weights and updates the "lost" flag.
Normalization. Normalize particle weights.
Resampling. Based on the particle weights, perform particles resampling.
end while.

They are many approaches to solve the association problem (Collins, 2012). In order to perform association, is proposed a per-particle approach: each tracker T_k on the ground plane (figure 1) is modeled by a set of L particles, p^l_k, for $l = 1..L$. The local trackers in each view j, t^j_k provide a distribution of positions on

the ground plane through the geometric mapping of its particles on the plane. Sum up these distributions at this level, through their first two moments (mean and covariance).

$$(\tilde{\mathbf{X}}^j_{k,\tau}, \sum\nolimits^j_{k,\tau}). \quad (3)$$

For each particle and for each view, we then have to decide which of the local tracker estimate, if any, they will take as a source of observation. Let us note the particles $p^l_k = (\tilde{\mathbf{X}}^l_k, \omega^l_k)$, where $\tilde{\mathbf{X}}^l_k$ is the candidate state and ω^l_k its corresponding weight. All the remaining processes presented here are done in parallel for all k and all l. Consider one of these particles $p=(\tilde{\mathbf{X}}, \omega)$, and disregard for the moment the tracker to which it is associated or its particle index. The data association problem consists then in choosing one of the local trackers, on the base of Mahalanobis distance defined by equation (4):

$$(\tilde{\mathbf{X}} - \tilde{\mathbf{X}}^j_{k,\tau})^T (\mathrm{S}^j_{k,\tau})^{-1} (\tilde{\mathbf{X}} - \tilde{\mathbf{X}}^j_{k,\tau}). \quad (4)$$

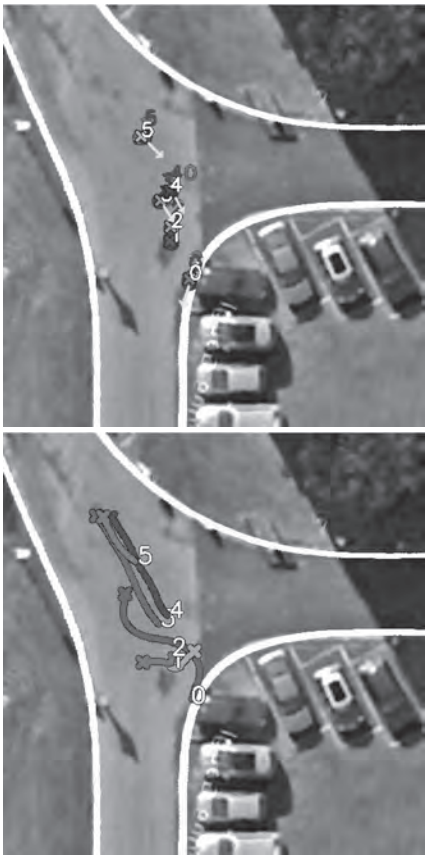

Figure 4. In (a), the set of "global" trackers and the associated observations, *i.e.* the reprojections of the local trackers means. The white arrows represent the estimated velocities. In (b), the constructed trajectories of a few targets in the ground plane.

The idea is then to choose for each view, the local tracker k that minimizes that Mahalanobis distance whenever this minimal distance passes under a given threshold. This idea is illustrated in figure 5. As the number of potential candidates is large, because the number of particles itself is typically of a few hundred particles, and as these operations are independent, we chose to implement it on GPGPU in the CUDA framework (NVIDIA CUDA Programming Guide 3.2, 2011).

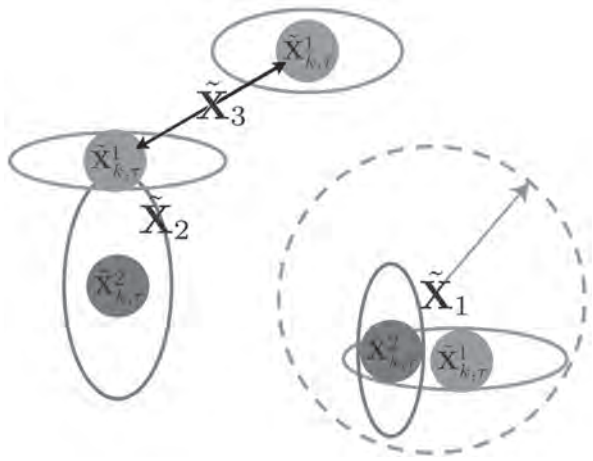

Figure 5. Data association problem. Association between particles $\tilde{\mathrm{X}}$ and projected estimation $\tilde{\mathrm{X}}^j_{k,\tau}$ according to Mahalanobis distance (black line). We only considered the elements with a distance under a threshold (orange circle) for association.

The entire process of data association and global tracking is done with the help of this framework. To speed up memory operation, we use the option of allocated memory (*cudaHostAllocMapped*) was used, which allows us to use RAM instead of GPU memory thus avoiding spending too much time in copying all the data.

The prediction step is performed first (the same as the local trackers one) by applying equation (1) in each particle in an independent thread. Then for all particles p the Mahalanobis distance is estimated and only those under the threshold and with minimum distance are kept. Note that this threshold has been determined empirically (*i.e.* 2 m). After that all distances are added to form a matrix of weights between observation of view j and global trackers. With this matrix the association with lower cost using the Hungarian algorithm is found. This procedure is repeated for all views.

Once the association problem is solved, we update the weight ω of all particles with its Mahalanobis distance is updated again. The normalization step is also done in parallel, by using the reduction technique to add all weights and divide each of them, on a per-thread basis.

RESULTS

We have tested our multiple-view tracking algorithm on the widely known PETS 2009 dataset (Performance Evaluation of Tracking and Surveillance 2009), which serves as a common benchmark in the area of tracking in video surveillance applications. The video sequences of this dataset has a resolution of 768 × 576 pixels. They are synchronized, according to the creators PETS 2009, but inconsistencies can be observed in some views. Although there are additional views available (8 in total), only the two most informative views were used. In figure 6, a few examples about the results are presented, by running the algorithm along a 795-frame video sequence across these two views. This sequence is challenging, as it is medium densely crowded, and several occlusions occur, therefore tracking in a single view would be difficult. The algorithm is applied here by fusing the local trackers on 2 views.

The two first lines depict three tracking frames on view 1 and 2, respectively. The last line includes the results of global trackers, integrating the re-projections of local trackers as observations. As it can be noted in fig. 6(g-i), local trackers keep a precise estimation of the pedestrian's trajectories, and as can be noted in fig. 6(g-i), reprojected estimates correspond quite well on the ground plane estimation.

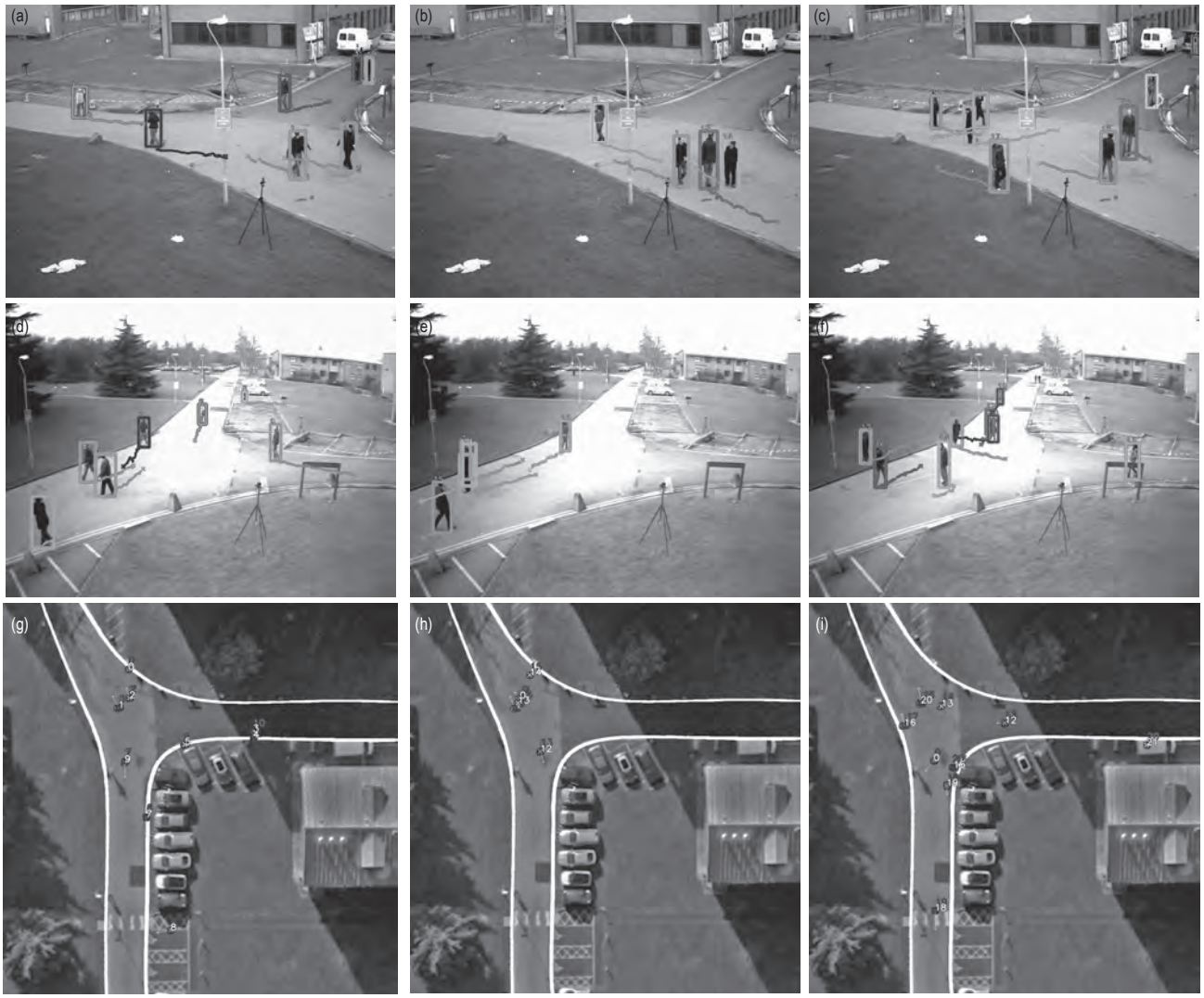

Figure 6. Some tracking results: the first two lines depict the local trackers on three frames, on two overlapping views. The third line provides the results of global trackers, integrating the reprojections of local trackers as observations.

Last, in figure 7, the effect of the implementation on GPU is studied and compared to CPU, in a synthetic experiment consisting of one tracker following a point in a plane (with additive Gaussian noise) and real data coming from PETS 2009, while varying the number of particles (in both plots, the horizontal axis is the log of the number of particles). The specifications of the CPU and the GPU were the following ones:

- CPU: 2.26 GHz Intel Core 2 Duo, with 8 GB DDR3 memory at 1 067 MHz.
- GPU: NVIDIA GeForce 9400 M 256 MB (16 CUDA cores).

As can be observed on the left, tracking errors are quite similar in both versions, even when they are a bit higher in the GPU case, for small numbers of particles. On the center and on the right, dramatic improvements in terms of computational times can be observed: With CPU, the time cost, which is linear in the number of particles, is reduced by a large constant factor, which is important when considering real-time application of the particle filtering framework with unknown data association. Indeed, with unknown data association, the real state space is in fact very large, and requires large numbers of particles, which typically makes CPU-based implementations quite ineffective.

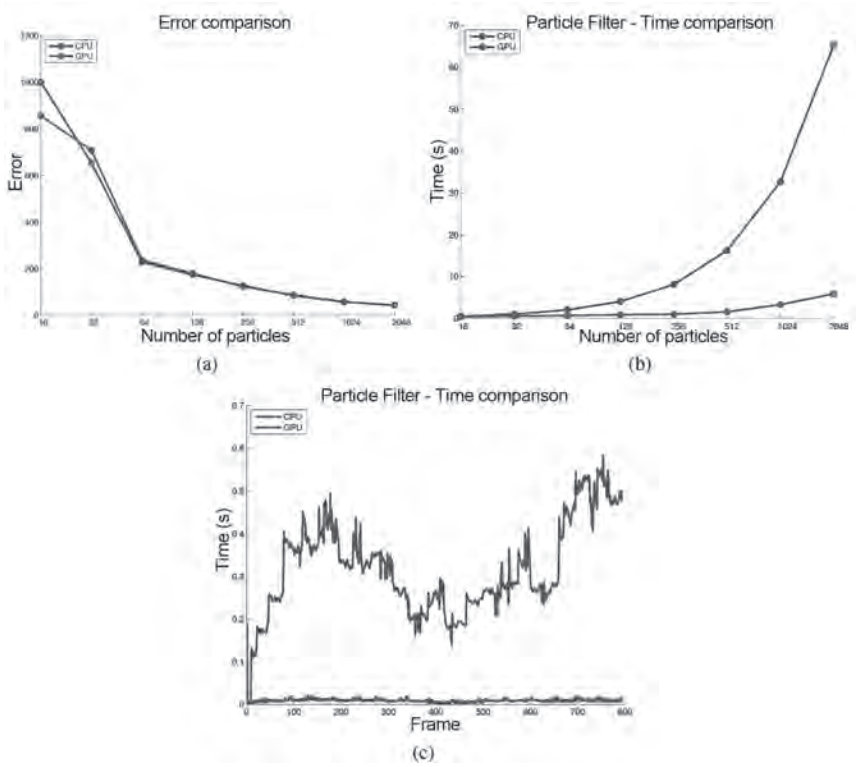

Figure 7. Performance comparison on the data association problem: CPU *vs.* GPU, in terms of (a) error *vs.* number of particles, (b) computation time *vs.* number of particles (a-b with synthetic data) and (c) time taken to process each frame of PETS 2009 data.

DISCUSSION

A particle filter-based multiple view tracker implementation that relies on particle filters at two levels, and exploits two types of parallelizing approaches is proposed: At the local level (on raw video sequences) multiple threads are run for each video and each detected target, which updates the estimates on the position and velocity of the target; at the higher level, another particle filter is run which fuses the information from local trackers into the ground plane stage; data association, *i.e.* the selection, among all re-reprojected local trackers, of the "closest" one to global trackers, is done by parallelizing all Mahalanobis distances computations through GPGPU with CUDA.

This approach has shown to be functional on moderately dense video sequences, and, above all, computational time improvements justify the use of GPGPU at the data association stage.

REFERENCES

Berclaz, J., Fleuret, F., Turetken, E. & Fua, P. (2011). Multiple Object Tracking Using K-Shortest Paths Optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 33*(9), 1806-1819.

Black, J. & Ellis. T. (2002). Multi-camera image measurement and correspondence. *Measurement, 32*(1), 61-71.

Collins, R. T. (2012). Multitarget data association with higher-order motion models. *IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), 1744-1751.

Das, A. & Kale, N. Vaswani. (2012). Particle filter with a mode tracker for visual tracking across illumination changes. *IEEE Transactions on Image Processing, 21*(4), 2340-2346.

Du, W., Hayet, J., Verly, J. & Piater, J. (2009). Ground-target tracking in multiple cameras using collaborative particle filters and principal axis-based integration. *IPSJ Transactions on Computer Vision and Applications, 1*(0), 58-71.

Hernandez–Lopez, F. & Rivera, M. (2010). Binary Segmentation of Video Sequences in Real Time. *Proceedings of the Ninth Mexican International Conference on Artificial Intelligence*, Pachuca, México, 163-168.

Intel Thread Building Blocks (TBB). (2012). Recuperado de: http://threadingbuildingblocks.org. Fecha de consulta, 6 de febrero de 2012.

Liem, M. & Gavrila, D. (September, 2009). Multi-person tracking with overlapping cameras in complex, dynamic environments. In A. Cavallaro, S. Prince & D. Alexander (Eds.), *Proceedings of the British Machine Conference,* (pp. 87.1-87.10). London, England: BMVA press. doi: 10.5244/c.23.87

NVIDIA CUDA Programming Guide 3.2, (2011). Recuperado de: http://developer.download.nvidia.com/compute/cuda/3 2/toolkit/docs/CUDA C Programming Guide.pdf. Fecha de consulta, 12 de abril de 2011.

Pérez, P., Vermaak, J. & Blake, A. (2004). Data fusion for visual tracking with particles. *Proceeding of the IEEE, 92*(3), 495-513.

Performance Evaluation of Tracking and Surveillance (PETS). (2009). Recuperado de: http://www.cvg.rdg.ac.uk/PETS2009/a.html. Fecha de consulta, 9 de enero de 2012.

Qu, W., Schonfeld, D. & Mohamed, M. (2007). Distributed bayesian multiple-target tracking in crowded environments using multiple collaborative cameras, *EURASIP Journal on Advanced Signal Processing* 2007, 1-15. doi: 10.1155/2007/38373

Wang, X. (2013). Intelligent multi-camera video surveillance: A review. *Pattern Recognition Letters, 34*(1), 3-19.

Yao, J. & Odobez, J. (2008). Multi-camera 3d person tracking with particle filter in a surveillance environment. *16th European Signal Processing Conference* (*EUSIPCO*).