Real Time Motion Estimation using a Neural Architecture Implemented on GPUs

Jose Garcia-Rodriguez Sergio Orts-Escolano Anastassia Angelopoulou
Alexandra Psarrou Jorge Azorin-Lopez Juan Manuel Garcia-Chamizo

Abstract

This work describes a neural network based architecture that represents and estimates object motion in videos. This architecture addresses multiple computer vision tasks such as image segmentation, object representation or characterization, motion analysis and tracking. The use of a neural network architecture allows for the simultaneous estimation of global and local motion and the representation of deformable objects. This architecture also avoids the problem of finding corresponding features while tracking moving objects. Due to the parallel nature of neural networks, the architecture has been implemented on GPUs that allows the system to meet a set of requirements such as: time constraints management, robustness, high processing speed and re-configurability. Experiments are presented that demonstrate the validity of our architecture to solve problems of mobile agents tracking and motion analysis.

1 Introduction

Algorithms for estimating and characterizing motion in video sequences are among the most widely used in computer vision. This need is sparked by the number of applications that require the fast and exact estimation of object motion. Such applications include the estimation of ego-motion for robot and autonomous vehicle navigation and the detection and tracking of people or vehicles for surveillance applications. Many other applications exist as well, including video compression and ubiquitous user interface design.

A sparse motion field can be computed by identifying a pair of points that correspond in two consecutive frames. The points used must be distinguished in some way so that they can be identified and located in both images. Detecting corner points or points of high interest should work. Alternatively, centroids of persistent moving regions from segmented color images might be used. Estimation of motion is also possible using higher level information such as corners or borders [1]. Viola *et al* [2] analyzes temporal differences between shifted blocks of rectangle filters with good results in low quality low resolution images. A large number of techniques have been proposed to analyze motion. In [3] a review of direct methods based on pixels can be found, while[4] reviews feature based methods.

A different approach analyzes motion following an optical flow computation, where a very small time distance between consecutive images is required, and no significant change occurs between two consecutive images. Optical flow computation results in motion direction and velocity estimation at image points determining a motion field. An early example of a widely used image registration algorithm is the patch-based translational alignment technique developed by Lucas and Kanade [5]. Several works have modified this method in different aspects improving its accuracy and accelerating its processing [6]. The optical flow analysis method can be applied only if the intervals between image acquisitions are very short. Motion detection based on correspondence of interest points works for inter-frame time intervals that cannot be considered small enough. Detection of corresponding object points in subsequent images is a fundamental part of this method, if feature correspondences are known, velocity fields can easily be constructed. The first step is to find in all images points of interest such as borders or corners that can be tracked over time. Point detection is followed by a matching process to find correspondences between these points. In Barron [7], a detailed evaluation of different optical flow methods can be found. In recent years, several improvements have been proposed to the classical optical flow estimators. Some of them proposed bioinspired optical flow VLSI implementations on embedded hardware [8, 9], FPGAs [10, 11, 12] or GPUs [13]. Due to temporal restrictions in most applications, some of the methods proposed relaxed versions of the algorithm to accelerate its processing [14, 15, 16, 16].

Although the understanding of issues involved in the computation of motion has significantly increased in the last decades, we are still far from a generic, robust, real-time motion estimation algorithm. The selection of best motion estimator is still highly dependent on the application [17, 18]. However, the parallelization of several computer vision techniques and algorithms to implement them on the GPU architecture reduces the computational cost of motion analysis and estimation algorithms.

Visual tracking is a very active research field related to motion analysis. The process of visual tracking in dynamic scenes often includes steps for modeling the environment, motion detection, classification of moving objects, tracking and recognition of actions developed. Most of the work is focused on tracking people or vehicles with a large number of potential applications such as: controlling access to special areas, identification of people, traffic analysis, anomaly detection and management alarms or interactive monitoring using multiple cameras [19].

Some visual tracking systems have marked important milestones. The visual tracking system in real-time W4 [20] uses a combination of analysis of shapes and tracking, building models of appearance to detect and track groups of people and monitor their behaviors even in the presence of occlusion and in outdoor environments. This system uses a single camera and a grayscale sensor. The system Pfinder [21] is used to retrieve a three-dimensional description of a person in a large space. It follows a single person without occlusions in complex scenes, and has been used in various applications. Another system to track a single person, the TI [22], detects moving objects in indoor scenes using motion detection, tracking is performed using first order prediction, and recognition is achieved by applying predicates to a behavior graph formed by matching objects links in successive frames. This system does not support small movements of objects in the background. The CMU system [23] can monitor activity on a wide area using multiple networked cameras. It can detect and track multiple people and vehicles in complex scenes and monitor their activities for long periods of time. Recognition of actions based on motion analysis has been extensively investigated [24, 25]. The analysis of trajectories is one of the main problems in understanding actions [26]. Relevant works on tracking objects can be found in [27, 28, 29] among others. Moreover, the majority of visual tracking systems depend on the use of knowledge about the scenes where the objects move in a predefined manner [30, 31, 25, 32].

Neural networks have been extensively used to represent objects in scenes and estimate their motion. In particular, there are several works that use the self-organizing models for the representation and tracking of objects. Fritzke [33] proposed a variation of the original Growing Neural Gas (GNG) model [34] to map non–stationary distributions that Frezza and Buet [35] apply to the representation and tracking of people. In [36], amendments to self-organizing models for the characterization of the movement are proposed. From the works cited, only Frezza and Buet [35] represent both local and global motion. However, there is no consideration of time constraints, and the algorithm does not exploit the knowledge acquired in previous frames for the purpose of segmentation or prediction. In addition the structure of the neural network is not used to solve the feature correspondence problem through the frames.

Considering the work in the area and previous studies about the representation capabilities of self-growing neural models [34], we propose a neural architecture capable of identifying areas of interest and representing the morphology of entities in the scene, as well as analyzing the evolution of these entities over time to estimate their motion. We propose the representation of the entities through a flexible model able to characterize morphological and positional changes of these over the image sequence. The representation should identify entities or mobile agents over time and establish feature correspondence through the different observations. This should allow the estimation of motion based on the interpretation of the dynamics of the representation model. It has been demonstrated that using architectures based on Fritzke's neural network, Growing Neural Gas (GNG) [37], can be applied on problems with the time restrictions such as tracking objects, with the ability to process sequences of images fast thus offering a good quality of representation that can be refined very quickly depending on the time available.

In order to accelerate the neural network learning algorithm, a redesign of the sequential algorithm executed onto the CPU to exploit the parallelism offered by the GPU has been carried out. Current GPUs have a large number of processors that can be used for general purpose computing. The GPU is specifically appropriate to solve computationally intensive problems that can be expressed as data parallel computations [38, 39]. However, implementation on GPU requires the redesign of the algorithms focused and adapted to its architecture. In addition, the programming of these devices has a number of constraints such as the need for high occupancy in each processor in order to hide latencies produced by memory access, management and synchronization of different threads running simultaneously, the proper use of the hierarchy of memories, and other considerations. Researchers have already successfully applied GPU computing to problems that were traditionally addressed by the CPU [38, 40]. The GPU implementation used in this work is based on NVIDIA's CUDA architecture [41], which is supported by most current NVIDIA graphics chips. Supercomputers that currently lead the world ranking combine the use of a large number of CPUs with a high number of GPUs.

The remainder of the paper is organized as follows: section 2 provides a description of the learning algorithm of the GNG with its accelerated version and presents the concept of topology preservation. In section 3 an explanation on how self-growing models can be adapted to represent and track 2D objects from image sequences is given. Section 4 presents the implementation of the system on GPGPU architectures, including some experiments, followed by our major conclusions and further work.

2 Neural Architecture to Represent and Track Objects

From the Neural Gas model [42] and Growing Cell Structures [43], Fritzke developed the Growing Neural Gas model [34], with no predefined topology of a union between neurons, which from an initial number of neurons new ones are added. Previous work [35], demonstrates the validity of this model to represent objects in images with its own structure and its capacity to preserve the input space topology. A new version of the GNG model called GNG-Seq has been created to manage image sequences under time constraints by taking advantage the GNG structure representation, obtained in previous frames, and by considering that at video frequency the slightly motion of the mobile agents present in the frames can be modeled. It is only necessary to relocate neural network structure without necessity to add or delete neurons. Moreover, the stable neural structure permits to use the neurons reference vectors as features to track in the video sequence. This fact allows us to solve one of the basic problems in tracking: features correspondence through time.

2.1 Growing Neural Gas Learning Algorithm

The Growing Neural Gas (GNG) [34] is an incremental neural model able to learn the topological relations of a given set of input patterns by means of competitive hebbian learning. Unlike other methods, the incremental character of this model, avoids the necessity to previously specify the network size. On the contrary, from a minimal network size, a growth process takes place, where new neurons are inserted successively using a particular type of vector quantization [42].

To determine where to insert new neurons, local error measures are gathered during the adaptation process and each new unit is inserted near the neuron which has the highest accumulated error. At each adaptation step a connection between the winner and the second-nearest neuron is created as dictated by the competitive hebbian learning algorithm. This is continued until an ending condition is fulfilled. In addition, in the GNG network the learning parameters are constant in time, in contrast to other methods whose learning is based on decaying parameters. In the remaining of this section we describe the growing neural gas algorithm. The network is specified as:

- A set A of nodes (neurons). Each neuron $c \in A$ has its associated reference vector $w_c \in \mathbb{R}^d$. The reference vectors are regarded as positions in the input space of their corresponding neurons.
- A set of edges (connections) between pairs of neurons. These connections are not weighted and its purpose is to define the topological structure. An edge aging scheme is used to remove connections that are invalid due to the motion of the neuron during the adaptation process.

The GNG learning algorithm to map the network to the input manifold is as follows:

1. Start with two neurons a and b at random positions w_a and w_b in $\mathbb{R}^d$.
2. Generate at random an input pattern ξ according to the data distribution $P(\xi)$ of each input pattern.
3. Find the nearest neuron (winner neuron) s_1 and the second nearest s_2.
4. Increase the age of all the edges emanating from s_1.
5. Add the squared distance between the input signal and the winner neuron to a counter error of s_1 such as:
$$\triangle error(s_1) = \|w_{s_1} - \xi\|^2 \tag{1}$$
6. Move the winner neuron s_1 and its topological neighbors (neurons connected to s_1) towards ξ by a learning step ϵ_w and ϵ_n, respectively, of the total distance:
$$\triangle w_{s_1} = \epsilon_w(\xi - w_{s_1}) \tag{2}$$
$$\triangle w_{s_n} = \epsilon_n(\xi - w_{s_n}) \tag{3}$$
For all direct neighbors n of s_1.
7. If s_1 and s_2 are connected by an edge, set the age of this edge to 0. If it does not exist, create it.
8. Remove the edges larger than a_{max}. If this results in isolated neurons (without emanating edges), remove them as well.
9. Every certain number λ of input patterns generated, insert a new neuron as follows:
   - Determine the neuron q with the maximum accumulated error.
   - Insert a new neuron r between q and its further neighbor f:
$$w_r = 0.5(w_q + w_f) \tag{4}$$

- Insert new edges connecting the neuron r with neurons q and f, removing the old edge between q and f.

10. Decrease the error variables of neurons q and f multiplying them with a consistent α. Initialize the error variable of r with the new value of the error variable of q and f.

11. Decrease all error variables by multiplying them with a constant γ.

12. If the stopping criterion is not yet achieved (in our case the stopping criterion is the number of neurons), go to step 2.

In summary, the adaptation of the network to the input space takes place in step 6. The insertion of connections (step 7) between the two closest neurons to the randomly generated input patterns establishes an induced Delaunay triangulation in the input space. The elimination of connections (step 8) eliminates the edges that no longer comprise the triangulation. This is made by eliminating the connections between neurons that no longer are next or that they have nearer neurons. Finally, the accumulated error (step 5) allows the identification of those zones in the input space where it is necessary to increase the number of neurons to improve the mapping (Figure 1).

2.2 GNG Representing Sequences GNG-Seq

To analyse the movement, for each image in a sequence, objects are tracked following the representation obtained with the neural network structure. That is, using the position or reference vector of neurons in the network as stable markers to follow. It is necessary to obtain a representation or graph that defines position and shape of the object at each input frame in the sequence.

One of the most advantageous features of the GNG is that it is not required to restart the learning of the network for each input frame in the sequence. Previous neural network structure, obtained from previous frames, is used as a starting point for new frames representation, provided that the sampling rate is sufficiently high. In this way, a prediction based on historical images and a small re-adjustment of the network, provides a new representation in a very short time (total learning time/ N), where total learning time is the complete learning algorithm that linearly depends on the number of input patterns λ and the number of neurons N. This provides a very high processing speed. This model of GNG for representation and processing of image sequences has been called GNG for sequences or GNG-Seq.

The process of tracking an object in each image is based on the following schedule:

1. Calculation of the transformation function Ψ to segment the object from the background based on information from previous frames stored in neural network structure.

2. Prediction of the new neurons reference vectors.

3. Re-adjustment of neurons reference vectors.

Figure 2 outlines the process to track objects, which differentiates the processing of the first frame, since no previous data is available and is needed to learn the complete network. In the second level, it predicts and updates the positions of the neurons (reference vectors) based on the available information from previous frames in the sequence that are stored in the neural network structure. In this way, the objects or agents can be segmented into the new frame, predict the new position and readjust the map based on information available from previous maps. The complete GNG algorithm is presented in Figure 1 and is used to obtain the representation for the first frame of the image sequence, by performing the full learning process. However, for next frames, the final positions (reference vectors) of the neurons obtained from the previous frame are used as starting points, doing only the reconfiguration model of the general algorithm, with no insertion or deletion of neurons, but moving neurons and deleting edges where necessary.

2.3 Topology Preservation

The final result of the self-organising or competitive learning process is closely related to the concept of Delaunay triangulation. The Voronoi region of a neuron consists of all points of the input space for what this is the winning neuron. Therefore, as a result of competitive learning a graph (neural network) is obtained whose vertices are the neurons of the network and whose edges are connections between them, which represents the Delaunay triangulation of the input space corresponding to the reference vectors of neurons in the network (Figure 3). This feature permits to represent the objects segmented in images preserving their original topology with a fast and robust representation model.

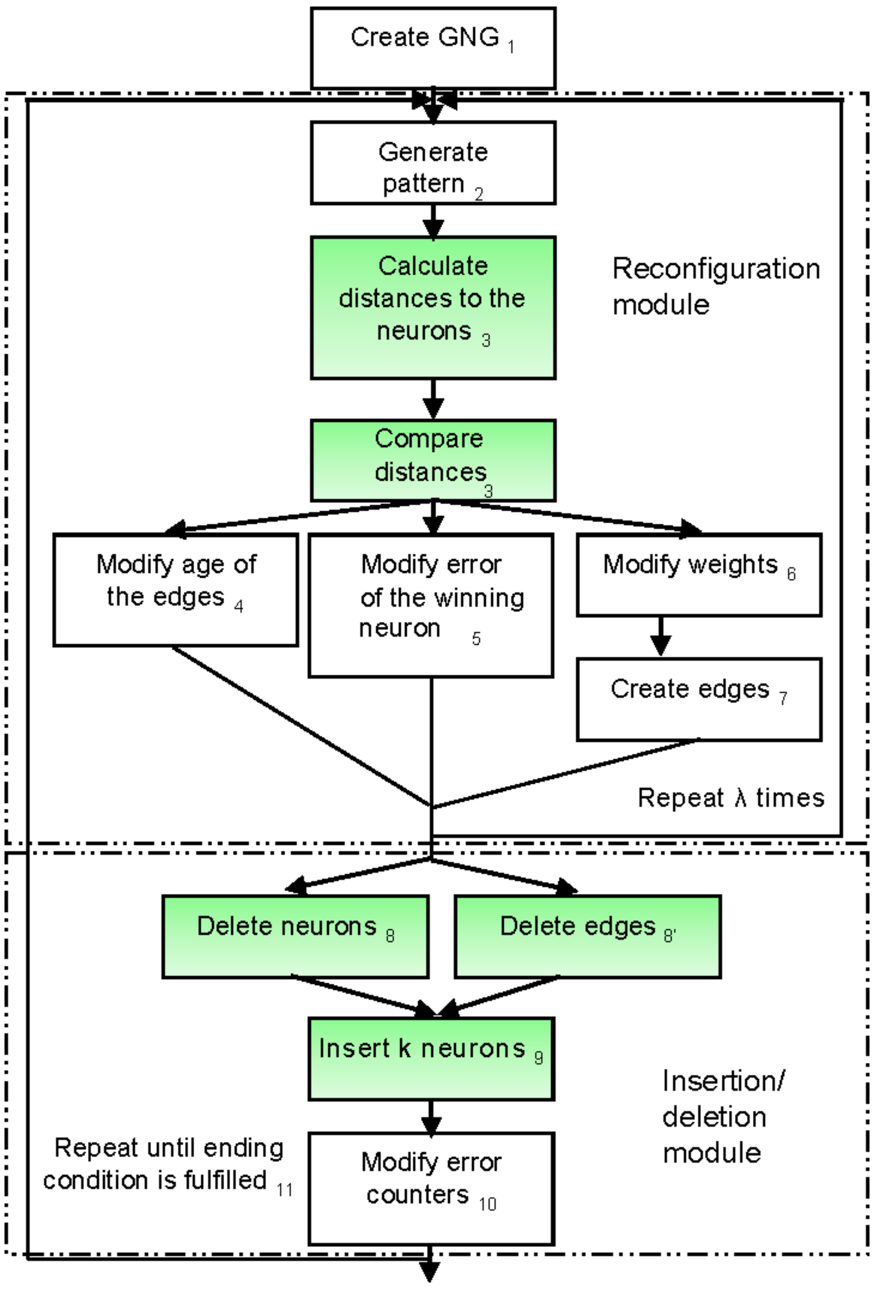

Figure 1: GNG Learning Algorithm.

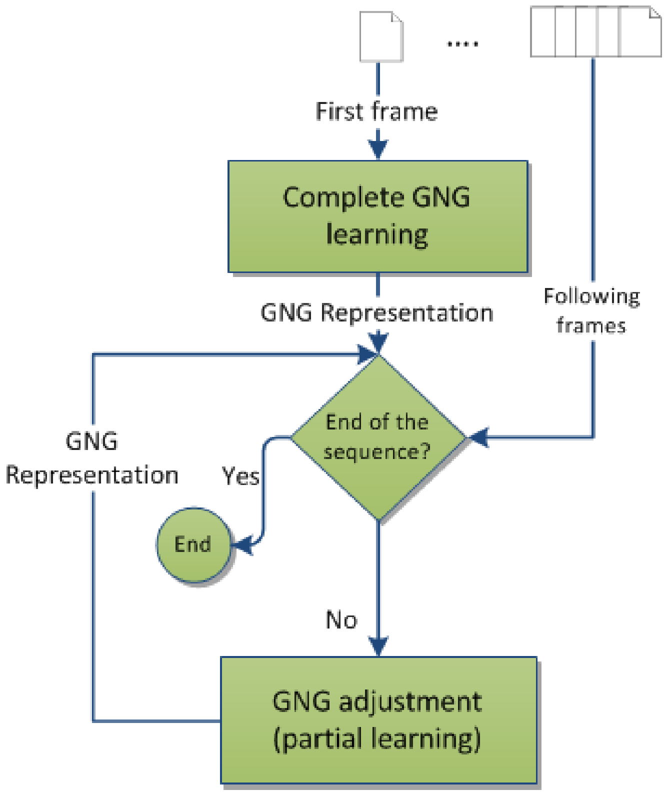

Figure 2: GNG-Seq flowchart

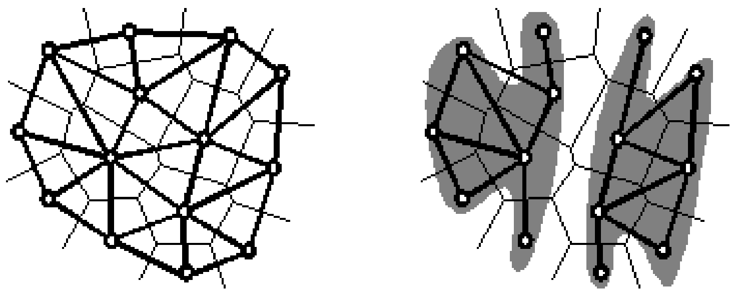

Figure 3: (a) Delaunay triangulation, (b) Induced Delaunay triangulation

3 Motion Estimation and Analysis with GNG-Seq

The ability of neural gases to preserve the topology will be employed in this work for the representation and tracking of objects. Identifying the points of the image that belong to objects allows the network to adapt its structure to this input subspace, obtaining an induced Delaunay triangulation of the object.

To analyse the movement, for each image in a sequence, objects are tracked following the representation obtained with the neural network structure. That is, using the position or reference vector of neurons in the network as stable markers to follow. It is necessary to obtain a representation graph for each of the instances, position, and shape of the object for all the images in the sequence.

The representation obtained with the neural network permits to estimate and represent the local and global motion described by multiple objects tracked in the scenes. That is, the system is able to estimate the motion of multiple targets due to the ability of the neural network to split its structure in different clusters that map to the different objects.

3.1 Motion Representation

Motion can be classified according to its perception. Common or global, and relative or local motion can be represented with the graph obtained from the neural network for every frame of the image sequence.

In the case of motion tracking in common or global mode, the analysis of the trajectory followed by an object can be done following the centroid of its representation throughout the sequence. This centroid can be calculated from the positions of the nodes in the graph that represent the object in each image. To track the movement in relative or local mode, changes in the position of each node with respect to the centroid of the object should be calculated for each frame. Following the trajectory of each node we can analyse and recognise the changes in the morphology of the object. One of the most important problems of tracking objects, the correspondence of features in a sequence of frames, can be intrinsically solved since the position of the neurons is known at any time without requiring any additional processing.

3.1.1 Common Motion

To analyse the common motion M_C, simply follow the centroid of the object based on the centroid of the neurons reference vectors that represent a single trajectory for the object. In this case, the M_C is regarded as the trajectory described by the centroid C_m of the graph representation GR obtained with the neural network structure over the frames 0 to f:

$$M_c = Tray_{c_m} = \{c_{m_{t_0}}, ..., c_{m_{t_f}}\} \tag{5}$$

3.1.2 Relative Motion

To analyse the relative movement of an object, the specific motion of individual neurons should be considered with respect to a particular point of the object, usually its centroid, and therefore will require specific tracking for each of the trajectories of the neurons that map to the object. Hence, the relative motion M_R is determined by the position changes of individual neurons with respect to the centroid C_m for every node i:

$$M_r = [Tray_i^{c_m}] \forall i \in A \tag{6}$$

where

$$Tray_i^{c_m} = \{w_{i_{t_0}} - c_{m_{t_0}}, ..., w_{i_{t_f}} - c_{m_{t_f}}\} \tag{7}$$

Where w_i is the reference vector of the node i, and C_m is the centroid of the graph obtained from the neural network that represents the image over the frames 0 to f.

3.2 Motion analysis

The analysis of motion in a sequence is done by tracking the individual objects or entities that appear in the scene. The analysis of the trajectory described by each object is used to interpret its movement. In this case the motion of an object is interpreted by the trajectories followed by each of the neurons GR:

$$M = [Tray_i], \forall i \in A \tag{8}$$

Where the trajectory is determined by the succession of positions (reference vectors) of individual neurons throughout the map:

$$Tray_i = \{w_{i_{t_0}}, ..., w_{i_{t_f}}\} \tag{9}$$

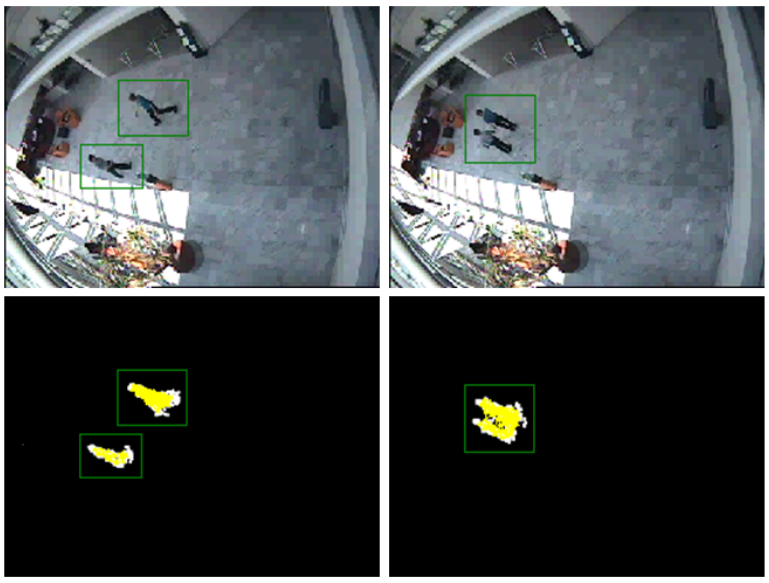

Figure 4: Examples of GNG Graph Representation

In some cases, to address the recognition of the movement a parameterization of the trajectories is performed. In [44] some proposals for parameterization can be found. Direct measures of similarity between trajectories, such as the modified Hausdorff distance [45] for comparison of trajectories and learning semantic scene models [46] are also used.

3.3 Tracking multiple objects in visual surveillance systems

There are several studies on the labeling and tracking of multiple objects, with some of them based on the trajectory [47] or the current state [48, 49, 50]. [51] explores the way in which they interact. There is an important field of study in related problems such as occlusion [52, 53]. The technique that we use to track multiple objects is based on the use of the GNG-Seq, since its fast algorithm separates the different objects present in the image. Once the objects in the image are separated, it is possible to identify groups of neurons and map each of them and follow them separately. These groups will be identified and labeled to use them as a reference and keep the correspondence between frames (Figure 4). The system has several advantages compared to other tracking systems:

- The graph obtained with the neural network structure permits the representation of local and global movement.
- The information stored in the structure of the neural network through the sequence permits the representation of motion and the analysis of the entities behaviour based on the trajectory followed by the neural network nodes.
- The correspondence features problem is solved using the structure of the neural network.
- The neural network implementation is highly parallel and suitable to be implemented on GPUs to accelerate it.

However some drawbacks should be considered:

- Quality of representation is highly dependent on the robustness of the segmentation results.

3.3.1 Merger and division

The ability of the growing neural gas network to break up to map all the input space is specially useful for objects that are divided. The network will eliminate unnecessary edges so that objects are represented independently by groups of the neurons. If the input space is unified again, the network adapts these changes by introducing new edges that reflect homogeneous input spaces. In all cases the neurons will remain without adding or deleting them so that objects or persons that appear together and split into groups after some time, can be identified and even tracked separately or together. This last feature is a great advantage of the representation model that gives the system great versatility in terms of track entities or groups of entities in video sequences. The merge of entities is represented as the union of the neural graph representation GR that mapped entities. That is, the necessary edges will be introduced to convert the isolated groups of neurons in only one big group. Figure 4 shows examples of neural graph representation.

$$GR_1 \bigcup GR_2 \bigcup ... \bigcup GR_n \Rightarrow GR_G \tag{10}$$

In the case of division of entities, the map that represents the group is split into different clusters. On the contrary to the merge process, edges among neurons will be deleted to create a number of clusters that represent the different entities in the scene.

$$GR_G \Rightarrow (GR_1, GR_2, ..., GR_n) \tag{11}$$

3.3.2 Occlusions

The modeling of individual objects during the tracking does not consider the interaction between multiple objects or interactions of these objects with the background. For instance, partial or total occlusion among different objects. The way in which the occlusions are handled in this work is to discard the frames where the objects are completely concealed by the background or by others objects. In each image, once an object is characterized and segmented, pixels belonging to each object are calculated. Frames are discarded, if percentage of pixels loss with respect to the average value calculated for the previous frames is very high and resumed the consideration of frames when the rate again becomes acceptable. In the case of partial occlusion with the background or between objects would be expected to adapt to the new transitive form since information from previous frames is available on the neural network structure.

3.4 Experimentation

To demonstrate the model capability to track multiple objects, some sequences from database CAVIAR (Context Aware Vision using Image-based Active Recognition) [54] have been used as input.The first section of video clips were filmed for the CAVIAR project with a wide angle camera lens in the entrance lobby of the INRIA Labs at Grenoble, France. The resolution is half-resolution PAL standard (384 x 288 pixels, 25 frames per second) and compressed using MPEG2. The file sizes are mostly between 6 and 12 MB, a few up to 21 MB. Figure 5 presents an example in which two people walk together and separate in a lobby. This example demonstrates the ability of the system to represent multiple objects, as well as its versatility to adapt to different divisions or merger of objects. Figures 5 and 6 describe the first frame in the top row, middle frame on the central row, and last frame in the bottom row from the sequence example. Showing the original image in the left column, segmented image and application of the network onto the image in central column and the trajectories described by the objects on the right one.

In Figure 5, we observe two people that start walking from distant points and then meet and walk together. The map starts with two clusters and then merges into a single one. In Figure 6, a group of people walk together and after a few meters split into groups. At first they are mapped by a single cluster but when they split, the map that represents them split into different clusters.

The system represents people with different clusters while walking separately and merged into a single cluster when they meet. This feature can be used for motion analysis systems. The definition of the path followed by the entities that appear in the image, depending on the path followed by the neurons that map the entity, allows us to study the behavior of those entities in time and give a semantic content to the evolution of the scene. By this representation will be possible to identify individuals who have been part of a group or have evolved separately since there are not deleted or added neurons and neurons involved in the representation of each entity remain stable over time. Different scenarios are stored in a database and can be analysed through measures to compare sets of points as the Hausdorff and extended or modified Hausdorff distances.

The fact that entities are represented by several neurons allows the study of deformations of these (local motion) and the interpretation of simple actions undertaken by such entities.

All of the above characteristics make the model of representation and analysis of motion in image sequences very powerful. Image sequences have more than 1000 frames with an area of 384x288 pixels and the processing speed obtained permits the system to work under time constraints. First frame takes more time to be processed since the complete learning algorithm should be used. However, for subsequent frames the speed is higher. Video acquisition time is not considered since this factor is highly dependent on the camera and communication bus employed. Based on a previous work [55], the number of neurons chosen is N of 1000 and the number of input patterns λ of 100. Other parameters have been also fixed based on our previous experience: $\epsilon_w = 0.1$, $\epsilon_n = 0.001$, $\alpha = 0.5$, $\gamma = 0.95$, $a_{max} = 250$. Examples of the paper experiments are available in www.dtic.ua.es/ jgarcia/experiments

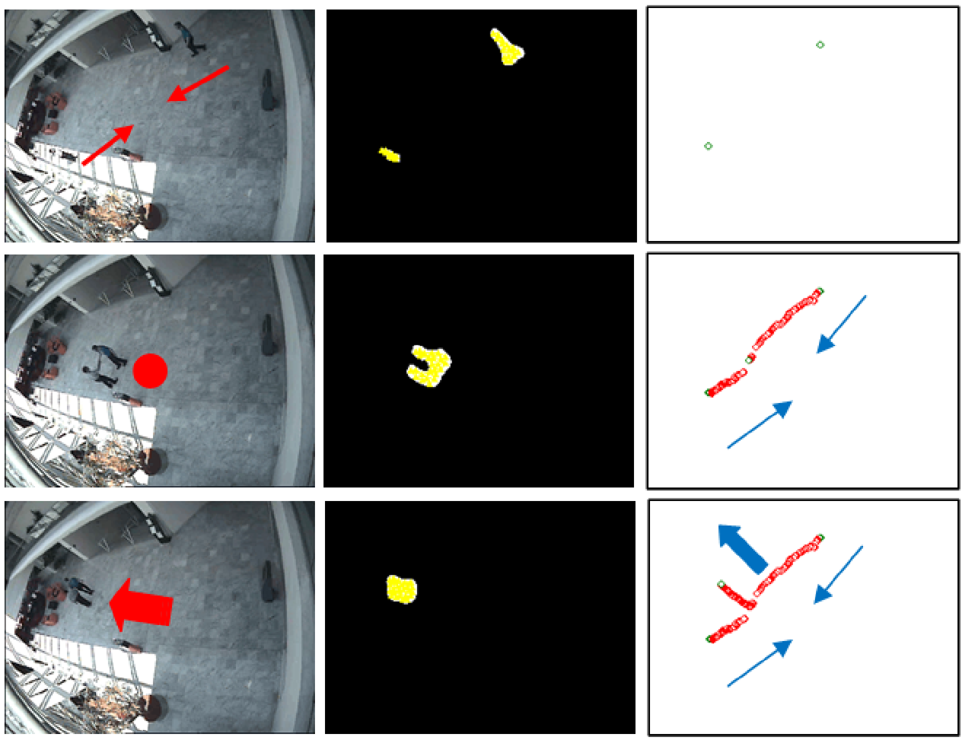

Figure 5: Motion Estimation with GNG Graph Representation. Merge

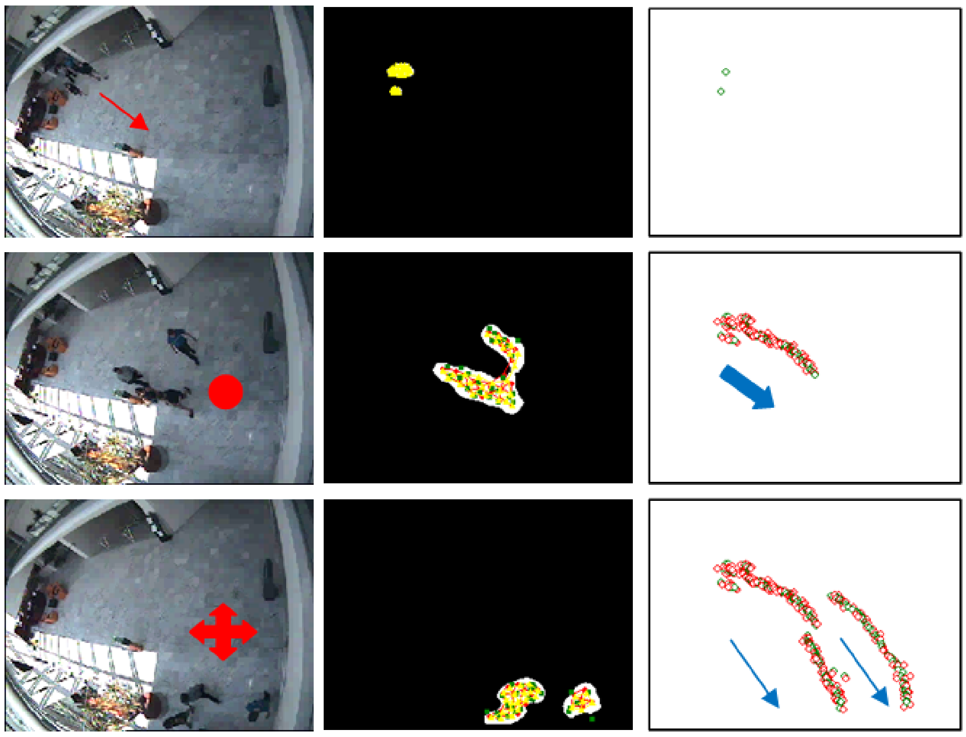

Figure 6: Motion Estimarion with GNG Graph Representation. Split

3.4.1 Tracking and motion estimation

In order to validate our proposal we performed some experiments to compute the trajectory error for different video sequences with people moving around the scene. We focused on four video sequences of the CAVIAR dataset that present a large range of movements and interactions between people. Figure 7 shows ground truth trajectories for different people in these four video sequences.

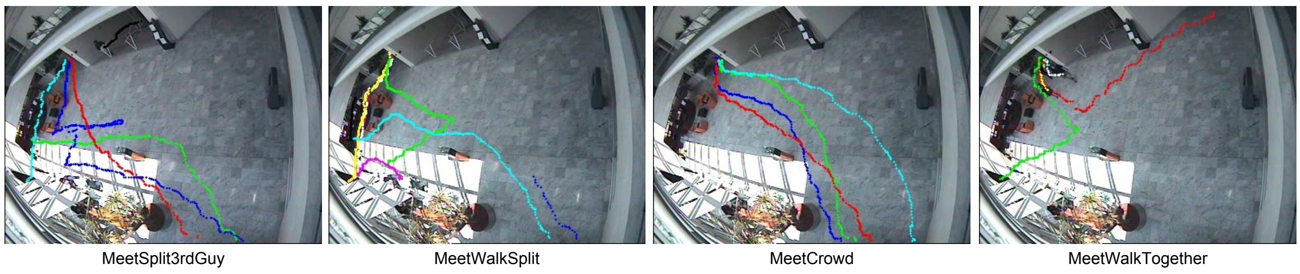

Figure 7: Ground truth trajectories of the selected video sequences from the CAVIAR dataset.

Since the CAVIAR dataset provides us with ground truth information about the trajectory of the objects and people in the scene, we calculated the Root Mean Square Error (RMSE) regard to ground truth information using the proposed method. We also compared our method with state-of-the-art Lucas-Kanade [5] method for tracking people in the scene. Lucas-Kanade tracking algorithm was manually initialized choosing keypoints over people representation in the scene. These keypoints were tracked over the sequence and the centroid of the estimated keypoint trajectories were used for comparison with the proposed method. Table 1 shows obtained RMSE for different trajectories in four video sequences. As it can be seen, the proposed method obtained a lower RMSE in most cases. This means that the estimated trajectory by using the GNG-based method is more accurate than the one obtained using the Lucas-Kanade method. Moreover, in some video sequences such as the MeetSplit3rdGuy, the Lucas-Kanade method was not able to track trajectories when two people walk, meet and then move to different directions, it failed tracking these kind of behaviours.

Video	Person	Lukas-Kanade	GNG-based
MeetSplit3rdGuy	1	119.108	7.08368
	2	17.3801	10.0998
	3	84.3251	16.3223
MeetWalkSplit	1	15.2826	7.5423
	2	17.4852	10.6566
MeetWalkTogether	1	10.7862	11.7721
	2	19.4851	13.8851
MeetCrowd	1	10.7769	14.0695
	2	12.3467	24.259
	3	32.7781	22.193
	4	15.2363	14.5812

Table 1: Computed Root Mean Squared Error (RMSE) for different trajectories regard to ground truth information. The proposed method has been validated in 4 video sequences with different number of people and trajectories. RMSE is expressed in pixels.

Figure 8 shows the behaviour introduced above. On the left of the figure, Lucas-Kanade (blue line) fails tracking the person (green line) and continues tracking another person when these two people meet together. On the right side, GNG-based method (blue line) succeeds tracking the person over its trajectory (green line).

In Figure 9, we show a visual example where the proposed method achieves a lower RMSE in the trajectory estimation (right side), whereas Lucas-Kanade method gets lost and the estimated trajectory has a higher error compared to the one obtained using GNG-based method.

Finally, we also performed some experiments using a dense optical flow estimation approach [56]. We were not able to obtain RMSE regard to the ground truth information since dense optical flow estimation approach is not able to obtain independent trajectories for each person moving in the scene. Dense optical flow estimation approach gives us motion estimation for the entire scene and it is not able to distinguish between trajectories

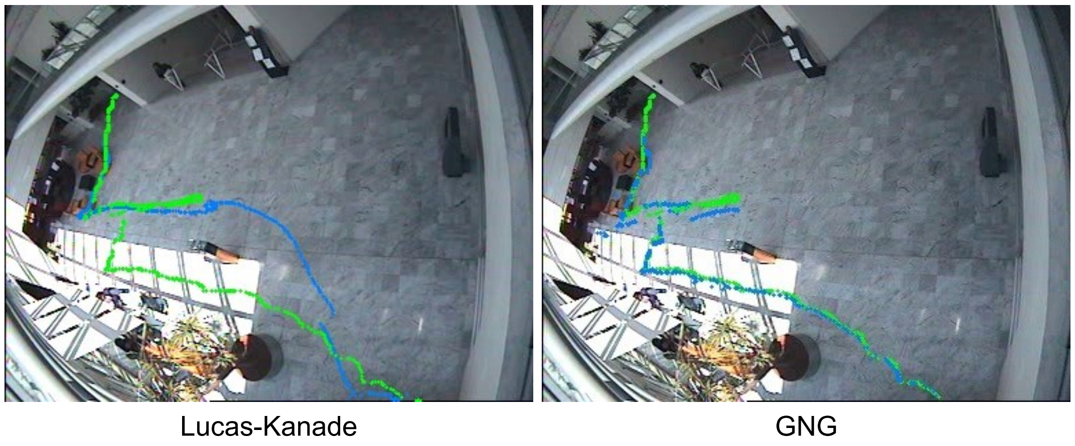

Figure 8: Estimated trajectory using GNG and Lucas-Kanade method in the MeetSplit3rdGuy video sequence. Left: Lucas-Kanade trajectory estimation fails due to pixel intensity similarities between different people moving around the scene. Right: GNG-based tracking method is able to correctly estimate the person trajectory. (Blue line: estimated trajectory. Green line: ground truth trajectory.)

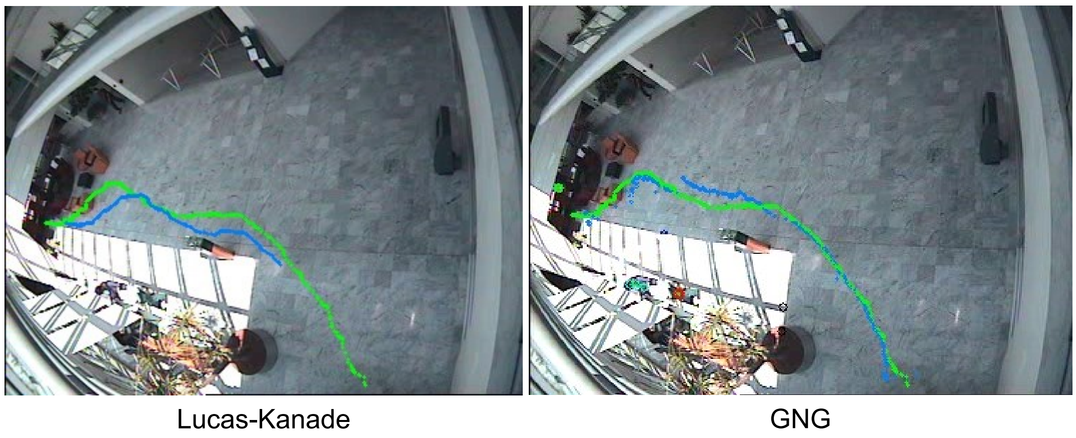

Figure 9: Estimated trajectory using GNG and Lucas-Kanade method in the WalkSplit video sequence. Left: Lucas-Kanade trajectory estimation. Right: GNG-based trajectory estimation. (Blue line: estimated trajectory. Green line: ground truth trajectory.)

performed by different people. Figure 14 shows motion estimation using a dense optical flow approach in different video sequences.

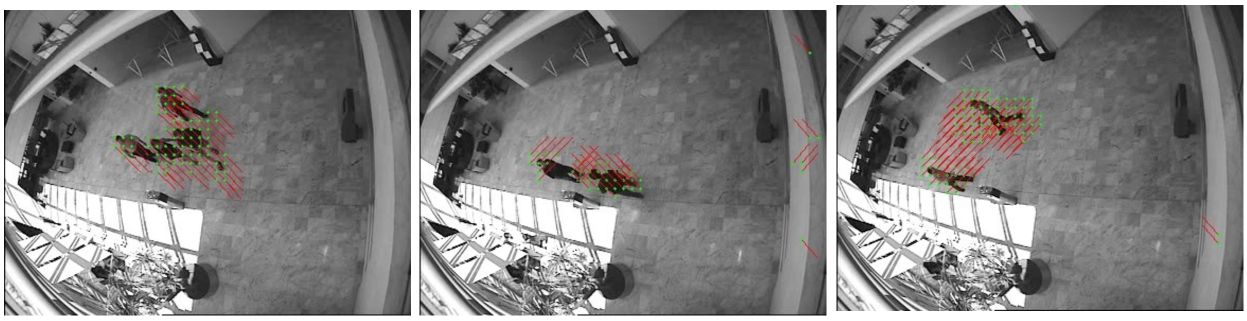

Figure 10: Dense optical flow estimation approach applied to CAVIAR dataset.

3.4.2 Discussion

From the experiments, we can conclude that the system is able to work under time constraints and be close to real time after processing, representing, and tracking multiple mobile agents in the first frame and estimating global and local objects motion. However, in these experiments the number of neurons and input patterns used in the neural network learning algorithm is low, less that 500 neurons. For more challenging scenes: with multiple targets, bigger images, high resolution images or even to process not only the moving objects but the whole scenario, or to scale the system to represent 3D data, it should be necessary to dramatically increase the number of neurons. In this case, the performance of the CPU version of the system will not be capable to represent and track thousands of neurons with time constraints. For that reason, we propose the parallel implementation of the neural model on a GPGPU architecture.

4 GPU Implementation

In this section we first introduce the GPGPU paradigm and apply it in order to parallelize and redesign the neural network learning algorithm. Once the algorithm has been redesigned and optimized, the motion estimation system based on the neural networks architecture is highly accelerated.

4.1 GPGPU Architecture

A CUDA compatible GPU is organized in a set of multiprocessors as shown in Figure 11 [41]. These multiprocessors called Streaming Multiprocessors (SMs) are highly parallel at thread level. However, the number of multiprocessors varies depending on the generation of the GPU. Each SM consists of a series of Streaming Processors (SPs) that share the control logic and cache memory. Each of these SPs can be launched in parallel with a huge amount of threads. For instance, GT200 graphics chips, with 240 SPs, are capable to perform a computing power of 1 teraflops, launching 1024 threads per SM, with a total of 30,000 threads. The current GPUs have up to 12 GBytes of DRAM, referenced in Figure 11 as global memory. The global memory is used and shared by all the multiprocessors but it has a high latency.

CUDA architecture reflects a SIMT model: single instruction, multiple threads. These threads are executed simultaneously working onto large data in parallel. Each of them runs a copy of the kernel (piece of code that is executed) on the GPU and uses local indexes to be identified. Threads are grouped into blocks to be executed. Each of these blocks is allocated on a single multiprocessor, enabling the execution of several blocks within a multiprocessor. The number of blocks that are executed depends on the resources available to the multiprocessor, scheduled by a system of priority queues.

Within each of these blocks, the threads are grouped into sets of 32 units to carry out fully parallel execution onto processors. Each set of 32 threads is called warp. In the architecture there are certain restrictions on the maximum number of blocks, warps and threads on each multiprocessor, but it varies depending on the generation and model of the graphics cards. In addition, these parameters are set for each execution of a kernel in order to ensure the maximum occupancy of hardware resources and obtain the best performance. The experiments section shows how to fit these parameters to execute our GPU implementation.

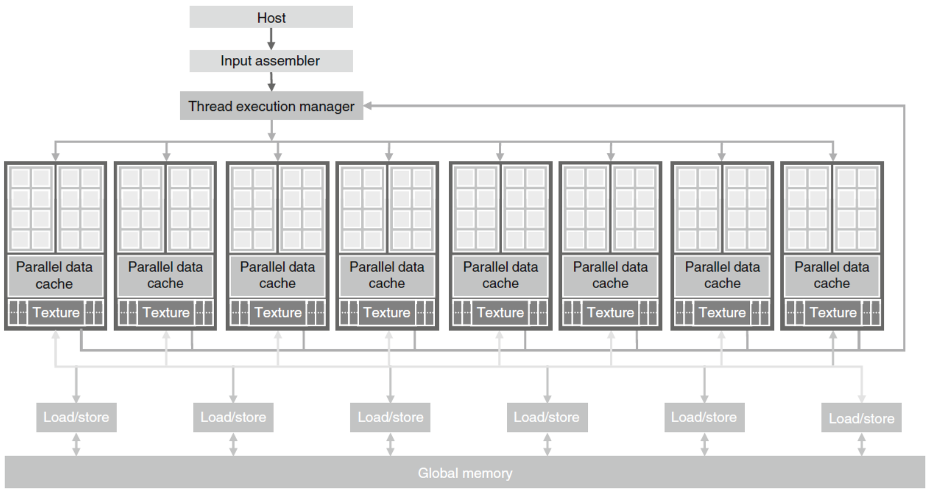

Figure 11: CUDA compatible GPU Architecture

CUDA architecture has also a memory hierarchy. Different types of memory can be found: constant, texture, global, shared and local registries. The shared memory is useful to implement caches. Texture and constant memory are used to reduce computational cost avoiding global memory access which has high latencies.

In recent years, a large number of applications have used GPUs to speed up processing of neural networks algorithms [57, 58, 59, 60, 61] applied to various computer vision problems such as: representation and tracking of objects in scenes [62], face representation and tracking [63] or pose estimation [64].

4.2 GPU implementation of GNG

The GNG learning algorithm has a high computational cost. For this reason, it is proposed to accelerate it using GPUs and taking advantage of the many-core architecture provided by these devices, as well as their parallelism at the instruction level. GPUs are specialized hardware for computationally intensive high-level parallelism that use a larger number of transistors to process data and fewer for flow control or management of the cache, compared to CPUs. We have used the architecture and set of programming tools (language, compiler, development environment, debugger, libraries, etc) provided by NVIDIA to exploit the parallelism of its hardware.

To accelerate the GNG algorithm on GPUs using CUDA, it has been necessary to redesign it so that it better suits the GPU architecture. Many of the operations performed in the GNG algorithm are parallelizable because they act on all the neurons of the network simultaneously. That is possible, because there is no direct dependence between neurons at operational level. However, there exists dependence in the adjustment of the network, which takes place at the end of each iteration and forces the synchronization of various parallel execution operations. Figure 1 shows the GNG algorithm steps that have been accelerated onto the GPU using kernels.

The accelerated version of GNG algorithm has been developed and tested on a machine with an Intel Core i3 540 3.07Ghz and a number of different CUDA capable devices. Table 1 shows different models that we have used and their features.

Device Model	Capability	SMs	cores per SM	Global Mem	Bandwidth Mem
Quadro 2000	2.1	4	192	1 GB	41.6 GB/s
GeForce GTX 480	2.0	15	480	1.5 GB	177.4 GB/s
Tesla C2070	2.0	14	448	6 GB	144 GB/s

Table 2: CUDA capable devices used in experiments

4.2.1 Euclidean distance calculation

The first stage of the algorithm that has been accelerated is the calculation of Euclidean distances performed in each iteration. This stage calculates the Euclidean distance between a random pattern and each of the neurons. This task may take place in parallel by running the calculation of each distance calculation onto as many threads as neurons the network contains. It is possible to calculate more than one distance per thread, but this is only efficient for large vectors where the number of blocks that are executed on the GPU is also very high.

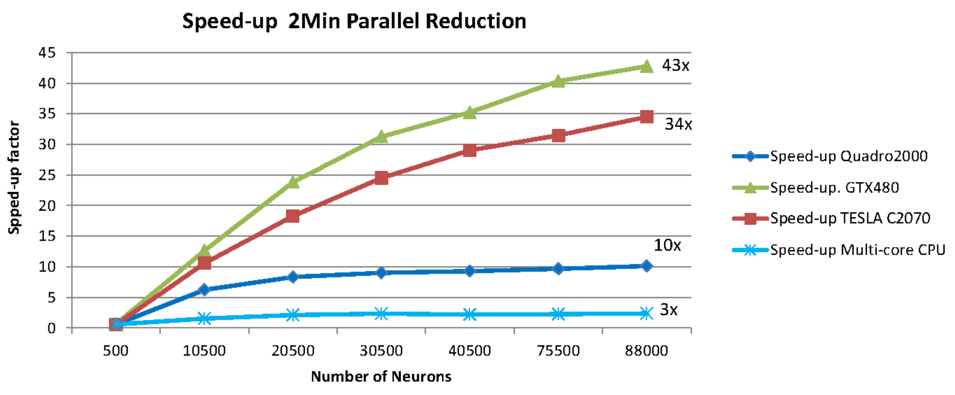

Figure 12: Parallel Reduction speedup with different devices

4.2.2 Parallel reduction

The second task parallelized is the search of the winning neuron: the neuron with the lower Euclidean distance to the pattern generated, and the second closest. For the search, we used the parallel reduction technique described in [35]. This technique accelerates operations such as the search for the minimum value in parallel in large data sets. For our work, we modified the original algorithm that we have called *2MinParallelReduction*, so that, with a single reduction it is not only obtained the minimum, but the two smallest values of the entire data set. Parallel reduction can be described as a binary tree where: for $log2(n)$ steps operated in parallel in sets of two elements, by applying an operation on these elements in parallel; at the end of the $log2(n)$ steps we obtained the final result of the operation onto a set of N elements.

We carried out experiments of *2MinParallelReduction* implementation with different graphics boards using 256 threads per block configuration for kernels launch. We obtained a speed-up factor up to 43x faster with respect to a single-core CPU and 40x faster with respect to multi-core CPU, in the task of taking adjustments of the network with a number of neurons close to 100k. As we can see in Figure 12 (bottom), the speed-up factor depends on the device on which we execute the algorithm and the number of cores it has. Figure 12 shows the evolution of the execution time in sequential reduction operation compared to the parallel version. It can also be appreciated how GPU implementation improves the acceleration provided by the parallel algorithm as the number of elements grows.

4.2.3 Other optimizations

To speed-up the remaining steps, we have followed the same strategy used during the first phase. Each thread is responsible to perform an operation on a neuron: check edges connections age and in the case that exceeded a certain threshold delete them; update local error of the neuron or adjusting neuron weights. In the stage of finding the neuron with maximum error, we followed the same strategy used in finding the winning neuron, but in this case, the reduction is seeking only the neuron with highest error. Regardless the parallelism of the algorithm, we have followed some good practices on the CUDA architecture to achieve better performance. For example, we used the constant memory to store the neural network parameters: ϵ_1 ,ϵ_2 , α , β, α_{max}. By storing these parameters in this memory, the access is faster than working with values stored in the global memory.

Our GNG implementation on GPU architecture is also limited by the memory bandwidth available. In the experiments section we show a specification report for each CUDA capable device used and its memory bandwidth. However, this bandwidth is only attainable under highly idealized memory access patterns. It does, however, provide us with an upper limit of memory performance. Although some memory access patterns, like moving data from the global memory into shared memories and registers, provide better coalesced access. To achieve the highest advantage of memory bandwidth, we used the shared memory within each multiprocessor. In this way shared memory acts as a cache to avoid frequent access to global memory in operations with neurons and allows threads to achieve coalesced reads when accessing neurons data. For instance, a GNG network composed of 20,000 neurons and auxiliary structures requires only 17 megabytes. Therefore, GPU implementation, in terms of size, does not present problems because currently GPU devices have enough memory to store it.

Memory transfers between CPU and GPU are the main bottleneck to obtain speed-up. These transfers have been avoided as much as possible. Initial versions of the algorithm failed to obtain performance over the CPU version because the complete neural network was copied from GPU memory to CPU memory and vice versa for each input pattern generated. This penalty, introduced due to the bottleneck of the transfer through the PCI-Express bus, was so high that the performance was not improved compared to the CPU version. After

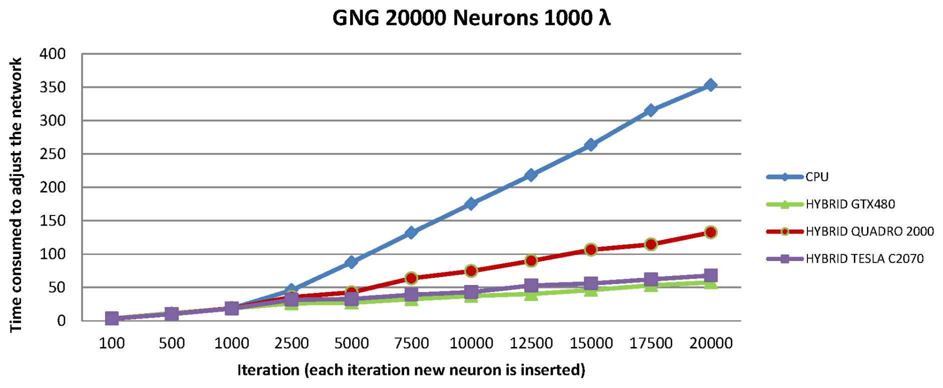

Figure 13: Hybrid version speedup with different devices

careful consideration about the flow of execution we decided to move the inner loop of pattern generation to the GPU, although some tasks are not parallelizable and had to be run on a single GPU thread.

4.2.4 GNG hybrid version

As we discussed in the previous experiments, GPU version has low performance in the first iterations of the learning algorithm, where the GPU cannot hide the latencies due to the small number of processing elements. To achieve even bigger acceleration of the GNG algorithm, we propose the use of the CPU in the first iterations of the algorithm, and then start processing data in the GPU only when there is an acceleration regarding CPU, thus achieving a bigger overall acceleration of the algorithm (see Figure 13). To determine the number of neurons necessary to start computing at GPU we have analyzed in detail the execution times for each new insertion, and concluded that each device, depending on its computing power starts being efficient at a different number of neurons. Following several tests, we have determined the threshold at which each device starts accelerating compared to the CPU version. As it can be seen in figure 9, threshold values for different devices are set to 1500, 1700, 2100 for GTX 480, Tesla C2070 and Quadro 2000 models respectively. The hybrid version is proposed as some applications need to operate under time constraints obtaining a solution of a specified quality within certain period of time. In cases when the objective is the disruption of learning due to the application requirements, it is important to insert the maximum number of neurons and perform the maximum number of adjustments to achieve the highest quality in a limited time. The hybrid version ensures a maximum performance in this kind of applications using the computational capabilities of the CPU or the GPU depending on the situation.

4.2.5 Rate of adjustments per second

We have performed several experiments where it is shown how the accelerated GNG version is not only capable to perform a complete learning cycle faster than CPU but also to perform more adjustments per second than the CPU implementation. For instance, after learning a network of 20000 neurons, we can perform 17 adjustments per second using the GPU while the CPU only gets 2.8 adjustments per second. This means that GPU implementation can obtain a good topological representation with time constraint. Figure 14 shows the different adjustments rate per second that performed by different GPU devices and CPU. It is also shown, how increasing the number of neurons in the CPU, it cannot handle a high rate of adjustments per second.

4.2.6 Discussion

From the experiments described above we can conclude that the number of threads per block that best fits in our implementation is 256 due to the following reasons: First, the amount of computation the algorithm performs in parallel. Second, the number of resources that each device has and finally the use that we have made of shared memories and registries. It is also demonstrated that in comparison to CPU implementation, the $2MinParallelReduction$ achieves a speed-up of more than 40x to find out a neuron at a minimum distance to the generated input pattern. Theoretical values obtained applying Amdahl's law and its comparison with real values obtained from the experiments indicates that GPGPU architecture has some implicit latencies: initialization time, data transfers time, memory access time, etc.

Experiments on the complete GNG algorithm showed that using the GPU, small networks under-utilize the device, since only one or a few multiprocessors are used. Our implementation has a better performance for large

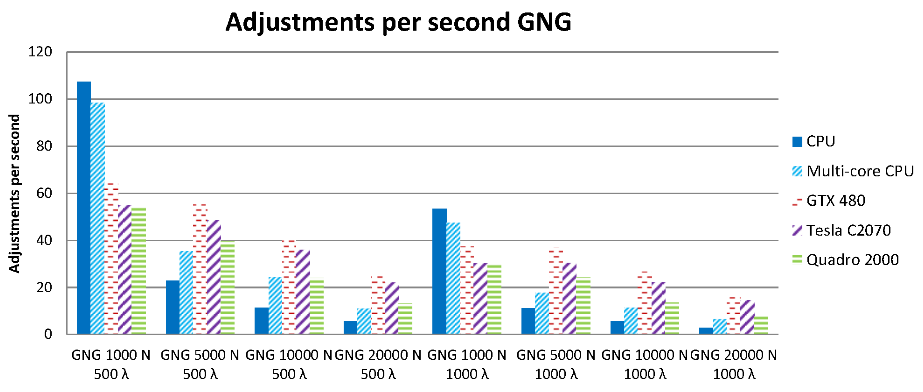

Figure 14: Rate of adjustments per second performed by different GPU devices and CPU

networks than for small ones. To get better results for small networks we propose a hybrid implementation. These results show that GNG learning with the proposed hybrid implementation achieves a speed-up 6 times higher than the single threaded CPU implementation.

Additionally, it is shown how our GPU implementation can process up to 17 adjustments of the network per second while single threaded CPU implementation only can manage 2.8, getting a speed-up factor of more than 6 times in the extreme situation of using 20000 neurons and 1000 input patterns.

Finally, we computed the MPps (MegaPixels per second) rate achieved by our proposal. Table 3 shows MPps rates for different number of neurons and λ patterns. It can be seen how the CPU version is able to manage large MPps rates for small number of neurons and λ patterns. However, for a number of neurons larger than $5,000$ the CPU version is not able to manage reasonable MPps rates whereas the GPU implementation obtains considerable higher rates. MPps rates where computed considering images resolution (384x288 pixels).

	Mpixels/sec CAVIAR				
	CPU	Quadro 2000	Tesla C2070	GTX 480	Multi-core CPU
GNG 1000 N 500 λ	11.88	6.02	6.09	7.14	10.88
GNG 5000 N 500 λ	2.53	4.38	5.35	6.10	3.90
GNG 10000 N 500 λ	1.26	2.67	3.97	4.50	2.68
GNG 20000 N 500 λ	0.62	1.46	2.43	2.80	1.21
GNG 1000 N 1000 λ	5.92	3.31	3.34	4.14	5.23
GNG 5000 N 1000 λ	1.24	2.67	3.37	4.08	1.95
GNG 10000 N 1000 λ	0.61	1.49	2.46	3.00	1.26
GNG 20000 N 1000 λ	0.31	0.83	1.60	1.89	0.72

Table 3: MegaPixels per second rates obtained for different number of neurons and λ patterns.

5 Conclusions and future work

In this work we presented a system based on GNG neural network capable of representing motion under time constraints. The proposed system incorporates mechanisms for prediction based on information stored within the network structure on the characteristics of objects such as shape or situation to anticipate certain operations such as segmentation and positioning of objects in subsequent frames. This provides for a more rapid adaptation to the objects in the image, restricting the areas of search and anticipating the new positions of objects.

Processing information on the neurons' position (reference vectors) through time is possible to construct the path followed by objects represented and interpret these. This evolution can be studied from global movement, using the centroids of the paths or from local movement, by studying the deformations of the object based on neural network structure changes. This is possible because the system does not restart the neural network every frame but only readjust the network structure starting from previous positions without inserting or deleting neurons. In this way the neurons are used as markers that define the stable form of objects.

The capabilities of the system for tracking and motion analysis have been demonstrated. The system automatically handles the mergers and divisions among entities that appear in the images and can detect and interpret the actions that are performed in video sequences. The GNG-Seq architecture enables to manage image sequences with time constraints but the system is limited and we have proposed the implementation of the model on a GPGPU architecture.

We identified the stages that employ more time in the learning algorithm and parallelize and redesign them to maximize the system performance. Since the GPU implementation improves the CPU one, only for a high number of neurons, a hybrid version has been designed that works on CPU and changes to GPU when the necessary number of neurons have been inserted. Experiments have been developed with different devices to demonstrate the validity of our system.

We can also conclude that although the understanding of issues involved in the computation of motion has significantly increased in the last years, we are still far from generic, robust, real-time motion estimation algorithm. The selection of the best motion estimator is still highly dependent on the application. However, the acceleration of several computer vision techniques and algorithms to fit them to the GPU architecture reduces the computational cost of motion analysis and estimation algorithms.

As a further work, we plan to improve the CPU version in some aspects such as segmentation and prediction. We also work in the refinement of the GPU version.

Acknowledgements

This work was partially funded by the Spanish Government DPI2013-40534-R grant and Valencian Government GV/2013/005 grant. Experiments were made possible with a generous donation of hardware from NVDIA.

References

[1] Wu, S.F., Kittler, J.: General motion estimation and segmentation (1990)

[2] Viola, P., Jones, M., Snow, D.: Detecting pedestrians using patterns of motion and appearance. In: Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on. (2003) 734–741 vol.2

[3] Irani, M., Anandan, P.: About direct methods. In: Proceedings of the International Workshop on Vision Algorithms: Theory and Practice. ICCV '99, London, UK, UK, Springer-Verlag (2000) 267–277

[4] Torr, P.H.S., Zisserman, A.: Feature based methods for structure and motion estimation. In: Proceedings of the International Workshop on Vision Algorithms: Theory and Practice. ICCV '99, London, UK, UK, Springer-Verlag (2000) 278–294

[5] Lucas, B.D., Kanade, T.: An iterative image registration technique with an application to stereo vision. In: Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2. IJCAI'81, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (1981) 674–679

[6] Baker, S., Matthews, I.: Lucas-kanade 20 years on: A unifying framework. Int. J. Comput. Vision **56**(3) (2004) 221–255

[7] Barron, J., Fleet, D., Beauchemin, S.: Performance of optical flow techniques. International Journal of Computer Vision **12**(1) (1994) 43–77

[8] Botella, G., Meyer-Base, U., Garcia, A.: Bio-inspired robust optical flow processor system for vlsi implementation. Electronics Letters **45**(25) (2009) 1304–1305

[9] Botella, G., Garcia, A., Rodriguez-Alvarez, M., Ros, E., Meyer-Baese, U., Molina, M.: Robust bioinspired architecture for optical-flow computation. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on **18**(4) (2010) 616–629

[10] Barranco, F., Tomasi, M., Diaz, J., Vanegas, M., Ros, E.: Parallel architecture for hierarchical optical flow estimation based on fpga. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on **20**(6) (2012) 1058–1067

[11] Diaz, J., Ros, E., Pelayo, F., Ortigosa, E., Mota, S.: Fpga-based real-time optical-flow system. Circuits and Systems for Video Technology, IEEE Transactions on **16**(2) (2006) 274–279

[12] Botella, G., Martín H., J.A., Santos, M., Meyer-Baese, U.: Fpga-based multimodal embedded sensor system integrating low- and mid-level vision. Sensors **11**(8) (2011) 8164–8179

[13] Ayuso, F., Botella, G., Garcia, C., Prieto, M., Tirado, F.: Gpu-based acceleration of bio-inspired motion estimation model. Concurrency and Computation: Practice and Experience **25**(8) (2013) 1037–1056

[14] Tao, M., Bai, J., Kohli, P., Paris, S.: Simpleflow: A non-iterative, sublinear optical flow algorithm. Comp. Graph. Forum **31**(2pt1) (2012) 345–353

[15] Zach, C., Pock, T., Bischof, H.: A duality based approach for realtime tv-l1 optical flow. In: Proceedings of the 29th DAGM Conference on Pattern Recognition, Berlin, Heidelberg, Springer-Verlag (2007) 214–223

[16] Sánchez Pérez, J., Meinhardt-Llopis, E., Facciolo, G.: TV-L1 Optical Flow Estimation. Image Processing On Line **2013** (2013) 137–150

[17] Stiller, C., Konrad, J.: Estimating motion in image sequences: A tutorial on modeling and computation of 2d motion. IEEE Signal Processing Magazine **16(4)** (1999) 70–91

[18] Li, Z., Yang, Q.: A fast adaptive motion estimation algorithm. In: Computer Science and Electronics Engineering (ICCSEE), 2012 International Conference on. Volume 3. (2012) 656–660

[19] Hu, W., Tan, T., Wang, L., Maybank, S.: A survey on visual surveillance of object motion and behaviors. Trans. Sys. Man Cyber Part C **34**(3) (2004) 334–352

[20] Haritaoglu, I., Harwood, D., Davis, L.S.: W4: Real-time surveillance of people and their activities. IEEE Transactions on Pattern Analysis and Machine Intelligence **22** (2000) 809–830

[21] Wren, C., Azarbayejani, A., Darrell, T., Pentland, A.: Pfinder: real-time tracking of the human body. Pattern Analysis and Machine Intelligence, IEEE Transactions on **19**(7) (1997) 780–785

[22] T. Olson, F.B.: Moving object detection and event recognition algorithms for smart cameras. In: In Proc. DARPA Image Understanding Workshop. (1997) 159–175

[23] Lipton, A.J., Fujiyoshi, H., Patil, R.S.: Moving target classification and tracking from real-time video. In: Proceedings of the 4th IEEE Workshop on Applications of Computer Vision (WACV'98). WACV '98, Washington, DC, USA, IEEE Computer Society (1998) 8–

[24] Collins, R.T., Lipton, A.J., Kanade, T.: Introduction to the special section on video surveillance. IEEE Trans. Pattern Anal. Mach. Intell. **22**(7) (2000) 745–746

[25] Howarth, R., Buxton, H.: Conceptual descriptions from monitoring and watching image sequences. Image and Vision Computing **18**(2) (2000) 105 – 135

[26] Hu, W., Xie, D., Tan, T.: A hierarchical self-organizing approach for learning the patterns of motion trajectories. Trans. Neur. Netw. **15**(1) (2004) 135–144

[27] Toth, D., Aach, T., Metzler, V.: Illumination-invariant change detection. In: Image Analysis and Interpretation, 2000. Proceedings. 4th IEEE Southwest Symposium. (2000) 3–7

[28] Lou, J., Yang, H., Hu, W., Tan, T.: Visual vehicle tracking using an improved ekf. In: in Proc. Asian Conf. Computer Vision, ACCV (2002) 296–301

[29] TIAN Yuan, TAN Tie-Niu, S.H.Z.: A novel robust algorithm for real-time object tracking. Acta Automatica Sinica **28**(05) (2002) 851

[30] André, E., Herzog, G., Rist, T.: On the simultaneous interpretation of real world image sequences and their natural language description: The system soccer. In: In: Proc. of the 8th ECAI. (1988) 449–454

[31] Brand, M., Kettnaker, V.: Discovery and segmentation of activities in video. IEEE Trans. Pattern Anal. Mach. Intell. **22**(8) (2000) 844–851

[32] Haag, M., Theilmann, W., Schäfer, K., Nagel, H.H.: Integration of image sequence evaluation and fuzzy metric temporal logic programming. In: Proceedings of the 21st Annual German Conference on Artificial Intelligence: Advances in Artificial Intelligence. KI '97, London, UK, UK, Springer-Verlag (1997) 301–312

[33] Fritzke, B.: A self-organizing network that can follow non-stationary distributions. In: Proceedings of the 7th International Conference on Artificial Neural Networks. ICANN '97, London, UK, UK, Springer-Verlag (1997) 613–618

[34] Fritzke, B. In: A Growing Neural Gas Network Learns Topologies. Volume 7. MIT Press (1995) 625–632

[35] Frezza-Buet, H.: Following non-stationary distributions by controlling the vector quantization accuracy of a growing neural gas network. Neurocomput. **71** (2008) 1191–1202

[36] Cao, X., Suganthan, P.: Video shot motion characterization based on hierarchical overlapped growing neural gas networks. Multimedia Systems **9**(4) (2003) 378–385

[37] Fritzke, B.: A growing neural gas network learns topologies. In Tesauro, G., Touretzky, D.S., Leen, T.K., eds.: Advances in Neural Information Processing Systems 7. MIT Press, Cambridge MA (1995) 625–632

[38] Hwu, W.m.W.: GPU Computing Gems Emerald Edition. 1st edn. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2011)

[39] Nickolls, J., Dally, W.J.: The gpu computing era. IEEE Micro **30** (2010) 56–69

[40] Horn, D.R., Sugerman, J., Houston, M., Hanrahan, P.: Interactive k-d tree gpu raytracing. In: Proceedings of the 2007 symposium on Interactive 3D graphics and games. I3D '07 (2007) 167–174

[41] : CUDA Programming Guide, version 3.2,. (2010)

[42] Martinetz, T., Berkovich, S., Schulten, K.: 'neural-gas' network for vector quantization and its application to time-series prediction. Neural Networks, IEEE Transactions on **4**(4) (1993) 558–569

[43] Fritzke, B.: Growing cell structures - a self-organizing network for unsupervised and supervised learning. Neural Networks **7** (1993) 1441–1460

[44] Cedras, C., Shah, M.: Motion-based recognition: A survey. Image and Vision Computing **13** (1995) 129–155

[45] Dubuisson, M.P., Jain, A.: A modified hausdorff distance for object matching. In: Pattern Recognition, 1994. Vol. 1 - Conference A: Computer Vision amp; Image Processing., Proceedings of the 12th IAPR International Conference on. Volume 1. (1994) 566–568 vol.1

[46] Wang, X., Tieu, K., Grimson, E.: Learning semantic scene models by trajectory analysis. In: Proceedings of the 9th European Conference on Computer Vision - Volume Part III. ECCV'06, Berlin, Heidelberg, Springer-Verlag (2006) 110–123

[47] Han, M., Xu, W., Tao, H., Gong, Y.: An algorithm for multiple object trajectory tracking. In: Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on. Volume 1. (2004) I–864–I–871 Vol.1

[48] Kim, D., Kim, D.: A novel fitting algorithm using the icp and the particle filters for robust 3d human body motion tracking. In Aghajan, H.K., Prati, A., eds.: VNBA, ACM (2008) 69–76

[49] Gao, T., Li, G., Lian, S., Zhang, J.: Tracking video objects with feature points based particle filtering. Multimedia Tools Appl. **58**(1) (2012) 1–21

[50] Argyros, A.A., Lourakis, M.I.A.: Real-time tracking of multiple skin-colored objects with a possibly moving camera. In: In: ECCV. (2004) 368–379

[51] Sullivan, J., Carlsson, S.: Tracking and labelling of interacting multiple targets. In: Proceedings of the 9th European Conference on Computer Vision - Volume Part III. ECCV'06, Berlin, Heidelberg, Springer-Verlag (2006) 619–632

[52] Papadourakis, V., Argyros, A.: Multiple objects tracking in the presence of long-term occlusions. Comput. Vis. Image Underst. **114**(7) (2010) 835–846

[53] Zhu, L., Zhou, J., Song, J.: Tracking multiple objects through occlusion with online sampling and position estimation. Pattern Recognition **41**(8) (2008) 2447 – 2460

[54] Fisher, R.: Pets04 surveillance ground truth data set. In: In Proc. Sixth IEEE Int. Work. on Performance Evaluation of Tracking and Surveillance. (2004) 1–5

[55] Garcia-Rodriguez, J., Angelopoulou, A., Garcia-Chamizo, J., Psarrou, A., Orts-Escolano, S., Morell-Gimenez, V.: Fast autonomous growing neural gas. In: Neural Networks (IJCNN), The 2011 International Joint Conference on. (2011) 725 –732

[56] Farnebäck, G.: Two-frame motion estimation based on polynomial expansion. In: Proceedings of the 13th Scandinavian Conference on Image Analysis. SCIA'03, Berlin, Heidelberg, Springer-Verlag (2003) 363–370

[57] Nageswaran, J.M., Dutt, N., Krichmar, J.L., Nicolau, A., Veidenbaum, A.: Efficient simulation of large-scale spiking neural networks using cuda graphics processors. In: Proc. Int. Joint Conf. Neural Networks IJCNN 2009. (2009) 2145–2152

[58] Jang, H., Park, A., Jung, K.: Neural network implementation using cuda and openmp. In: Proc. DICTA '08.Digital Image Computing: Techniques and Applications. (2008) 155–161

[59] Juang, C.F., Chen, T.C., Cheng, W.Y.: Speedup of implementing fuzzy neural networks with high-dimensional inputs through parallel processing on graphic processing units. IEEE T. Fuzzy Systems (4) (2011) 717–728

[60] Garcia-Rodriguez, J., Angelopoulou, A., Morell, V., Orts, S., Psarrou, A., Garcia-Chamizo, J.M.: Fast image representation with gpu-based growing neural gas. In: IWANN (2). (2011) 58–65

[61] Igarashi, J., Shouno, O., Fukai, T., Tsujino, H.: 2011 special issue: Real-time simulation of a spiking neural network model of the basal ganglia circuitry using general purpose computing on graphics processing units. Neural Netw. **24** (2011) 950–960

[62] Garcia-Rodriguez, J., Garcia-Chamizo, J.M.: Surveillance and human-computer interaction applications of self-growing models. Applied Soft Computing **11**(7) (2011) 4413 – 4431

[63] Nasse, F., Thurau, C., Fink, G.: Face detection using gpu-based convolutional neural networks. In Jiang, X., Petkov, N., eds.: Computer Analysis of Images and Patterns. Volume 5702 of LNCS. Springer (2009) 83–90

[64] S. Oh, K.J.: View-point insensitive human pose recognition using neural network and cuda. World Academy of Science, Engineering and Technology **60** (2009)