Neural Networks 60 (2014) 203–221

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Online computing of non-stationary distributions velocity fields by an accuracy controlled growing neural gas

Hervé Frezza-Buet *

Supélec, 2, rue Édouard Belin, 57070 Metz, France
Georgia Tech Lorraine, UMI 2958, 2-3, rue Marconi, 57070 Metz, France

ARTICLE INFO

Article history:
Received 9 March 2014
Received in revised form 29 August 2014
Accepted 31 August 2014
Available online 8 September 2014

Keywords:
Vector quantization
Growing neural gas
Velocity field

ABSTRACT

This paper presents a vector quantization process that can be applied online to a stream of inputs. It enables to set up and maintain a dynamical representation of the current information in the stream as a topology preserving graph of prototypical values, as well as a velocity field. The algorithm relies on the formulation of the accuracy of the quantization process, that allows for both the updating of the number of prototypes according to the stream evolution and the stabilization of the representation from which velocities can be extracted. A video processing application is presented.

1. Introduction

Our every day life interaction with the world we are immersed in relies on our ability to recognize people, objects, places, from the flow of analogical signals provided by our sensors. In other words, our brain is able to process dynamical, multi-modal and continuous perceptive signals in a way that enables us to be aware of our reality through the mental handling of symbols, which are the constitutive discrete elements involved in our cognition. Bridging the gap between the discrete, serial and predicative nature of our mind (including speech) and the analogous, high-dimensional, complex and of course unlabeled information provided by the world, is done effortlessly by each of us. Nevertheless, endowing a machine with the least of such skills proved to be a great challenge for computer scientists since the earliest days of artificial intelligence and machine learning. Indeed, automated manipulation of logical predicates hardly meets signal processing techniques, even if both fields provide advanced computing paradigms. This difficulty is referred to as the anchoring problem (Coradeschi & Saffiotti, 2003), that every robotic engineer has experienced when s/he spends hours in adjusting, in vain, the thresholds of crucial decision-making parts of his/her control system.

In the field of machine learning and statistics, vector quantization techniques offer a battery of tools for representing the distribution of vectors sampled according to some unknown and often continuous, process. This representation relies on a finite number of prototypical vectors, determined according to the statistics of the population of the sampled vectors. A straightforward procedure introduced in Martinez and Schulten (1994) enables to connect the obtained prototypes in order to form a *graph* that reflects the topology of the input vector distribution given by the process. Considering this graph as a full symbolic representation of this input is certainly improper, but the fact remains that a graph is a discrete representation of the input process that delivers the vectors, for which graph-related algorithms can be used. This is why we consider the topology preserving vector quantization techniques as relevant approaches to the anchoring problem. The work presented in this paper is a step in this direction. An overview of clustering approaches for artificial intelligence can be found in Qin and Suganthan (2004) and a focus on topology preserving techniques in García-Rodríguez et al. (2012).

Most approaches of vector quantization consider stationary input distributions, but modifications of the algorithms have been proposed in order to cope with non-stationarity (Frezza-Buet, 2008; Fritzke, 1997). This consists in designing algorithms which are robust to changes in the input statistics, so that they keep on representing the instantaneous properties of the input while it changes. In this paper, we aim at going one step further and represent (by a topology preserving vector quantization) the current input statistics staying sensitive to their variations. Indeed, such temporal variations of the input may contain the relevant cues for understanding the semantics of the input. This is for example the motivation for the approaches based on optical flows in computer vision (Chao, Gu, & Napolitano, 2014). In the approach presented in this paper, as opposed to other vector quantization methods for

* Correspondence to: Supélec, 2, rue Édouard Belin, 57070 Metz, France. Tel.: +33 387764735.
E-mail address: Herve.Frezza-Buet@supelec.fr.

http://dx.doi.org/10.1016/j.neunet.2014.08.014
0893-6080/

non stationary data, temporal variations are represented by a velocity field built and updated over the graph representing the data distribution.

The paper is organized as follows. In Section 2 are introduced general notations, as well as a way to model the input data. This model allows for a *quantitative interpretation* of the parameters of our algorithm. Fitting a real data flow to this model provides the semantics for the most central parameters of our approach and thus rationalizes the settings. This section also introduces the *Voronoï contribution* that measures the quality of some vector quantization process. Relying on this measure to control the quantization process is a contribution of this paper and this is what Section 3 addresses. Section 4 presents a new version of our previous GNG-T algorithm (Frezza-Buet, 2008). The explicit use of Voronoï contribution and a real semantics of the parameters enables us to reformulate this previous work into a much more stable algorithm. Because of this significant improvement, GNG-T can be decorated with few additional instructions in order to capture the velocity field of the input distribution, as Section 5 shows. Section 6 presents experiments on 2D non-stationary distributions of pixels, taken from a video as well as a comparison with optical flows. Section 7 concludes.

Last, let us underline that throughout the paper, all algorithms are provided, as well as links to videos and code available from our web site. This allows for reproducing our experiments as well as using GNG-T for addressing broader application domains.

2. Notations and properties

One crucial problem in vector quantization is choosing the appropriate number of prototypes. In case of the clustering into a previously known number of clusters, as for example for clustering digits into ten groups, the appropriate number of prototypes is induced by the problem itself. However, when quantization aims at summarizing a continuous distribution by a discrete set of prototypes, one has to determine the number of prototypes to be used. This choice is even more delicate when non stationary distributions are concerned, since this number of prototypes needs to be adjusted while the distribution changes, in order to be kept permanently appropriate. After all, the meaning of *appropriate* is the core question when the number of prototypes has to be determined.

Besides particularities in their learning rules, the approaches in the Growing Neural Gas (GNG) literature implement a strategy for controlling the number of prototypes. However, this process mainly relies on empirically tuned threshold values, except for RGNG (Qin & Suganthan, 2004) that is based on a minimum description length criterion and for approaches like HOGNG (Cao & Suganthan, 2003) for which GNG participates in a supervised learning where the empirical risk is available for being used as a stopping criterion. For the approaches involving a classical GNG, as it was done recently for character recognition in Fujita (2013), the number of prototypes grows until it reaches a preset maximal value or until the error accumulated for each prototype, when the whole data set is presented, is below a predefined threshold. As the point is to find the suitable number of prototypes for a *fixed* data set, increasing the number of prototypes until some error-based stopping criterion is met remains feasible and then the process stops. Some modified GNG, like GANG (Cselényi, 2005) or SGNG (Tence, Gaubert, Soler, De Loor, & Buche, 2013) applies a similar strategy. SGONG (Stergiopoulou & Papamarkos, 2009) slightly differs since the number of samples for which a given prototype wins is used. The network grows until this number is lower than some threshold. Nevertheless, for the application to hand gesture recognition, the authors also adopt a strategy consisting of using 33 prototypes, from empirical considerations. Last, let us mention GWR (Marsland, Shapiro, & Nehmzow, 2002) that adds a habituation criterion to the error-based criterion. Both are driven by thresholds. Moreover, the habituation calculation makes sense for stationary distributions only.

For our GNG-based approach as well, a strategy for controlling the number of prototypes is proposed. It has the advantage of enabling the user to define a priori, by setting a scalar parameter, what an *appropriate* number of prototypes actually means, according to how s/he intends to use the data. Once defined, this parameter is kept constant even when the data distribution changes, as opposed to the previous methods considering stationary inputs only. As forthcoming Section 3 shows, the meaning of *appropriate* relies on two concepts, introduced beforehand in this section. The first one (Section 2.1) is a view of the data as a rejection sampling process, applied at each time since the data is non stationary. The second one (Section 2.2) is the Voronoï contribution, that is a measure used in the control strategy introduced in Section 3. The relevance of this latter concept is shown by empirical measures, given at the end of Section 2.2.

2.1. Modeling a non stationary input

Let us denote by X a bounded input set from which input samples $\xi \in X$ are drawn. Let us model the variation of input samples concentration over X as a density function $p \in [0,1]^X$, where B^A is the set of functions from A to B. Note that density p is not a probability density since it is not required to be normalized. Let us denote by $a \sim \mathcal{U}_A$ a value a sampled according to a uniform distribution over A. Let us define a sample set $S_p^N \subset X, N \in \mathbb{N}$ a finite set of samples obtained according to p by Algorithm 1.

Algorithm 1 Computation of S_p^N.

```
S_p^N ← ∅ // Start with an empty set.
for i ← 1 to N do
    // Let us consider N attempts to add a sample in S_p^N.
    ξ ~ U_X,  u ~ U_[0,1[ // Choose a random position ξ.
    if u < p(ξ) then
        // The test will pass with a probability p(ξ).
        S_p^N ← S_p^N ∪ {ξ} // ξ is kept (i.e. not rejected).
    end if
end for
return S_p^N
```

Such a procedure is close to a rejection sampling of the probability density related to p (Andrieu, de Freitas, Doucet, & Jordan, 2003). The actual number of samples in S_p^N depends on both p and N, which will be discussed further.

A non stationary input is modeled here as a sequence of sample sets. Let $p^t \in [0,1]^X$ be a non stationary density and N^t some arbitrary sampling number at time t. A non stationary input stream is defined throughout the paper as the sequence $\left(S_{p^t}^{N^t}\right)_{t\in T}$. Let us stress that the input stream defined by this way is discrete, since time instants are organized as a sequence and each $S_{p^t}^{N^t}$ is a finite set of samples.

2.2. Voronoï contribution

Vector quantization basically consists in summarizing a distribution by a finite set of representative values, usually called the prototypes. The prototypes are representative since they are chosen in order to minimize a distortion function. Classical definitions of vector quantization concepts can be found in Patra (2011), where densities of probability are concerned. Let us recall here these definitions in the restricted case of a finite input sample set

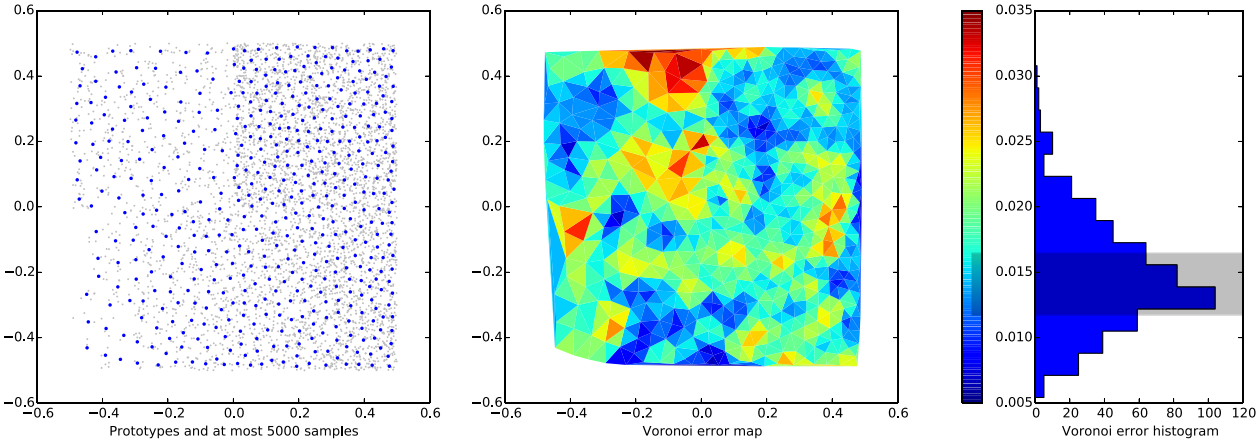

Fig. 1. Voronoï contribution distribution over the optimal prototypes. $N = 50\,000$, $\kappa = 500$, $p(\xi) = \mathbb{1}_{0\leq\xi_y} \times \left(0.2 \times \mathbb{1}_{\xi_x\leq 0} + \mathbb{1}_{0<\xi_x}\right) + \mathbb{1}_{\xi_y<0} \times (\xi_x + 0.5)$, $d(\xi, \xi') = \left\|\xi - \xi'\right\|^2$. The shortest 50% confidence interval is $[1.173 \times 10^{-2}, 1.646 \times 10^{-2}]$ (see the shaded area on the histogram). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

S rather than distributions. Let us name $\Omega_\kappa = \{\omega_1, \omega_2, \ldots, \omega_\kappa\} \in X^{|\kappa|}$ the $\kappa \in \mathbb{N}^\star$ prototypes used to represent the samples in S. The notation $X^{|\kappa|}$ stands for the set of all subsets of X having exactly κ elements. The representation distortion $\mathcal{E}^S_{\Omega_\kappa}$ induced by Ω_κ on S is

$$\mathcal{E}^S_{\Omega_\kappa} = \frac{1}{|S|} \sum_{\xi\in S} d(\xi, \Omega_\kappa) \tag{1}$$

where d is a similarity measure, typically the squared Euclidean distance; $d(\xi, \Omega_\kappa)$ is the distance from ξ to the closest element of Ω_κ. The goal of most vector quantization algorithms is to find $\Omega^\star_{\kappa,S} = \text{argmin}_{\Omega_\kappa\in X^{|\kappa|}} \mathcal{E}^S_{\Omega_\kappa}$.

Let us split S into a partition of κ subsets $\left\{V^S_{\omega_1}, V^S_{\omega_2}, \ldots, V^S_{\omega_\kappa}\right\}$ according to the κ prototypes in Ω_κ. Let us first define for any $\xi \in S$ the closest prototype $\omega^\star_\xi$ as the element in the set $\text{argmin}_{\omega_i\in\Omega_\kappa} d(\xi, \omega_i)$ with the lowest index. The set is usually a singleton, since having several prototypes equidistant to some sample is very unlikely in floating point computation. $\omega^\star_\xi$ enables to define each element in the partition of S as $V^S_\omega = \left\{\xi \in S \mid \omega^\star_\xi = \omega\right\}$. V^S_ω contains the samples of S belonging to the Voronoï region of the prototype ω. V^S_ω is a finite set, referred to as a *Voronoï cell* in the following.

Let us define the *Voronoï contribution* $\mathcal{V}^S_\omega$, i.e. the contribution of a Voronoï cell to the global distortion $\mathcal{E}^S_{\Omega_\kappa}$, as

$$\mathcal{V}^S_\omega = \sum_{\xi\in V^S_\omega} d(\xi, \omega). \tag{2}$$

As the $\left\{V^S_\omega\right\}_{\omega\in\Omega_\kappa}$ form a partition of S, it is obvious that

$$\mathcal{E}^S_{\Omega_\kappa} = \frac{1}{|S|} \sum_{\omega\in\Omega_\kappa} \mathcal{V}^S_\omega. \tag{3}$$

The Voronoï contribution distribution over the prototypes is considered in this paper rather than the global distortion. The underlying idea is that when the minimal distortion is reached, the prototypes in $\Omega^\star_{\kappa,S}$ attain some kind of equilibrium where the errors between samples and prototypes are equally shared. This share is a common Voronoï contribution value, that represents the *quantization accuracy*. In the GNG algorithm by Fritzke (1995c), a neuron accumulates the error with the current sample when it wins the competition, i.e. when the sample lies within the neuron Voronoï region. This error accumulation is close to the computation of the Voronoï contribution introduced here.

Let us illustrate this on an artificial example. Let $X = [-0.5, 0.5]^2$ and $p(\xi) = \mathbb{1}_{0\leq\xi_y} \times \left(0.2 \times \mathbb{1}_{\xi_x\leq 0} + \mathbb{1}_{0<\xi_x}\right) + \mathbb{1}_{\xi_y<0} \times (\xi_x + 0.5)$. Let us set up $S = S_p^{50\,000}$ following the procedure given in Section 2.1, use $\kappa = 500$ prototypes and compute $\Omega^\star_{\kappa,S}$ with the Linde–Buzo–Gray algorithm (Linde, Buzo, & Gray, 1980). In Fig. 1, on the left, input samples in S are plotted (only 5000 of them indeed, for the sake of clarity), as well as the prototypes in $\Omega^\star_{\kappa,S}$ (the thicker dots). The center plot in the figure shows a Delaunay triangulation of the prototypes, with a color depending on their respective Voronoï contribution. The color bar aside this plot is aligned to the histogram on the right so that the vertical scales fit. This histogram shows the distribution of Voronoï contribution values among the prototypes. In Fig. 2, the mean prototype error, i.e. $\frac{1}{\left|V^S_\omega\right|} \sum_{\xi\in V^S_\omega} d(\xi, \omega) = \frac{1}{\left|V^S_\omega\right|} \mathcal{V}^S_\omega$ is represented. It is obvious that the values are related to the spatial extension of the V^S_ω and the concentration of prototypes at higher sample density regions is visible on the central plot in Fig. 2 in blue.

As opposed to Fig. 2, Fig. 1 shows that the Voronoï contribution fluctuations are not dependent on the density of points over the input space, i.e. not dependent on p. As wide Voronoï cells correspond to sparse samples (see for example top-left prototypes in Fig. 1), the value $\mathcal{V}^S_\omega$ for such ωs is due to the summation of few terms (see Eq. (2)), but each term has a quite high value. On the contrary, for narrow Voronoï cells where samples are dense (see for example top right prototypes in Fig. 1), the value $\mathcal{V}^S_\omega$ is the summation of many small values.

It has been mentioned previously that the optimal prototypes reach an equilibrium where the Voronoï contribution is shared equally, whatever the local value of p around the prototypes. Fig. 1 shows that this assertion is idealized, since the real Voronoï contribution distribution is made of contiguous regions of higher or lower values. As far as the author knows, this has not been addressed formally in the vector quantization field. Such variations seem to occur systematically, as Figs. 3–6 show. In these figures, the number of prototypes is determined as $|S|/100$ so that the Voronoï contributions lie in the same range of values in the four figures. Fig. 3 shows a uniform density in a square. Fig. 4 shows a disk-shaped density, that is likely to reduce side effects that might have been induced by the corner of the square. Fig. 5 implements a variation of density and Fig. 6 a mono-dimensional density. These experiments suggest, empirically, that irregularities in shape, concentration or dimension may not influence the distribution of Voronoï contribution variations.

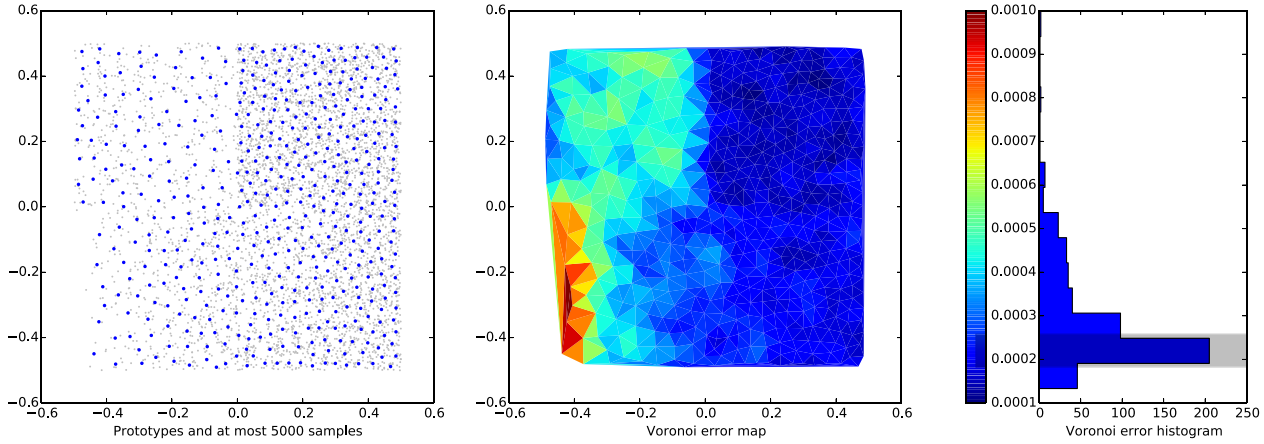

Fig. 2. Mean prototype error distribution over the optimal prototypes. $N = 50\,000$, $\kappa = 500$, $p(\xi) = \mathbb{1}_{0\leq\xi_y} \times \left(0.2 \times \mathbb{1}_{\xi_x\leq 0} + \mathbb{1}_{0<\xi_x}\right) + \mathbb{1}_{\xi_y<0} \times (\xi_x + 0.5)$, $d\left(\xi, \xi'\right) = \left\|\xi - \xi'\right\|^2$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

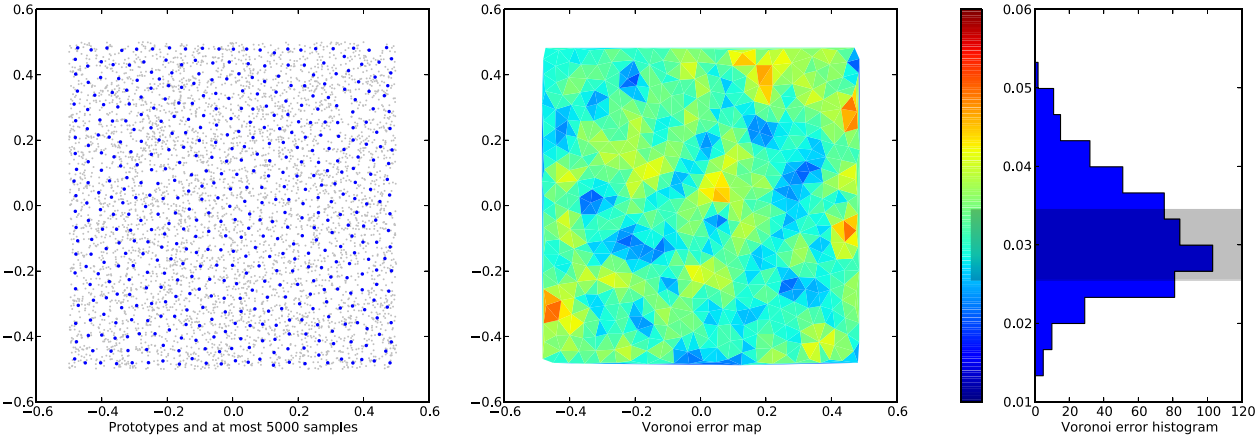

Fig. 3. Voronoï contribution distribution over the optimal prototypes. $N = 50\,000$, $\kappa = 500$, $p(\xi) = 1$, $d\left(\xi, \xi'\right) = \left\|\xi - \xi'\right\|^2$. The shortest 50% confidence interval is $[2.55 \times 10^{-2}, 3.45 \times 10^{-2}]$ (see the shaded area on the histogram).

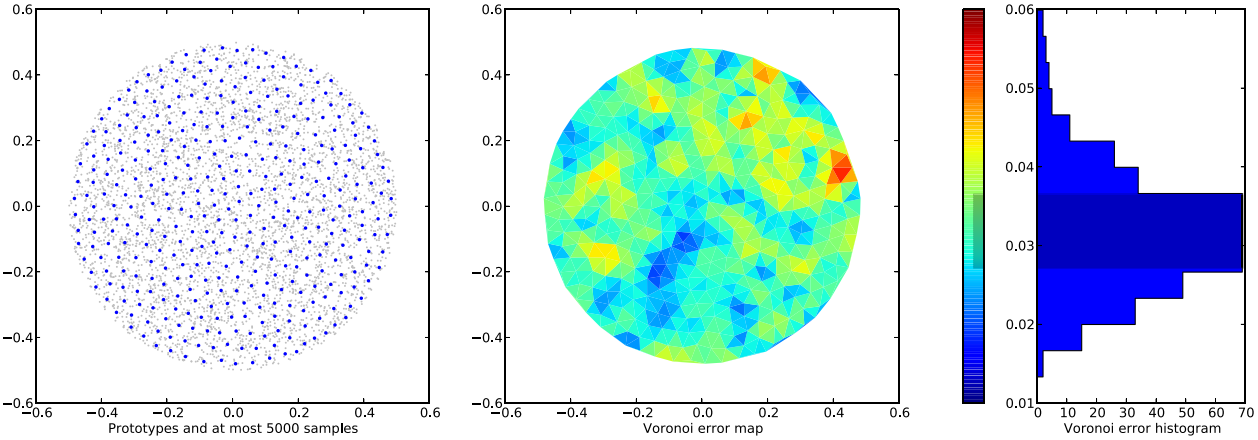

Fig. 4. Voronoï contribution distribution over the optimal prototypes. $N = 50\,000$, $\kappa = 391$, $p(\xi) = \mathbb{1}_{\|\xi\|\leq 0.5}$, $d\left(\xi, \xi'\right) = \left\|\xi - \xi'\right\|^2$. The shortest 50% confidence interval is $[2.715 \times 10^{-2}, 3.655 \times 10^{-2}]$ (see the shaded area on the histogram).

3. Controlling the quantization accuracy

Usually, in vector quantization algorithms, the number of prototypes is a crucial value that has to be determined empirically. For the κ-means (Linde et al., 1980), choosing the right value of κ when the number of clusters in the samples is unknown is difficult and the choice is left to the user. The same stands for Kohonen self-organizing maps (SOMs) (Kohonen, 2001) since the map architecture is given a priori. For growing structures, as Growing Neural Gas (Fritzke, 1995c), Growing Grids (Fritzke, 1995a), the growth is stopped when some user-defined number of nodes is reached, as previously stated. Let us recall here that controlling the number of prototypes κ according to a desired error-based quantization accuracy is the motivation of GNG-T (Frezza-Buet, 2008)

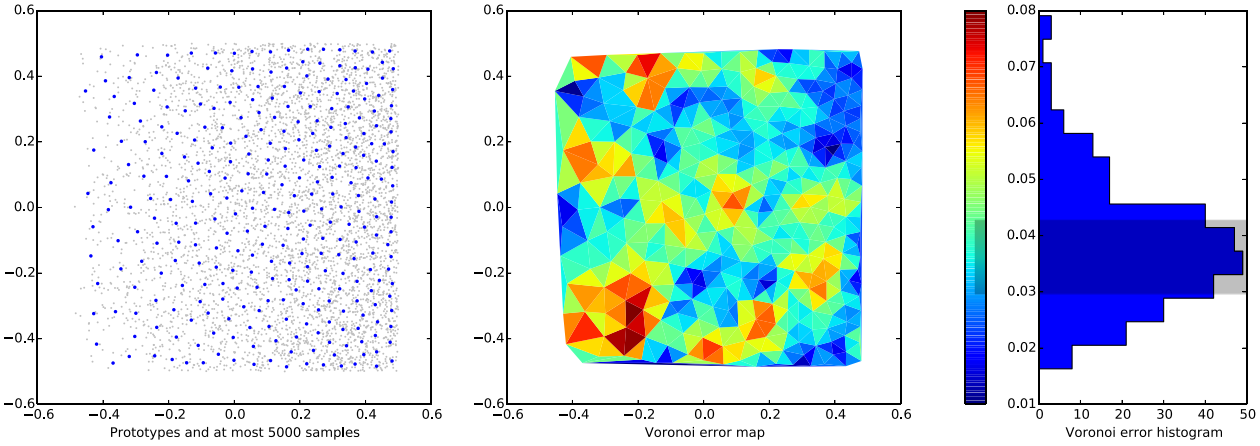

Fig. 5. Voronoï contribution distribution over the optimal prototypes. $N = 50\,000$, $\kappa = 300$, $p(\xi) = \xi_x + 0.5$, $d(\xi, \xi') = \|\xi - \xi'\|^2$. The shortest 50% confidence interval is $[2.965 \times 10^{-2}, 4.272 \times 10^{-2}]$ (see the shaded area on the histogram).

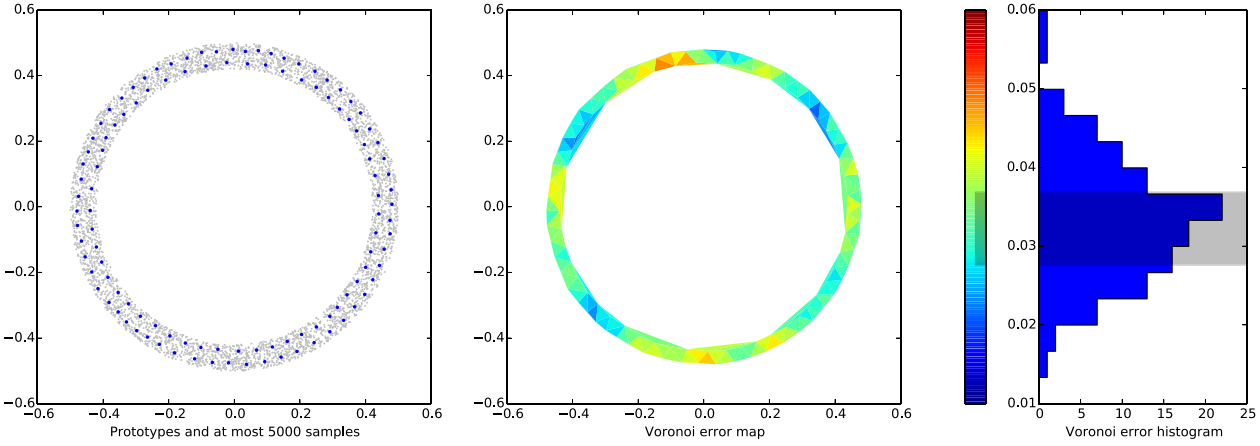

Fig. 6. Voronoï contribution distribution over the optimal prototypes. $N = 50\,000$, $\kappa = 114$, $p(\xi) = \mathbb{1}_{0.42 \le \|\xi\| \le 0.5}$, $d(\xi, \xi') = \|\xi - \xi'\|^2$. The shortest 50% confidence interval is $[2.76 \times 10^{-2}, 3.686 \times 10^{-2}]$ (see the shaded area on the histogram).

as well and a significant improvement of this control is proposed in the following, based on a geometrical interpretation of the error value.

3.1. Definition of a scalar accuracy measure

From the input modeling introduced in Section 2.1 and the way the input sample set S_p^N is obtained, let us notice that the Voronoï contribution $\mathcal{V}_\omega^{S_p^N}$ introduced in Section 2.2, Eq. (2), depends on the number N used in Algorithm 1. As Algorithm 1 is a random process, the forthcoming equations are approximations valid for a large N, rather than strict equalities. These approximations aim at justifying the definition of an accuracy measure used further. The influence of N is revealed by noticing that for a given $k \in \mathbb{N}$, κ, p and Ω_κ, the following obviously stands:

$$\left|S_p^{k\cdot N}\right| \approx k \cdot \left|S_p^N\right|, \quad \forall \omega \in \Omega_\kappa, \qquad \left|V_\omega^{S_p^{k\cdot N}}\right| \approx k \cdot \left|V_\omega^{S_p^N}\right| \tag{4}$$

and thus

$$\forall \omega \in \Omega_\kappa, \quad \mathcal{V}_\omega^{S_p^{k\cdot N}} \approx k \cdot \mathcal{V}_\omega^{S_p^N}. \tag{5}$$

The Voronoï contribution values are thus directly linked to N, that is *not* the number of samples in S_p^N. Let us denote by $T \in \mathbb{R}^+$ the value used to control the quantization accuracy, called the *target* as in Frezza-Buet (2008). Let us also determine the δ-shortest confident interval (Guenther, 1969) from the Voronoï contribution set $\left\{\mathcal{V}_\omega^S\right\}_{\omega \in \Omega}$, where $\Omega = \Omega_{\kappa,S}^\star$ and $S = S_p^N$. We consider the quantization as fitting the target when $N \cdot T$ belongs to this δ-shortest confidence interval (δ-SCI, see Algorithm 2). The use of $N \cdot T$ makes explicit the influence of N on the contribution values. As the distribution of Voronoï contributions is non symmetrical, the δ-SCI represents the most informational values better than the mean or the median. The fitting decision is then much more stable than the mean previously used in Frezza-Buet (2008), as shown further. We use $\delta = 0.5$ in the experiments reported in Section 3 and the δ-SCI is shown as a dark range in the histograms in the figures.

3.2. Interpretation of the target value

Driving the quantization process (more precisely controlling κ) according to a scalar value T requires to choose this value carefully for a given problem. Trial and error adjustment can be done, but the input modeling introduced in Section 2.1 allows for an interpretation of the value T.

Let us consider $p = 1$, i.e. $S = S_p^N$ consists of exactly N random samples from X. Let us denote by $\kappa^\star$ the number of prototypes that the user would like to obtain in this simple case. When the optimal prototypes $\omega \in \Omega^\star = \Omega_{\kappa^\star,S}^\star$ are found, let us consider the induced Voronoï regions $\{\nu_\omega\}_{\omega \in \Omega^\star}$, with $\nu_\omega = \left\{\xi \in X \mid \omega_\xi^\star = \omega\right\}$. See Section 2.2 for the definition of $\omega_\xi^\star$. These regions form an homogeneous partition of X (this is an idealized view of the optimal distribution of prototypes, as discussed at the end of Section 2.2). Let us denote by $|A|$, for any subset A of X, the size $\int_A \mathrm{d}\xi$. We can

consider roughly that, for any $\omega \in \Omega^\star$, $|v_\omega| \approx \frac{|X|}{\kappa^\star}$ since X is tiled by $\kappa^\star$ similar Voronoï regions.

Let us note that $d_\omega = \frac{1}{|v_\omega|}\int_{v_\omega} d(\xi,\omega)\,\mathrm{d}\xi$ is the expectancy of the error between a sample and the prototype within a Voronoï region. This expectancy is approximated by the average error measured $\mathcal{V}^S_\omega / \left|V^S_\omega\right|$ within the Voronoï cell V^S_ω. The following then stands

$$\frac{\mathcal{V}^S_\omega}{\left|V^S_\omega\right|} \approx d_\omega \Rightarrow \mathcal{V}^S_\omega \approx \left|V^S_\omega\right| \cdot d_\omega \Rightarrow \mathcal{V}^S_\omega \approx \frac{N}{\kappa^\star}\cdot d_\omega \tag{6}$$

since S contains N samples that are roughly divided up equally into Voronoï cells, i.e. $\left|V^S_\omega\right| \approx N/\kappa^\star$. In Section 3.1, the target T has been introduced as the value such as $N \cdot T$ is the most meaningful Voronoï contribution value. Identifying $N \cdot T$ with Eq. (6) leads to the interpretation of the target as

$$T = \frac{d_\omega}{\kappa^\star}. \tag{7}$$

Let us take the examples of Figs. 3–6 to illustrate the interpretation of T. Let us consider that the desired accuracy corresponds to the use of $\kappa^\star = 500$ samples when $p = 1$. This is what Fig. 3 actually shows. As $X = [-0.5, 0.5]^2$, $|X| = 1$ and thus $|v_\omega| = 1/\kappa^\star$ for the tiling Voronoï regions. Let us approximate their shape as a disk, whose surface is $1/\kappa^\star$, centered at ω. As $d\left(\xi,\xi'\right) = \left\|\xi - \xi'\right\|^2$, d_ω can be computed analytically as $d_\omega = 1/(2\pi\kappa^\star)$. Thus, according to Eq. (7), $T = 1/(2\pi\kappa^{\star 2})$. In Fig. 3, $N = 50\,000$ is used, so $N \cdot T \approx 0.0318$. It actually belongs to the 50%-SCI shaded in Fig. 3 histogram. The same stands for Figs. 4 and 6, since the number of prototypes has been chosen accordingly (see the end of Section 2.2), as opposed to Fig. 1 where the number of prototypes have been fixed arbitrarily as $\kappa = 500$. For this figure, 0.0318 is above the 50%-SCI, which means that there are too many prototypes if the goal was to obtain the same quantization accuracy as in Figs. 3–6. Indeed, on the upper right corner of the input space in Fig. 1, the density of prototypes is locally higher than the density in Fig. 3, whereas $p(\xi) = 1$ in the upper right corner of both figures.

Let us re-plot Fig. 1 with a number of prototypes determined from the target. The procedure consists in determining κ by dichotomy, until $N \cdot T = N/2\pi\kappa^{\star 2}$ belongs to the 50%-SCI of the Voronoï contributions. Fig. 7 shows the result; the upper right corner is now similar to Fig. 3.

Algorithm 2 `shortest_confidence_interval`(V, δ)

Require: $\delta \in [.5, 1]$ // $V = \{v_i\}_{0\le i<N}$ is the set of N values.
1: $l \leftarrow$ (int) $(\delta.N)$ // (int) (x) stands for the closest integer to x
2: sort V in an increasing order
3: // r is set greater than any possible interval length.
4: $r \leftarrow v_{N-1} - v_0 + 1$
5: **for** $k \leftarrow 0$ **to** $N - l$ **do**
6: // Values from k to $k + l$ are a fraction δ of all the values.
7: // r' is the width of the range of these values.
8: $r' \leftarrow v_{k+l-1} - v_k$
9: **if** $r' < r$ **then**
10: $r \leftarrow r',\ j \leftarrow k$ // The minimum found is saved.
11: **end if**
12: **end for**
13: **return** (v_j, v_{j+l-1})

4. Growing Neural Gas with Targeting (GNG-T)

In Frezza-Buet (2008), an adaptation of Growing Neural Gas (GNG) by Fritzke has been proposed (GNG-T) in order to track fast changing and non-stationary distributions. From an appropriate modeling of non-stationary inputs and subsequent Voronoï contribution definition (Section 2), as well as from a qualitative and quantitative interpretation of the target in terms of quantization accuracy (Section 3), a re-writing of GNG-T can be proposed.

4.1. The algorithm

The GNG algorithm by Fritzke handles input samples online, since examples are provided one by one to the learning process. Even if GNG-T is close to GNG, there is a need for chunking the sample stream into *epochs*. This chunking is crucial for GNG-T, since it triggers a reset of the internal values used for computing the statistics. From the input modeling of Section 2.1, let us define here an epoch as the setting up of a sample set S^N_p. The connection with real data sampling will be discussed further. The handling of an epoch by GNG-T is described in Procedure 3, where G is the graph of prototypes.

Procedure 3 `gngt_epoch`$(G, S, \alpha,$ dyn_topo$)$

1: // Use dyn_topo = **true** for GNG-T, it enables the graph evolution.
2: // $\alpha \in [0, 1]$ is the learning rate.
3: $S' \leftarrow S$
4: // An epoch is a pass involving all the samples in S.
5: `gngt_open_epoch`(G) // This resets internal variables.
6: **repeat**
7: $\xi \sim \mathcal{U}_{S'},\ S' \leftarrow S' \setminus \{\xi\}$ // Extracts a random sample from S'.
8: `gngt_submit`$(G, \xi, \alpha,$ dyn_topo$)$ // This updates prototypes.
9: **until** $S' = \emptyset$
10: `gngt_close_epoch`$(G,$ dyn_topo$)$ // Topology is updated here.

Procedure 4 `gngt_open_epoch`(G)

1: **for all** $v \in \mathrm{V}(G)$ **do**
2: $v_\mathrm{e} \leftarrow 0$ // v_e is the Voronoï error accumulated by v.
3: $v_\mathrm{n} \leftarrow 0$ // v_n is the number of times v_ω has been the closest to an input sample.
4: **end for**

Let us denote by $\mathrm{V}(G)$ the set of vertices of G and let us note $\kappa^G = |\mathrm{V}(G)|$. Each of the κ^G vertices of G gathers several quantities. For a vertex $v \in \mathrm{V}(G)$, the main of these quantities is $v_\omega \in X$ that is the prototype handled by the vertex. G is a *non oriented* graph. Let us denote by $\mathrm{e}^{v\leftrightarrow v'}$ an edge between v and v', by $\mathrm{E}(v)$ the edges involving the vertex v and $\mathrm{V}(v) = \left\{v' \in \mathrm{V}(G) \mid \exists \mathrm{e}^{v\leftrightarrow v'} \in \mathrm{E}(v)\right\}$ the neighboring vertices of v. As opposed to GNG where accumulated errors are continuously updated (and shared when new nodes are created), GNG-T clears the accumulated errors at each epoch (see Procedure 4).

After this clearing, samples are submitted one by one, the submission procedure being very close to what GNG does (see Procedure 5). The submission is where prototypes are updated according to the submitted input. This update has to reduce the distance between the prototype and the sample, according to a coefficient $\alpha \in [0, 1]$, such that the prototype is left unchanged when $\alpha = 0$ and is set to the sample when $\alpha = 1$. In this paper, the classical vector quantization learning rule (see Procedure 6) is used.

Once all the samples of S^N_p are submitted, the epoch has to be closed and statistics computed during the submission stage are used to update the topology of graph G. In other versions of GNG, the time for updating the topology is often related to the number of input samples submitted from the start and the current number of vertices. Here again, considering epochs makes the occurrence of the topology evolution more rational. The topology is updated according to Procedure 7.

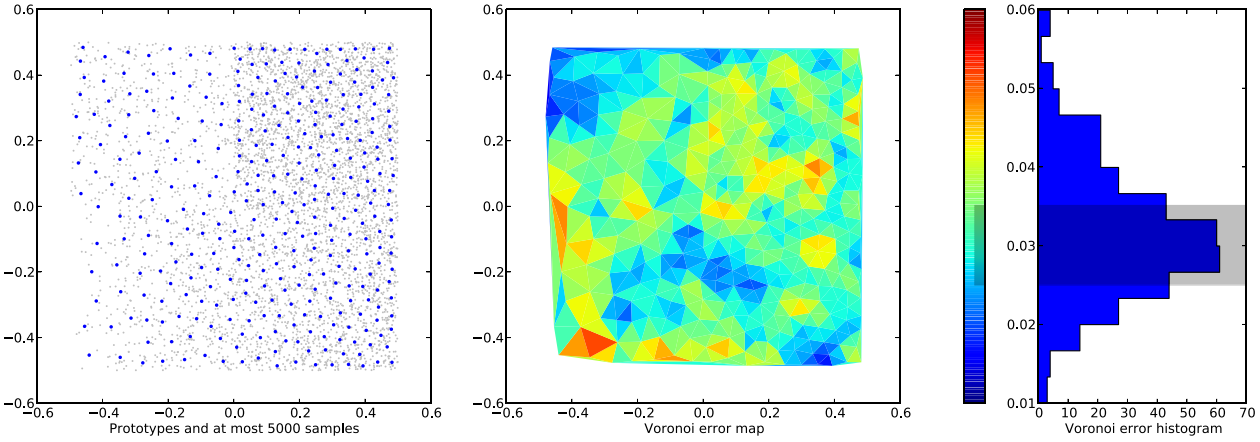

Fig. 7. Voronoï contribution distribution over the optimal prototypes. $N = 50\,000$, $p(\xi) = \mathbb{1}_{\xi_y \le 0} \times \left(0.2 \times \mathbb{1}_{\xi_x \le 0} + \mathbb{1}_{0 < \xi_x}\right) + \mathbb{1}_{0 < \xi_y} \times (\xi_x + 0.5)$, $d(\xi, \xi') = \left\|\xi - \xi'\right\|^2$. The shortest 50% confidence interval is $[2.501 \times 10^{-2}, 3.511 \times 10^{-2}]$ (see the shaded area on the histogram). The value $\kappa^\star$ has been adjusted so that $N \cdot T = N/2\pi\kappa^{\star 2}$ is in the darkened range of the histogram. This leads to $\kappa^\star = 343$ in this experiment.

Procedure 5 `gngt_submit`$(G, \xi, \alpha,$ dyn_topo$)$

1: **if** $\kappa^G < 2$ **then**
2: **if** dyn_topo **then**
3: Add a new vertex v to G.
4: $v_\omega \leftarrow \xi$ // The prototype takes the value of the sample.
5: $v_\mathrm{e} \leftarrow 0$
6: $v_\mathrm{n} \leftarrow 1$
7: **end if**
8: **return** // Exit the algorithm here.
9: **end if**
10: $v^\star \leftarrow \mathrm{argmin}_{v \in \mathrm{V}(G)} d(v_\omega, \xi)$ // $v^\star$ is the closest vertex.
11: **if** dyn_topo **then**
12: $v^{\star\star} \leftarrow \mathrm{argmin}_{v \in \mathrm{V}(G) \setminus \{v^\star\}} d(V_\omega, \xi)$ // $v^{\star\star}$ is the second closest vertex.
13: create the edge $\mathrm{e}^{v^\star \leftrightarrow v^{\star\star}}$ if it does not exist yet.
14: $\mathrm{e}_\mathrm{a}^{v^\star \leftrightarrow v^{\star\star}} \leftarrow 0$ // Set/reset the age of the edge at a 'newborn' value.
15: **for all** $\mathrm{e} \in \mathrm{E}(v^\star) \setminus \left\{\mathrm{e}^{v^\star \leftrightarrow v^{\star\star}}\right\}$ **do**
16: $e_\mathrm{a} \leftarrow e_\mathrm{a} + 1$
17: **if** $e_\mathrm{a} > \varphi$ **then**
18: remove e // Too old edges are removed.
19: **end if**
20: **end for**
21: **end if**
22: $v_\mathrm{e}^\star \leftarrow v_\mathrm{e}^\star + d\left(v_\omega^\star, \xi\right)$ // Update winner statistics.
23: $v_\mathrm{n}^\star \leftarrow v_\mathrm{n}^\star + 1$
24: `vq_learn`$(\alpha, v_\omega^\star, \xi)$
25: **for all** $v \in \mathrm{V}(v^\star)$ **do**
26: `vq_learn`$(\zeta\alpha, v_\omega, \xi)$ // $\zeta \in]0, 1]$ so that the learning of neighbors is weaker.
27: **end for**

Procedure 6 `vq_learn`(α, ω, ξ)

Require: $\alpha \in [0, 1]$
1: $\omega \leftarrow \omega + \alpha . (\xi - \omega)$

There are slight modifications from GNG in the network evolution procedure described by Procedure 7. First, at line 12, the decision of adding, removing or keeping the current number of neurons is taken according to the state of the prototypes (indeed the accumulated error v_e for each of them, as will be discussed later), while GNG essentially adds neurons. A second change is line 26. In GNG (and our previous version of GNG-T), $v_\omega \leftarrow \frac{v_\omega^\star + v_\omega^{\star\star}}{2}$ was used, i.e. $\lambda = 0.5$. Nevertheless, with this rule, the newly created neuron can be isolated between two clusters of samples and then

Procedure 7 `gngt_close_epoch`$(G,$ dyn_topo$)$

1: **if not** dyn_topo **then**
2: **return** // Exit the algorithm here.
3: **end if**
4: **for all** $v \in \mathrm{V}(G)$ **do**
5: **if** $v_\mathrm{n} = 0$ **then**
6: remove v from G. // v has never won any competition.
7: **end if**
8: **end for**
9: **if** $\kappa^G = 0$ **then**
10: **return** // Exit the algorithm here.
11: **end if**
12: evol $\leftarrow$ `compute_evolution`(G)
13: **if** evol $=$ `None` **then**
14: **return** // Exit the algorithm here.
15: **else if** evol $=$ `Remove` **then**
16: remove $\mathrm{argmin}_{v \in \mathrm{V}(G)} v_\mathrm{e}$
17: **else if** evol $=$ `Add` **then**
18: find $v^\star = \mathrm{argmax}_{v \in \mathrm{V}(G)} v_\mathrm{e}$
19: create v and add it in the graph.
20: $v_\omega \leftarrow v_\omega^\star$ // The winning node is cloned
21: **if** $\mathrm{E}(v^\star) = \emptyset$ **then**
22: add $\mathrm{e}^{v \leftrightarrow v^\star}$ // Connect the two clones...
23: $\mathrm{e}_\mathrm{a}^{v \leftrightarrow v^\star} \leftarrow 0$ // ... with a 'newborn' edge.
24: **else**
25: find $v^{\star\star} = \mathrm{argmax}_{v \in \mathrm{V}(v^\star)} v_\mathrm{e}$ // $v^{\star\star}$ is the neighbor of $v^\star$ with the highest accumulated error.
26: `vq_learn`$(\lambda, v_\omega, v_\omega^{\star\star})$ // The clone is moved slightly toward $v^{\star\star}$.
27: add $\mathrm{e}^{v \leftrightarrow v^\star}$ and $\mathrm{e}^{v \leftrightarrow v^{\star\star}}$ to the graph.
28: $\mathrm{e}_\mathrm{a}^{v \leftrightarrow v^\star} \leftarrow 0$, $\mathrm{e}_\mathrm{a}^{v \leftrightarrow v^{\star\star}} \leftarrow 0$ // Make the new edges 'newborn'.
29: **end if**
30: **end if**

immediately deleted at next epoch at line 6... and then created again, and so on. Using a positive $\lambda \approx 0$ at line 26 avoids such instabilities.

4.2. Controlling the number of prototypes

The control of the number of prototypes in GNG-T depends on the actual procedure used at line 12 of Procedure 7. Let us first consider a straightforward control based on the mean (see Procedure 8) and illustrate the behavior of GNG-T with this control on the following example. $X = [-0.5, 0.5]^2$, as in Section 2.2 and successive epochs $t, t+1, \ldots$ are run with a constant density function

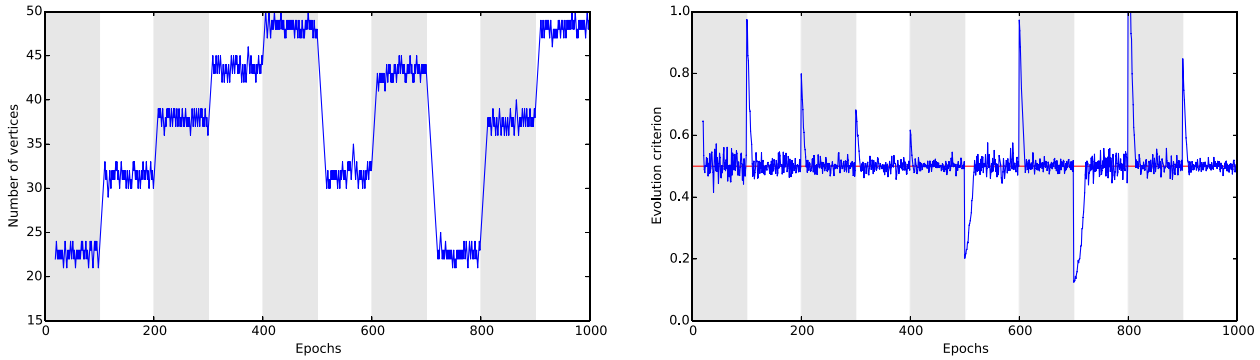

Fig. 8. Evolution of the number of nodes computed by GNG-T for successive epochs. The distribution $p^t(\xi) = \text{const}$ at each epoch, the constant value differs from one chunk of 100 epochs to the following. Chunks are marked as stripes in the plots and const is taken in $\{0.2, 0.4, 0.6, 0.8, 1, 0.4, 0.8, 0.2, 0.6, 1\}$. On the left, the number of nodes is plotted. The right plot shows the value μ computed by Procedure 8, as well as the reference value $N \cdot T = 0.5$ that drives the network evolution (see the red horizontal line). Parameters of Algorithms 5 and 7 are $\varphi = 50$, $\alpha = 0.005$, $\zeta = 0.5$, $\lambda = 10^{-3}$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

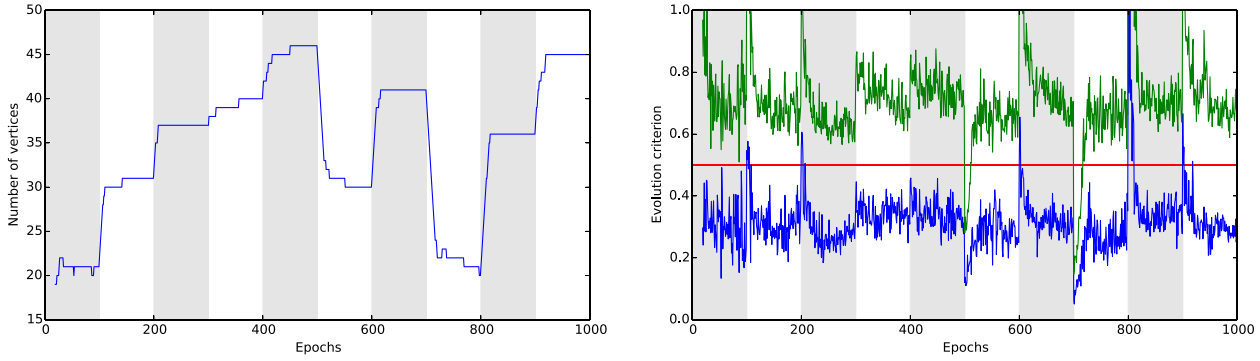

Fig. 9. Experiments and graphical convention are similar to Fig. 8. On the right plot, min and max values computed by Procedure 9 are plotted. The parameters of Procedure 9 are $\delta = 75\%$, $\mu = 0.2$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$\forall \xi \in X,\ p^t(\xi) = d^t$, with $\forall t, d^t \in [0, 1]$ (the notations are introduced in Section 2.1). More precisely, d^t is kept constant for 100 epochs and then set to another value for the next 100 epochs, and so on. At each epoch t, a set of input samples $S_{p^t}^{N=5000}$ is built, according to Algorithm 1. This set is used to train GNG-T (see Procedure 3). The target is $T = 10^{-4}$, so that the $p^t = 1$ leads to $\kappa^\star \approx 40$ prototypes (see. Eq. (7) and the text following it). Fig. 8 shows that GNG-T actually adjusts the number of nodes according to $N \cdot T$, but this number of nodes is very unstable. Nodes are permanently created or removed in order to stick to the target. This leads to a constant spurious motion of the prototypes that permanently readjust their position according to their current number.[1]

Procedure 8 `compute_evolution`(G)

$\mu = \frac{1}{\kappa^G} \sum_{v \in G} v_e$ // μ is the average of accumulated errors.
if $N.T < \mu$ **then**
  return `Add`
else
  return `Remove`
end if

The naive control (Procedure 8) is actually the one that was used in our previous implementation (Frezza-Buet, 2008). The spurious motion was not prohibitive since no velocity extraction was expected from the graph, which is not the case in this paper. Fortunately, the spurious motion can be significantly reduced by considering δ-SCI introduced in Section 3.1. Indeed, Procedure 9 can be used instead of Procedure 8. This leads to the behavior depicted in Fig. 9. It can be seen that the number of prototypes is much more stable.

Procedure 9 `compute_evolution`(G)

(min, max) $\leftarrow$ `shortest_confidence_interval`$(\{v_e\}_{v \in G}, \delta)$
$\Delta \leftarrow \max - \min$
$a \leftarrow \min + \mu\Delta$ // $\mu \in [0, .5[$, [min, max] shrinks to $[a, b]$.
$b \leftarrow \min + (1 - \mu)\Delta$
if $N.T < a$ **then**
  return `Add`
else if $b < N.T$ **then**
  return `Remove`
else
  return `None`
end if

In this paper, a refinement of Procedure 9 is proposed in order to drive the adjustment of the graph size. It consists in low-pass filtering of the δ-SCI bounds measured at successive epochs (see Procedure 10). Fig. 10 shows the behavior of GNG-T in this case.[2]

5. Velocity field

The motivation for the stabilization of GNG-T presented in this paper is the computation of the velocity field of some

[1] See the movie http://malis.metz.supelec.fr/depot/Vq/Paper/expe-004-101.avi.

[2] See the movie http://malis.metz.supelec.fr/depot/Vq/Paper/expe-004-103.avi.

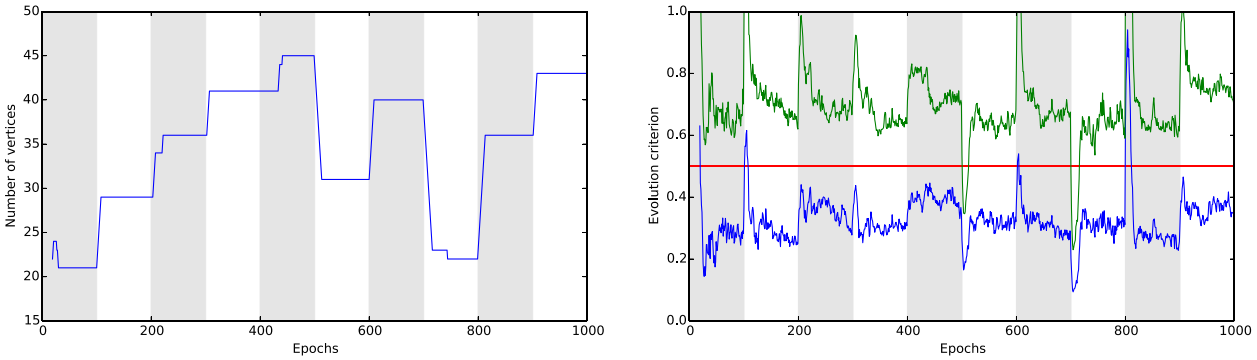

Fig. 10. Experiments and graphical convention are similar to Fig. 8. On the right plot, min and max values computed by Procedure 10 are plotted. The parameters of Procedure 10 are $\delta = 75\%$, $\mu = 0.2$, $\nu = 0.4$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

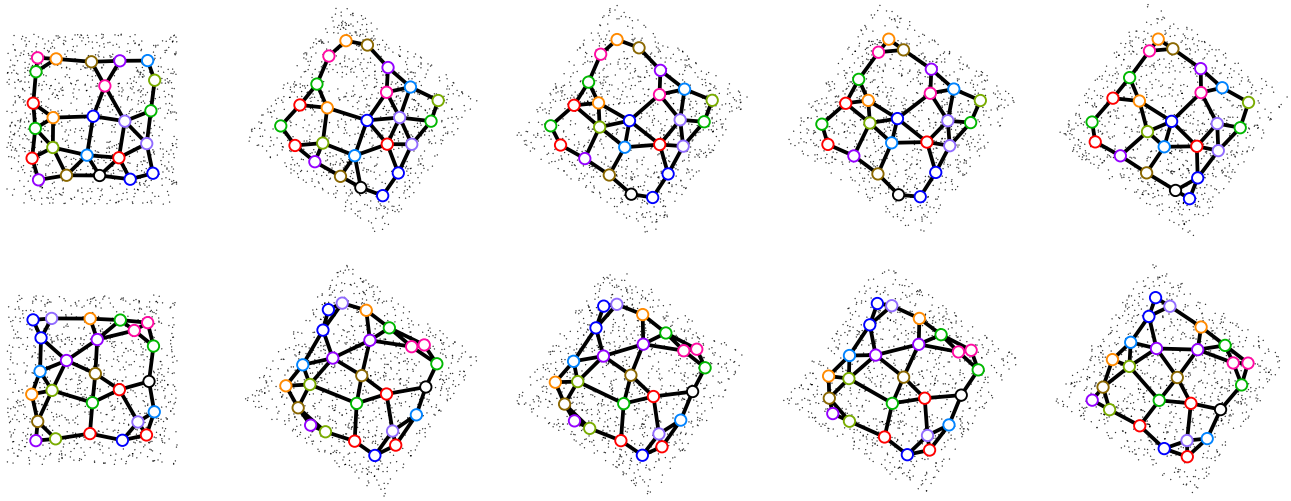

Fig. 11. Evolution of GNG-T prototypes when p^t rotates abruptly. The target T is kept constant during the transition. The top line shows the graph adaptation when topology adaptation is enabled. It is disabled at the bottom line.

Procedure 10 `compute_evolution`(G)

Require: min and max are persistent variables, updated by successive calls of this algorithm.

$(k, l) \leftarrow$ `shortest_confidence_interval`$(\{v_e\}_{v\in G}, \delta)$
min $\leftarrow$ min $+ \nu(k -$ min$)$ // $\nu \in [0, 1]$.
max $\leftarrow$ max $+ \nu(l -$ max$)$
$\Delta \leftarrow$ max $-$ min
$a \leftarrow$ min $+ \mu\Delta$ // $\mu \in [0, .5[$, [min, max] shrinks to $[a, b]$.
$b \leftarrow$ min $+ (1 - \mu)\Delta$
if $N.T < a$ **then**
  return `Add`
else if $b < N.T$ **then**
  return `Remove`
else
  return `None`
end if

non-stationary distribution. Indeed, the stabilization proposed in Section 4 removes spurious motions of the prototypes, so that the remaining prototype movements from one epoch to another reflects reliably the input distribution variation. The velocity field (Hildreth, 1984) is the distribution of the speed vectors over a moving object. It is difficult to retrieve when the object is perceived through some apparatus sampling the time. The usual case is the motion perceived by a video sensor. In that case, the moving object is the 2D distribution of pixels, whose velocity field is referred to as the optical flow. It may differ from the projection of the object velocity-field onto a 2D image.[3] Even when only the optical flow is concerned, without any attempt to retrieve the real velocity field of the perceived object from it, the computation has to face an ill-posed problem. Indeed, the pixel evolution has to be deduced from successive frames and the variation from one frame to the next could be explained by many velocity fields. This ambiguity is the origin of the *aperture problem* (Nakayama & Silvermann, 1988), detailed further. To cope with that ambiguity, most optical flow techniques add extra hypotheses on the velocity field (smooth variation over space for example) so that the processing consists in finding a flow that fits the data the best while meeting the constraints (see a review in Baker et al., 2011).

In our approach, from the model of input proposed in Section 2.1, successive $S^N_{p^t}$ and $S^N_{p^{t'}}$ are very similar to successive frames in a video sequence, even if the model is more general than image processing. The aperture problem thus stands as well, since the velocity field cannot be deduced from the content of two successive temporal samples. As for optical flow techniques, constraints have to be added. One original contribution of our paper is to use an emerging effect of self-organizing maps (Kohonen, 2001) to this end. This effect is illustrated in the forthcoming section and then the velocity field computation is introduced and illustrated on artificial data.

[3] For example, the velocity field of a spinning screw is perpendicular to its axis, while the pixels of that screw seen on a movie move along the screw axis.

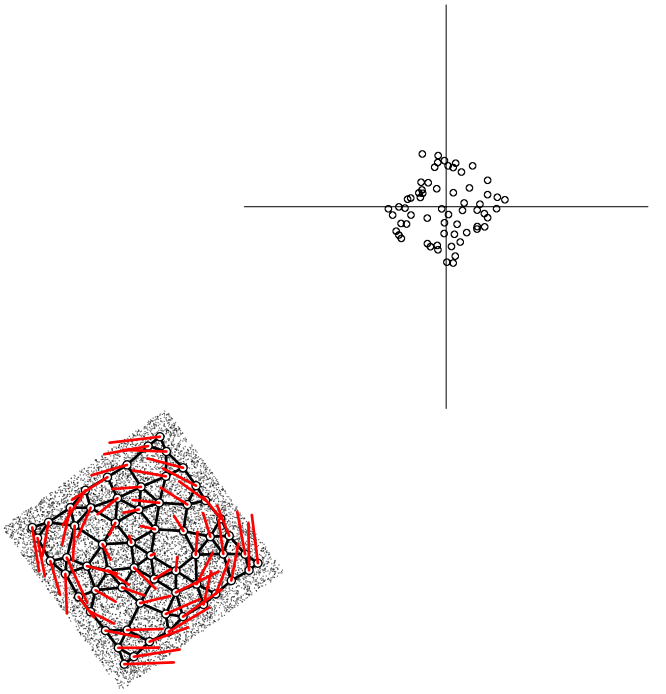

Fig. 12. Input space is $X = [-1, 4] \times [-0.9, 0.9]$, where the square area for which $p^t(\xi) = 1$ has a 1-length side. The bottom of the figure shows the nodes as well as the velocity field (red lines). The opposite of the node speed is rather represented by the red lines for the sake of clarity. A plot of the node speed values is also given on the top of the figure. This snapshot has been taken during the rotating phase (clockwise). The velocity field is computed over the whole distribution, in accordance with the rotation. Other parameters are $N = 100\,000$, $T = 5 \times 10^{-6}$, $n_{\text{struct}} = 10$, $n_{\text{evol}} = 5$, $n_{\text{mean}} = 100$, $\delta = 75\%$, $\nu = 0.4$, $\mu = 0.2$, $\varphi = 50$, $\zeta = 0.5$, $\alpha_{\text{fast}} = 0.005$, $\alpha_{\text{slow}} = 0.0005$, $\lambda = 10^{-3}$. The values for N and n_{mean} are quite high, in order to reduce the noise. Values more compliant with real-time computation are used in Section 6. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

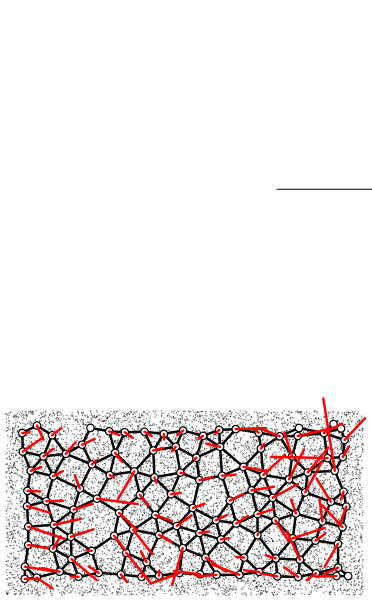

Fig. 13. Same experiment as Fig. 12, but the snapshot is taken at the end of the expanding stage. The speeds are distributed along the horizontal axis, in accordance with the horizontal expansion velocity field.

5.1. Structure-based temporal smoothing

In Section 2.1, the proposed model of dynamical input distribution considers the sampling of time. At each time sample t, the input is modeled by p^t, from which $S^N_{p^t}$ is constructed. Let us consider two successive samples t and t'. If the sampling rate is low, the consecutive p^t and $p^{t'}$ density functions may differ significantly, even if p^t depends continuously on t. The topology preservation of GNG can be used to cope with low temporal sampling rates as follows.

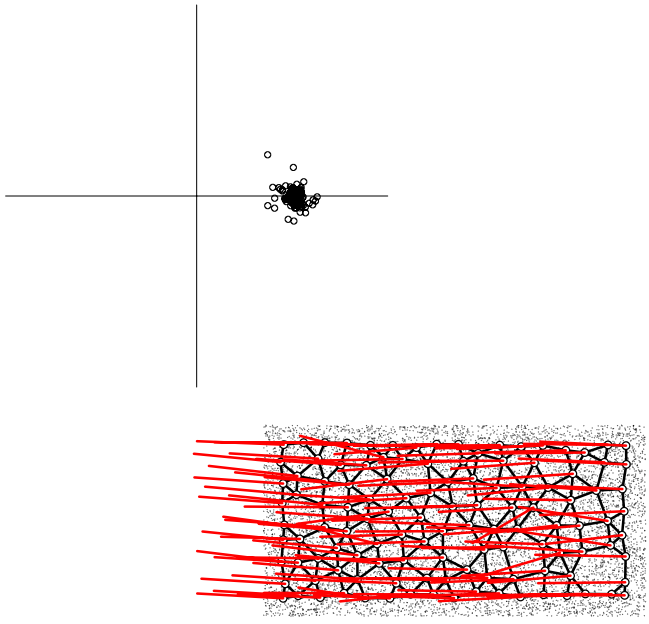

Fig. 14. Same experiment as Fig. 12, but the snapshot is taken during the translation to the right. The velocities are the same for all nodes, in accordance with the global translation.

Let us consider 2D input sets as in the previous examples. The density function p^t used from the beginning of the experiment until time t is a square, $p(\xi) = \mathbb{1}_{[-0.3,0.3]^2}(\xi)$. Then, from next t', it suddenly changes into $p^{t'}$ that is the same square, but rotated abruptly with a 35° angle. The dynamical update of the prototypes is shown at the top line in Fig. 11. When the rotation occurs, the prototypes that become closest to the corners, i.e. those which where in the middle of the square sides before the rotation, are attracted to the corners.[4] The rest of the graph adapts its topology accordingly.

Let us now restart the experiment and freeze the adaptation of the topology at time t', when the rotation occurs. This is achieved by setting dyn_topo to `false` in Procedure 3. GNG-T then behaves as a Kohonen map with the current topology (constant learning rate and direct neighbors winner-take-most) and the graph obtained for the initial p^t tries to fit the new $p^{t'}$. To do so, since no topological modification is allowed, GNG-T, behaving as a Kohonen map, rotates the graph, as bottom line in Fig. 11 shows.[5] This effect is an emerging as well as a powerful feature of Kohonen SOMs, that copes here smartly with the input variation discontinuity by preserving a structural information across time samples.

5.2. Velocity field computation

From an appropriate modeling of an input stream (Section 2.1), the control of vector quantization accuracy (Section 3) and the preservation of the structure (Section 5.1), the velocity field of a data stream can be computed. The quality of the speed measures relies on the efforts for stabilizing GNG-T (Section 4.2).

[4] See the movie http://malis.metz.supelec.fr/depot/Vq/Paper/expe-005-001.avi.
[5] See the movie http://malis.metz.supelec.fr/depot/Vq/Paper/expe-005-002.avi.

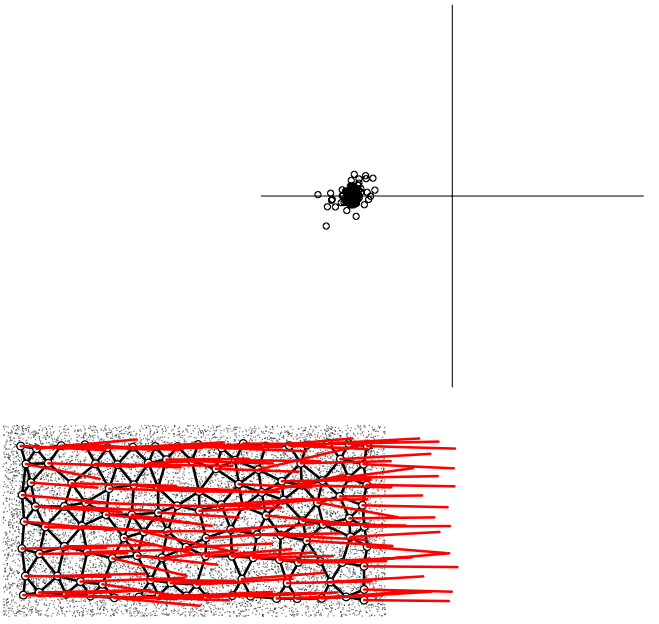

Fig. 15. Same experiment as Fig. 12, but the snapshot is taken during the translation to the left.

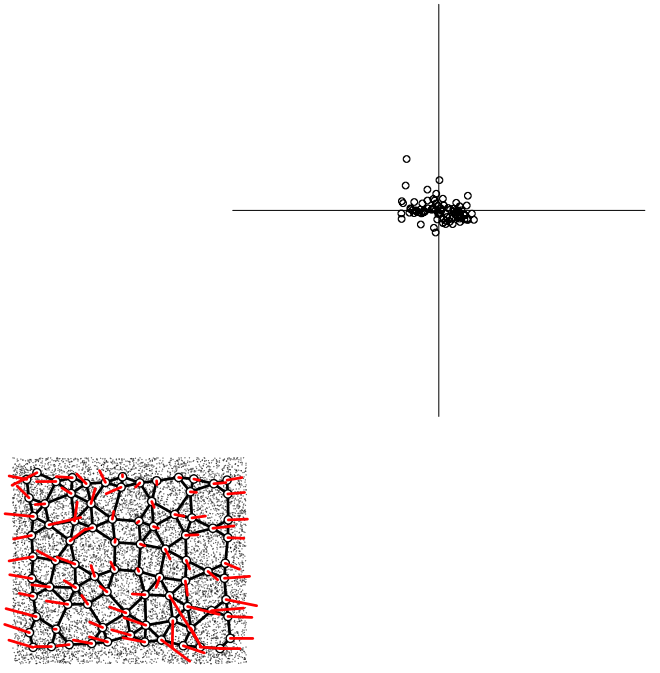

Fig. 16. Same experiment as Fig. 12, but the snapshot is taken at the end of the shrinking stage. The speeds are linearly distributed along the horizontal axis, in accordance with the horizontal shrinking velocity field.

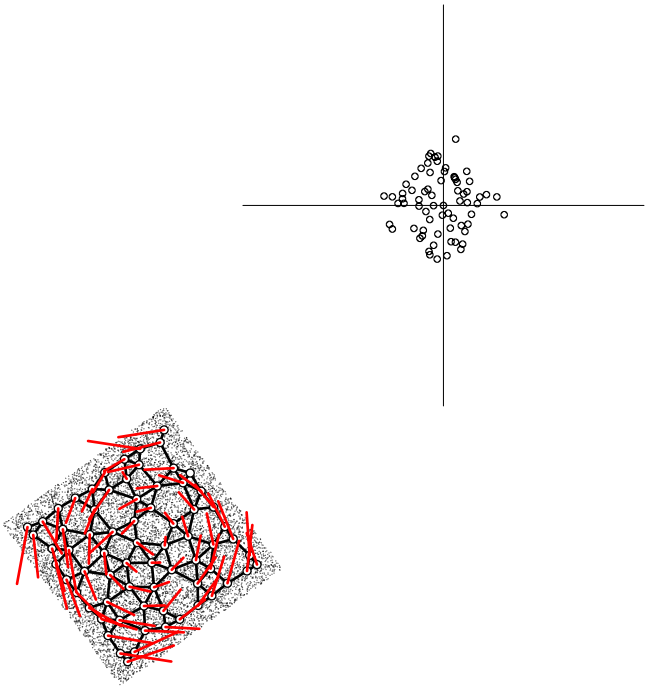

Fig. 17. Same as Fig. 12, except that Procedure 8 is used instead of 10.

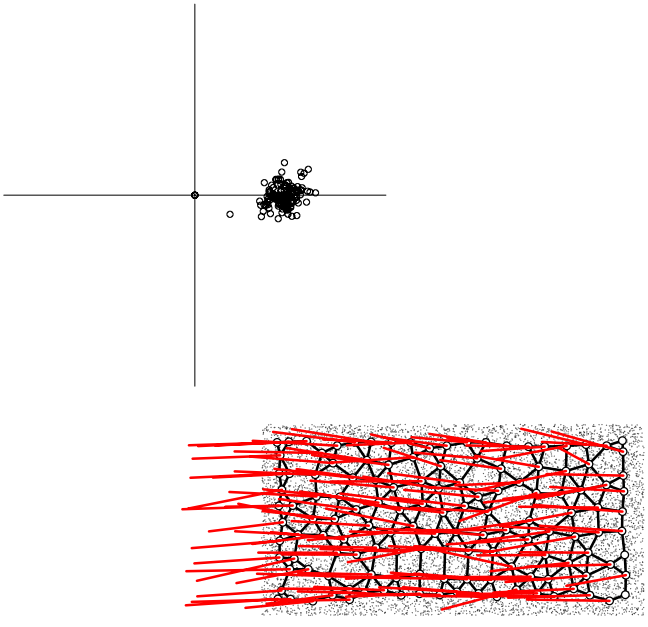

Fig. 18. Same as Fig. 14, except that Procedure 8 is used instead of 10.

The velocity field computation requires to handle supplementary data for each prototype in the graph. To the previously defined v_ω, v_e and v_n handled by a vertex v are thus added $v_{\mathrm{d}\omega}$ that stores the vertex speed, v_{-1} that stores the previous vertex position (i.e. the previous v_ω), v_Σ and $v_\#$ that are used to compute the average speed of the neighboring nodes and $v_\mathrm{s} \in \{\texttt{New}, \texttt{NoPrev}, \texttt{Ok}\}$ that is the node status, according to the different phases of the speed computation algorithm.

As Section 5.1 suggests, several epochs are required for the time instance t in order to apply a structure-based temporal smoothing. It means that Procedure 3 has to be executed several times for each instance of p^t. Indeed, the first epochs correspond to few epochs for which the topology evolution rules are frozen. Then, few supplementary ones can be used to allow for several node insertions or removals. Last, few other epochs are performed, with a freeze of the topology evolution again, in order to compute statistics of the prototypes positions. This leads to the update for each time step detailed in Algorithm 11.

In this procedure, the last chunk of epochs aims at computing the average position of each prototype. As topology evolution may cause a spurious re-arrangement of the nodes, it is blocked during this stage, as previously mentioned and a smaller learning rate is applied for a better smoothing. The mean position computation is performed by Procedure 12 (see line 11 in Procedure 11). This prototype position averaging is more reliable to estimate the speed than the current position. This is why, for each time and for each

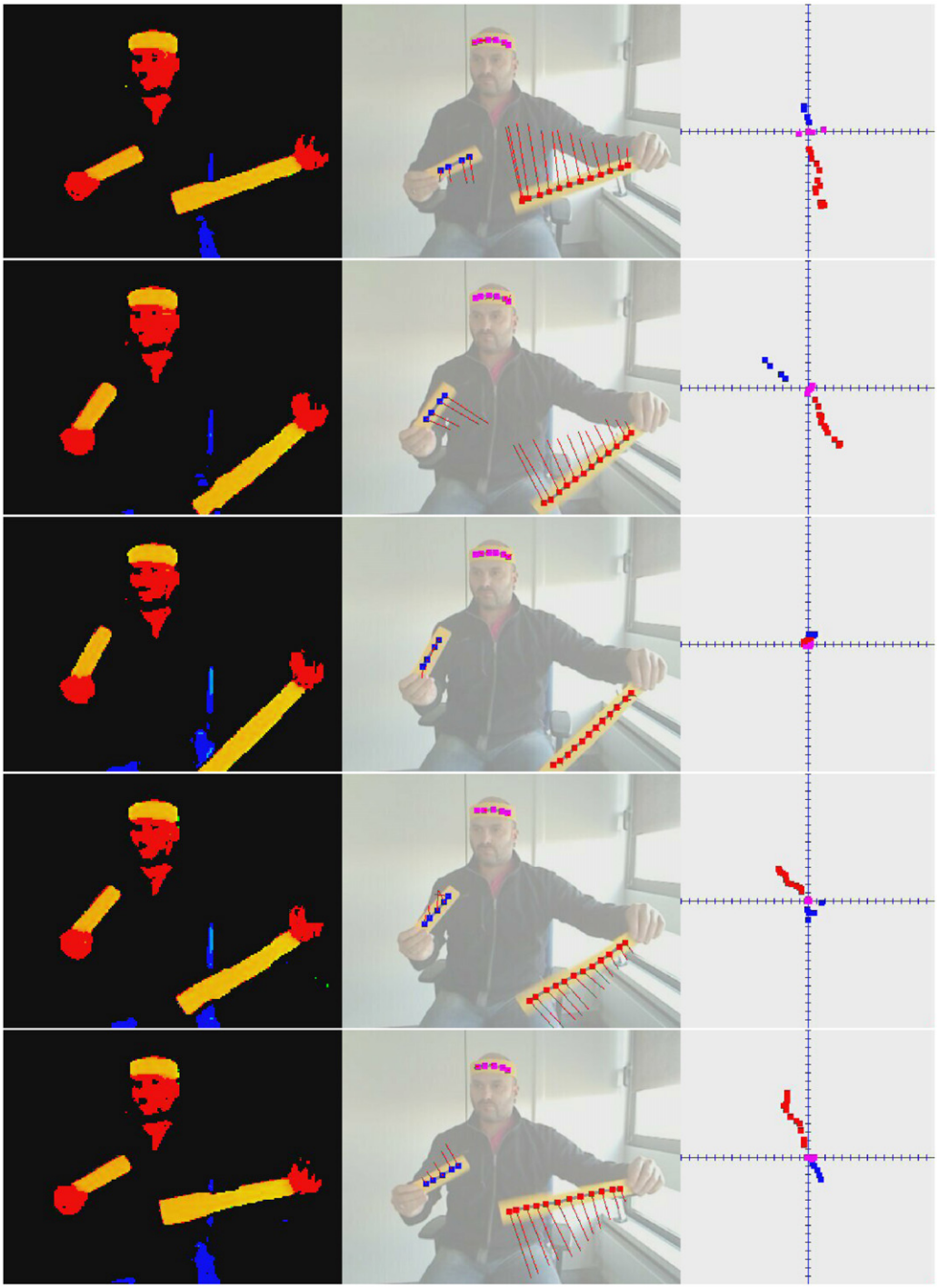

Fig. 19. Consecutive snapshots of our gstreamer plugin during a 'drum roll' gesture, taken every five frames. On the left, the color-base selection filter is shown. Only yellow pixels are submitted to the algorithm, with a maximum of `maxsamples` = 250. The center image shows the graph computed by our algorithm. Connected components are displayed with different colors. The speeds are shown with red lines. For the sake of clarity, the inverse of the speeds, re-scaled, is shown. Nevertheless, on the right, the actual speed values are plotted for each vertex, with the color of the connected component it belongs to. The velocity field is consistent with a rotation of the strips and a fixed head. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

prototype, the value of this average is compared with the one at the previous time step, in order to compute the speed of each prototype. This is what Procedure 13 does at the start of each time computation (see line 2 in Procedure 11).

To sum up, the velocity field computation consists in considering successive positions of the vertices. This is relevant since the graph "slides" when the distribution changes, so that a vertex at the next time step still represents the same part of the distribution. This sliding effect has been used in the context of video surveillance for predicting next vertex positions in order to preset the vertex positions for the next time step. This preset accelerates the learning at next time step (García-Rodríguez & García-Chamizo, 2011), even if the speeds are not accurately computed. This is particularly true in video surveillance where the distribution variations are often smooth translations of pedestrians or cars. Here, due to the structure-based temporal smoothing introduced in Section 5.1

Fig. 20. Same as Fig. 19, but with `maxsamples` = 2000. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

and the handling of speeds presented in this section, more general velocity fields can be considered and computed, as the next section shows.

5.3. Velocity field properties

The quality of the velocity field is strongly related to two features presented so far. The first feature is the stabilization of the number of nodes presented in Section 4.2. Indeed, stabilization avoids a permanent re-adjustment of the node positions. Without stabilization, the re-adjustment creates a hustle and bustle in the node population, that damages the velocity measurements. The second feature is the structure-based temporal smoothing applied at frame transitions, presented in Section 5.1. This allows to identify the velocity field even if the distribution does not change locally, thus solving the well known and previously mentioned aperture problem encountered in velocity field computation.

Let us illustrate these two features on an artificial distribution. The distribution is made of a uniform square, that first rotates. Then, it expands horizontally and becomes a rectangle. This rectangle is shifted to the right and then shifted back to the left. Last, it shrinks back horizontally to the initial squared shape. This

Procedure 11 `temporal_frame`$(G, S, \Delta t)$

1: // The temporal transition $t \rightarrow t'$ has just occurred. $\Delta t = t' - t$ and $S = S^N_{p^{t'}}$.
2: `temporal_speed`$(G, \Delta t)$
3: **for** $k \leftarrow 1$ **to** n_{struct} **do**
4: `gngt_epoch`$(G, S, \alpha_{\text{fast}}$, **false**$)$ // Structural smoothing epochs
5: **end for**
6: **for** $k \leftarrow 1$ **to** n_{evol} **do**
7: `gngt_epoch`$(G, S, \alpha_{\text{fast}}$, **true**$)$ // Evolution epochs
8: **end for**
9: **for** $k \leftarrow 1$ **to** n_{mean} **do**
10: `gngt_epoch`$(G, S, \alpha_{\text{slow}}$, **false**$)$ // Position statistics epochs
11: `prototype_mean`(G)
12: **end for**

Procedure 12 `prototype_mean`(G)

1: // For a newly created vertex v, $v_{\text{s}} =$ `New`, $v_\Sigma = 0$ and $v_\# = 0$.
2: **for all** $v \in \mathrm{V}(G)$ **do**
3: **if** $v_{\text{s}} \neq$ `New` **then**
4: $v_\Sigma \leftarrow v_\Sigma + v_\omega$
5: $v_\# \leftarrow v_\# + 1$
6: **end if**
7: **end for**

Procedure 13 `temporal_speed`$(G, \Delta t)$

1: **for all** $v \in \mathrm{V}(G)$ **do**
2: **if** $v_{\text{s}} \neq$ `Ok` **then**
3: $N \leftarrow \left\{v' \in \mathrm{V}(v) \text{ and } v'_{\text{s}} = \texttt{Ok}\right\}$ // N gathers the neighbors of v for which speed is computed.
4: $\overline{\mathrm{d}\omega} \leftarrow 1/|N| \sum_{v' \in N} v'_{\mathrm{d}\omega}$
5: **end if**
6: **if** $v_{\text{s}} =$ `New` **then**
7: $v_{\text{s}} \leftarrow$ `NoPrev`
8: $v_{\mathrm{d}\omega} \leftarrow \overline{\mathrm{d}\omega}$ // The speed is obtained from neighboring vertices.
9: $v_{-1} \leftarrow 0$
10: **else if** $v_{\text{s}} =$ `NoPrev` **then**
11: $v_{\text{s}} \leftarrow$ `Ok`
12: $v_{\mathrm{d}\omega} \leftarrow \overline{\mathrm{d}\omega}$ // The speed is obtained from neighboring vertices.
13: $v_{-1} \leftarrow v_\Sigma / v_\#$ // $v_\Sigma / v_\#$ is the average position of v_ω during last non-transient epochs.
14: **else if** $v_{\text{s}} =$ `Ok` **then**
15: $v_{\mathrm{d}\omega} = \left(v_\Sigma / v_\# - v_{-1}\right) / \Delta t$
16: $v_{-1} \leftarrow v_\Sigma / v_\#$
17: **end if**
18: $v_\Sigma \leftarrow 0$, $v_\# \leftarrow 0$
19: **end for**

cycle is repeated three times.[6] Snapshots of this dynamical process are shown in Figs. 12–16. In these figures, it can be noticed that the velocity field is retrieved for all the nodes, even for the ones lying at the center of the distribution, where no local change is visible. For example, Fig. 14 shows a distribution where the point density only varies at the vertical edges of the rectangle but for which a correct uniform velocity field over all the vertices is reconstructed.

In order so illustrate the stabilization effect induced by Procedure 10 invoked at line 12 of Procedure 7, let us replace Procedure 10 by Procedure 8. The process in thus very close to the former version of GNG-T proposed in Frezza-Buet (2008). Figs. 17 and 18, as well as the video where the effect is more sensitive,[7] show the lack of precision of the velocity field retrieval, supporting

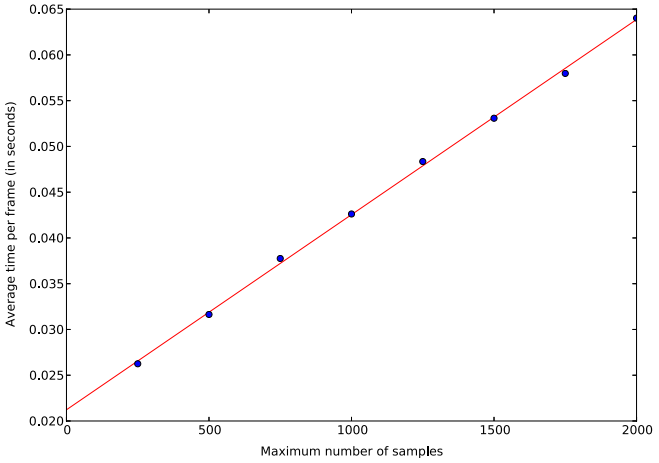

Fig. 21. Average frame computation time of our gstreamer plugin, when included in a pipeline and fed with the benchmark movie used in Figs. 19 and 20, according to the `maxsamples` parameter.

the relevance of the improvement presented in this paper. This is also consistent with Figs. 8 and 10.

6. Video stream application

In this section, let us consider the context of real-time video stream analysis. This requires an efficient computation of GNG-T, that has been implemented by a dedicated memory management in the library we provide.[8] Even if efficiency consideration is not the point of this paper, let us mention that recent works stress the relevance of implementing Growing Neural Gas methods efficiently (Fišer, Faigl, & Kulich, 2013; García-Rodríguez et al., 2012). Only plots of time consumption are given here, without details about implementation issues.

6.1. Extracting samples from video frames

Let us implement a color-based pixel selection (see yellow pixels on the left frames in Fig. 19). For each frame, the input set is made of the coordinates of the selected (yellow) pixels. Let us then consider the formalization introduced in Section 2.1. The density function p is the function returning 1 for the position of the selected pixels and 0 otherwise. Providing the algorithm with *all* the selected positions is qualitatively similar to setting $N = w \times h$ in Algorithm 1, with (w, h) being the video frame dimension. As a consequence, if only a proportion α of the selected pixels are submitted to the algorithm, for efficiency reasons, it is as if $N = \alpha \cdot (w \times h)$ were used in Algorithm 1. Thus, one needs to specify the desired quantization accuracy by setting the target value T and determine the value N from the proportion of the selected pixels actually used. Reducing this proportion allows to speed up the algorithm (and thus increase the framerate).

We have implemented a gstreamer[9] video plugin in order to test our application. The movie size is $(w, h) = (400, 300)$. The plugin grabs the frame as fast as it can and considers the measured time delay from the last frame to the current one as the Δt value for Procedure 12. The plugin also uses a `maxsamples` parameter, that is the maximum number of input samples submitted to the algorithm. If there are more than `maxsamples` selected samples in the current frame, only `maxsamples` are used. The value N

[6] See the movie http://malis.metz.supelec.fr/depot/Vq/Paper/expe-005-003.avi.
[7] See the movie http://malis.metz.supelec.fr/depot/Vq/Paper/expe-005-004.avi.
[8] http://malis.metz.supelec.fr/spip.php?article182.
[9] www.gstreamer.net.

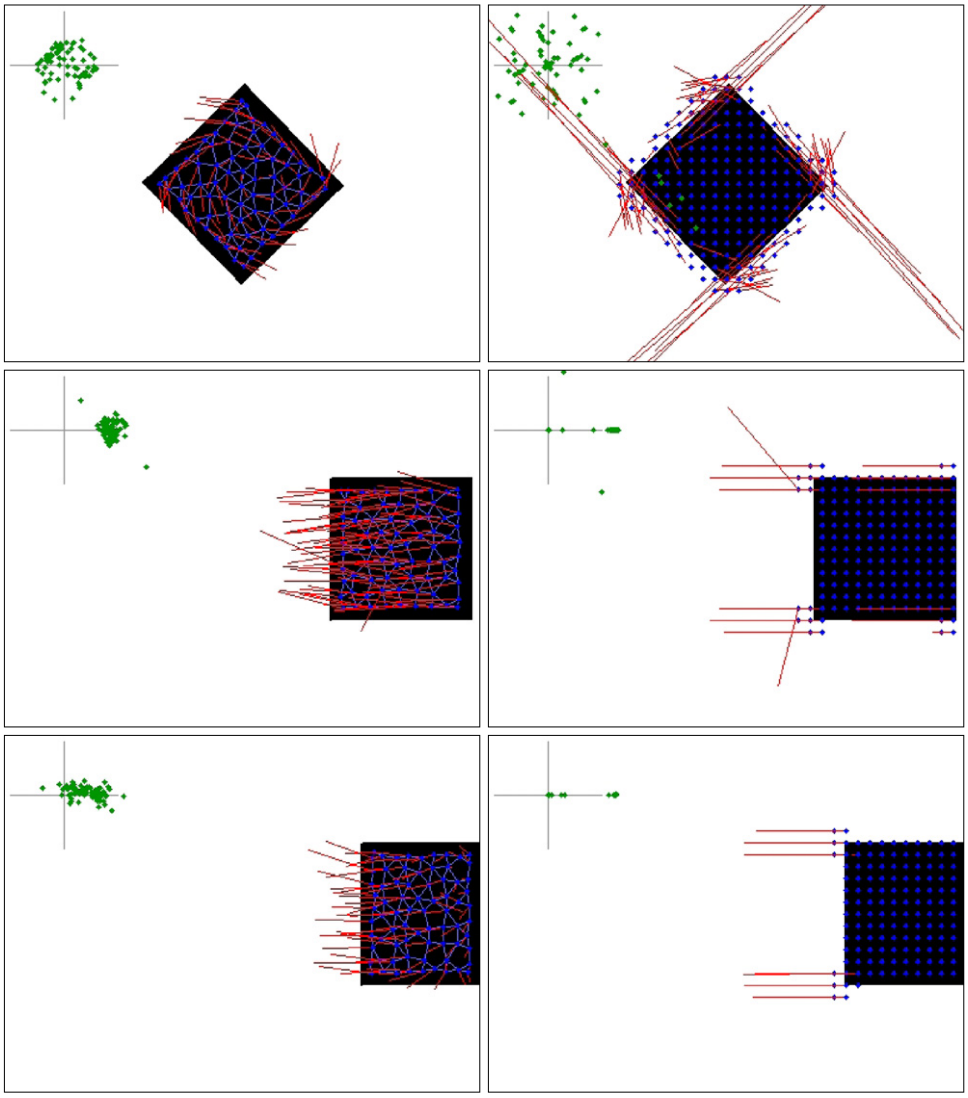

Fig. 22. Comparison between GNG-T (left) and Farneback optical flow (right) on a moving square 25 fps movie. Snapshots are taken at the middle of the rotation sequence (top), the end of the translation to the right (middle) and first steps after the start of the exit of the camera's range (bottom).

actually used in Procedure 10 is adapted accordingly. As explained before,

$$N = \frac{\min(\texttt{maxsamples}, \texttt{size})}{\texttt{size}} \times w \times h \tag{8}$$

where `size` is the number of selected (yellow) pixels in the current frame.

The `maxsamples` parameter has influence on both the computation time and the quality of the result. It is measured from a recorded movie and not from an online webcam video stream. In that benchmarking process, the movie frames are provided one by one to the plugin and an arbitrary constant value $\Delta t = 75$ ms is considered for the plugin computation whatever the time really spent for the frame processing. The results for `maxsamples` $=$ 250 and `maxsamples` $=$ 2000 are qualitatively the same,[10,11] as Figs. 19 and 20 show. As in Section 5.3, we have also run the `maxsamples` $=$ 250 experiment with Procedure 8 instead of Procedure 10 in order to compare with the former version of GNG-T. The movie[12] actually shows a degradation of the velocity field extraction quality. On the computer used for the test, an Intel Core i5 CPU 650 4 × 3.20 GHz with a 3.8Go RAM running under a Fedora 18 Linux distribution, the performances of our implementation are plotted in Fig. 21. It shows that the `maxsamples` parameter affects the efficiency in a linear way, the offset to the origin being due to the gstreamer overhead mainly.

Other parameters are $n_{\text{struct}} = 10$, $n_{\text{evol}} = 5$, $n_{\text{mean}} = 50$ for Procedure 11, $\delta = 0.75$, $\nu = 0.4$, $\mu = 0.2$, $T = 0.1$ for Procedure 10, N is determined from Eq. (8) for each frame, $\varphi = 50$, $\zeta = 0.5$, $\alpha_{\text{fast}} = 0.005$, $\alpha_{\text{slow}} = 0.001$, $\lambda = 0.001$ for Algorithms 5 and 7.

6.2. Comparison with optical flows

Even if the method introduced in this paper is not specific to video processing, it can be compared to optical flow when it is applied to video frames. Let us perform this comparison on the basis of a video made of binary images, such that the distribution

[10] See the movie http://malis.metz.supelec.fr/depot/Vq/Paper/movie-velocity-250-evol0.mpeg.

[11] See the movie http://malis.metz.supelec.fr/depot/Vq/Paper/movie-velocity-2000-evol0.mpeg.

[12] See the movie http://malis.metz.supelec.fr/depot/Vq/Paper/movie-velocity-250-evol1.mpeg.

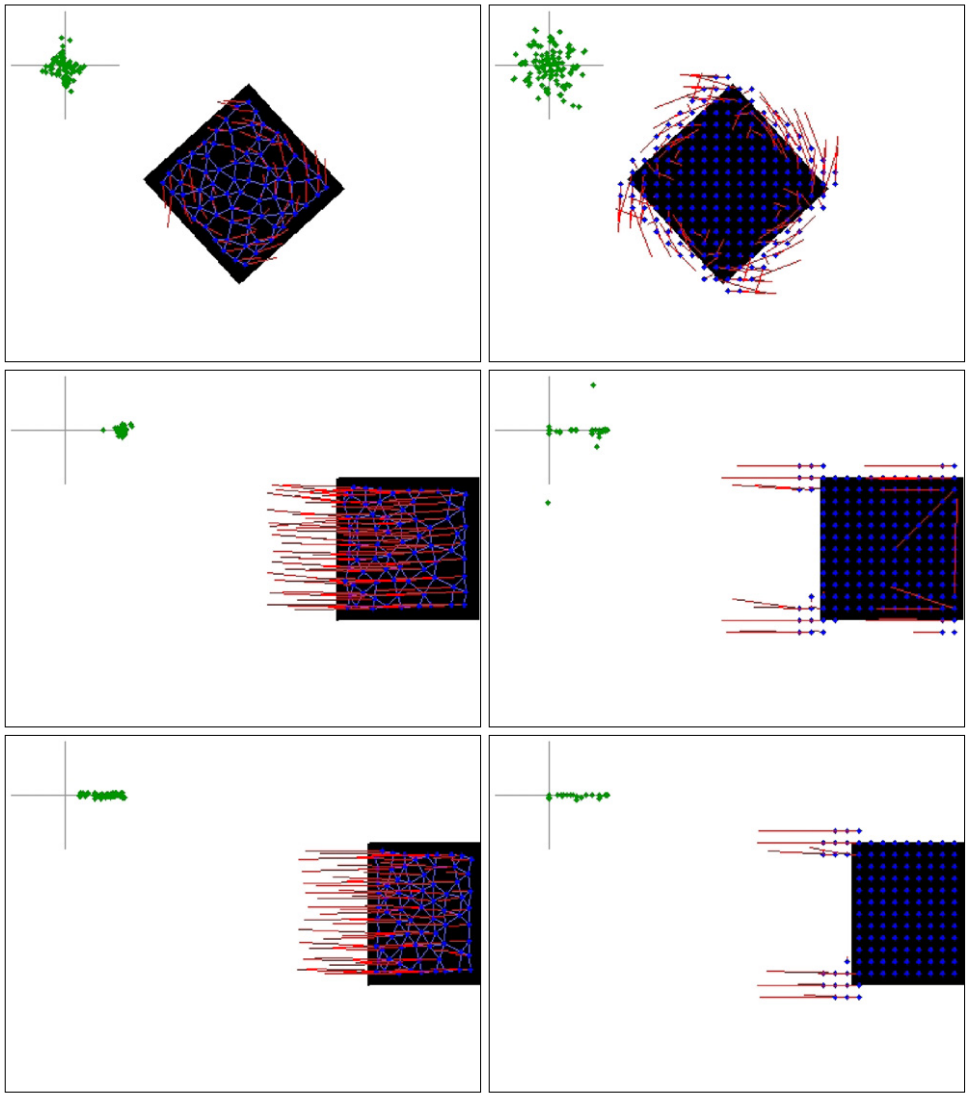

Fig. 23. Same as Fig. 22, but with a 5 fps movie.

of black pixels is similar to the experiment in Section 5.3. The video size is 400 × 300. It shows a horizontal square appearing progressively from the left and scrolled until the center of the image. Then, a quarter turn is performed and the square is scrolled on the right until it disappears. The high framerate (25 fps) video for this scenario is made of 300 frames (100 for each left, rotate and right part), whereas a lower framerate (5 fps) video is made of one frame over five.

The optical flow algorithm used for the experiments is provided by `openCV`.[13] It is an implementation of the method described in Farneback (2003). The flow is computed at every pixel of the image, but it is displayed only at pixels belonging to a 40 × 30 grid over the image. Low speed pixels on the grid are ignored, except if they belong to black regions in the image. Parameters are `pyr_scale` = 0.5, `levels` = 3, `winsize` = 15, `iterations` = 3, `poly_n` = 5 and `poly_sigma` = 1.2 (see the `openCV` documentation). The parameters for GNG-T are similar as previously for most of them. More epochs are used for structural smoothing since low framerates are considered in the experiments. However, the total number of epochs is kept unchanged. Parameters are thus $n_{\text{struct}} = 20$, $n_{\text{evol}} = 5$, $n_{\text{mean}} = 40$, $\delta = 75\%$, $\nu = 0.4$, $\mu = 0.2$, $T = 5 \times 10^{-2}$, $\varphi = 50$, $\zeta = 0.5$, $\alpha_{\text{fast}} = 0.005$, $\alpha_{\text{slow}} = 0.001$, $\lambda = 10^{-3}$, `maxsamples` = 2000.

The behavior of GNG-T and the optical flow are shown in Fig. 22 (high framerate[14,15]) and Fig. 23 (low framerate[16,17]). One can see that the optical flow fails to reconstruct the velocity field inside black areas, in both movies, whereas GNG-T copes with the lack of velocity information in these areas. When the square gets out of the camera range, the distribution of pixels behaves as if the square were shrinking, which may be considered as wrong. The speed profile (see bottom left frames in Figs. 22 and 23) is similar to Fig. 16.

However, optical flows exploit the shade distribution over images and thus using binary images with wide monochrome regions impairs their performance. Using gray-scale images does

[13] http://opencv.org.

[14] See the movie http://malis.metz.supelec.fr/depot/Vq/Paper/EF-01-gngt-movie.avi.

[15] See the movie http://malis.metz.supelec.fr/depot/Vq/Paper/EF-01-flow-movie.avi.

[16] See the movie http://malis.metz.supelec.fr/depot/Vq/Paper/EF-02-gngt-movie.avi.

[17] See the movie http://malis.metz.supelec.fr/depot/Vq/Paper/EF-02-flow-movie.avi.

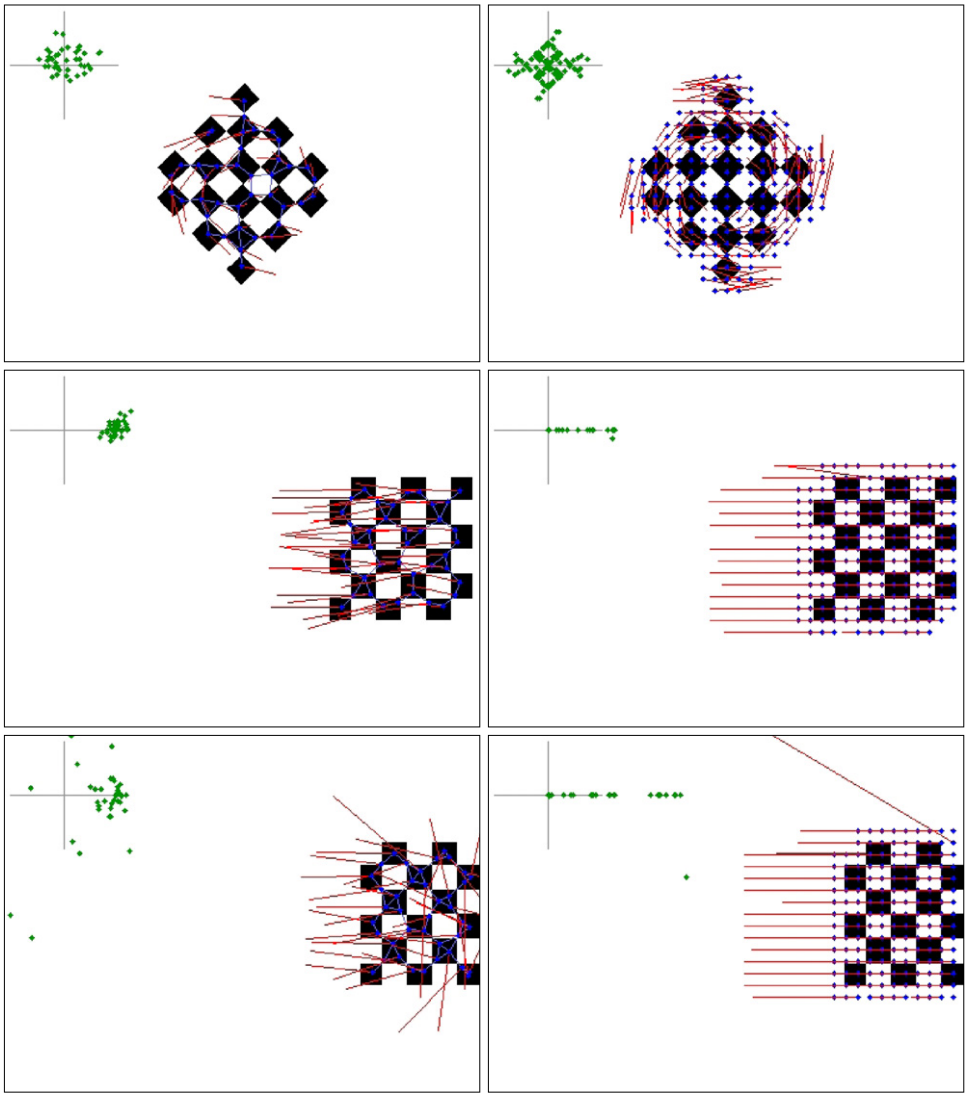

Fig. 24. Same as Fig. 22, with a checker board textured moving object.

not fit the pixel selection process presented in Section 6.1, so binary images are required for comparing the two methods. Instead of using richer gray-scaled images, let us use a binary checker board as a moving object, in order to provide the optical flow with local texture information. The result is shown in Fig. 24 (high framerate[18,19]) and Fig. 25 (low framerate[20,21]). With such a textured object, the optical flow and GNG-T behave in a similar way. GNG-T takes benefits from the checker board distribution as well since it stabilizes GNG-T vertices positions, that are more constrained. It is interesting to notice that both algorithms, with both moving objects, are more accurate in the low framerate movie.

For all the experiments, the time consumption for both algorithms is similar with the used parameter, the computer being the same as the one used for the experiments in Section 6.1. The experiments have been implemented in `opencv` and our library, without CPU acceleration. The measure of time, shown in Table 1, is taken just before the application of the algorithms, i.e. where the computation differs between the two codes.

Table 1
Comparative average duration of a single frame computation.

Texture	Framerate (fps)	Duration (ms)	
		Optical flow	GNG-T
Black	25	117	134
Black	5	109	132
Checker	25	97	81
Checker	5	96	78

7. Conclusion

This paper extends a previous work on the vector quantization of non-stationary distributions for the retrieval of the velocity field induced by the distribution evolution. By doing so, a definition of a scalar quantization accuracy is given. As opposed to other approaches where the quantization accuracy is determined empirically, this scalar has an identified geometrical meaning, allowing a setting a priori according to the application. The

[18] See the movie http://malis.metz.supelec.fr/depot/Vq/Paper/EF-03-gngt-movie.avi.
[19] See the movie http://malis.metz.supelec.fr/depot/Vq/Paper/EF-03-flow-movie.avi.
[20] See the movie http://malis.metz.supelec.fr/depot/Vq/Paper/EF-04-gngt-movie.avi.
[21] See the movie http://malis.metz.supelec.fr/depot/Vq/Paper/EF-04-flow-movie.avi.

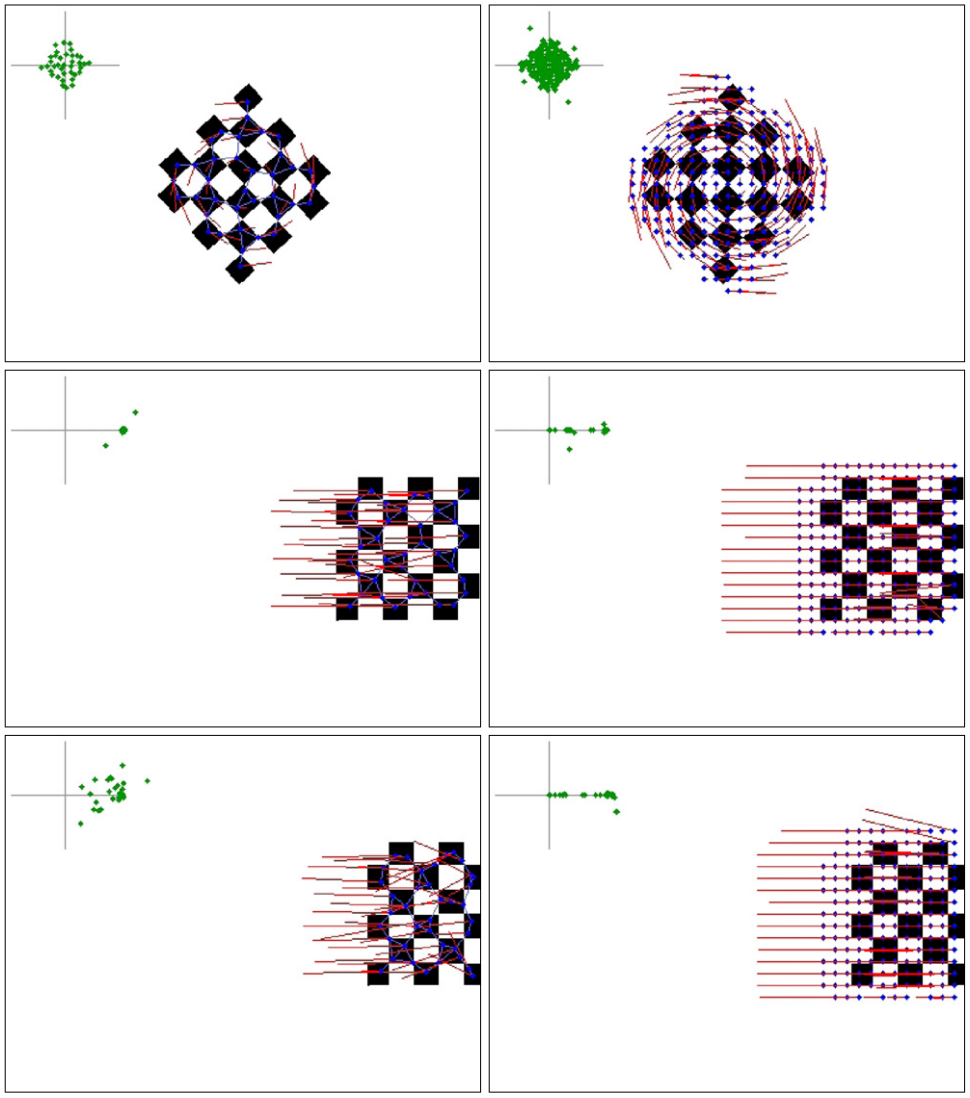

Fig. 25. Same as Fig. 24, but with a 5 fps movie.

quantization accuracy is central in most of the GNG-based approaches since this is the argument underlying the tuning of the appropriate number of prototypes. Here, it is used to continuously monitor this number while the input distribution changes. Moreover, modeling the non-stationary distribution with a rejection sampling process reconsidered at each time step unveils the role of the parameter N in Algorithm 1 that is actually not the number of input samples. It allows for adjusting this number according to the time available for computation, without changing the quality of the quantization, as illustrated for example by the `maxsample` parameter in the video application.

The paper shows that having such a prototype monitoring process is not sufficient when the goal is to retrieve the velocity field, since once the appropriate number of prototypes for the desired accuracy is reached, stability issues have to be considered. The fluctuations of this number around the appropriate value have actually dramatic effects on the velocity measurements since it induces significant spurious prototype motions. This is why a specific effort to tackle this problem is made in this paper.

The speed retrieval performed in our GNG-based approach relies on the comparison between two consecutive time steps. As discussed, this process is marred by the aperture problem and an original structure-based temporal smoothing solution, based on an emergent property of self-organized maps, is proposed here. When applied to a stream of binary images, the GNG-T approach meets the processing of optical flows. Nevertheless, both of them rely on dramatically different paradigms. On the one hand, optical flows compute a vector field from the variations of the image intensity distribution over the image surface. This vector field is a continuous object by nature (in spite of its sampling at each pixels). On the other hand, the graph computed by GNG-T, even when decorated with velocity information, remains a discrete object. By the way, the discrete nature of the GNG-T graph has been introduced in this paper as a motivation for addressing the anchoring problem, which is not claimed to be the purpose of optical flows. However, the experiments presented here on binary images, where both methods can be applied, show very close time consumption and performances, except for non-textured images where, naturally, optical flows are penalized. It shows that both algorithms are suitable for a real-time computation.

In light of the results presented so far, future extension of the approach should be considered, concerning a further improvement of the prototype position stabilization. Indeed, it appears to be the core problem when data is made of big contiguous and homogeneous spatially extended regions. Indeed, high framerate and thus slight motion from one time step to the next, combined with wide homogeneous distribution (see Fig. 22 compared with Fig. 25) appears to be the most difficult distribution stream for

GNG-T. An ongoing study, not reported here, suggests that the instability of edge updating,[22] in spite of the stability of the number of prototypes, is responsible for frequent prototype motions when edges are added or removed sporadically. These motions propagate in a wave-like manner while the graph readjusts to the data, especially in the aforementioned extended static homogeneous regions. Reducing this effect may allow for less epochs for each frame, since the prototype positions may be more stable. This has to be confirmed and solved in future work.

Last, let us stress that the strategy applied here for monitoring the number of prototype can be applied to other incremental algorithms, as Growing Grid (Fritzke, 1995b) for example. The edge related instability issues mentioned above should not occur since the edge creation is only driven by the number of prototypes and not by the local (and quite unstable) competitive Hebbian learning process used in GNG. The monitoring proposed here, as well as the computation of the velocity field, can thus be extended to main vector quantization techniques, allowing for a wide range of applications where non-stationary data are considered, without any restriction to the context of image processing.

References

Andrieu, C., de Freitas, N., Doucet, A., & Jordan, M. I. (2003). An introduction to MCMC for machine learning. *Machine Learning*, *50*(1), 5–43.

Baker, S., Scharstein, D., Lewis, J., Roth, S., Black, M. J., & Szeliski, R. (2011). A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, *92*, 1–31.

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

Chao, H., Gu, Y., & Napolitano, M. (2014). A survey of optical flow techniques for robotics navigation applications. *Journal of Intelligent & Robotic Systems*, *73*(1–4), 361–372.

Coradeschi, S., & Saffiotti, A. (2003). An introduction to the anchoring problem. *Robotics and Autonomous Systems*, *43*, 85–96.

Cselényi, Z. (2005). Mapping the dimensionality, density and topology of data: The growing adaptive neural gas. *Computer Methods and Programs in Biomedicine*, *78*(2), 141–156.

Farneback, G. (2003). Two-frame motion estimation based on polynomial expansion. In *Lecture notes in computer science. Vol. 2749* (pp. 363–370).

Fišer, D., Faigl, J., & Kulich, M. (2013). Growing neural gas efficiently. *Neurocomputing*, *104*, 72–82.

Frezza-Buet, H. (2008). Following non-stationary distributions by controlling the vector quantization accuracy of a growing neural gas network. *Neurocomputing*, *71*(7–9), 1191–1202.

Fritzke, B. (1995a). Growing grid—a self-organizing network with constant neighborhood range and adaptation strength. *Neural Processing Letters*, *2*, 9–13.

Fritzke, B. (1995b). Growing grid—a self-organizing network with constant neighborhood range and adaptation strength. *Neural Processing Letters*, *2*, 9–13.

Fritzke, B. (1995c). A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, & T. K. Leen (Eds.), *Advances in neural information processing systems. Vol. 7* (pp. 625–632). Cambridge, MA: MIT Press.

Fritzke, B. (1997). A self-organizing network that can follow non-stationary distributions. In *ICANN'97: international conference on artificial neural networks* (pp. 613–618). Springer.

Fujita, K. (2013). Extract an essential skeleton of a character as a graph from a character image. *International Journal of Computer Science Issues*, *10*(1).

García-Rodríguez, J., Angelopoulou, A., García-Chamizo, J. M., Psarrou, A., Escolano, S. O., & Giménez, V. M. (2012). Autonomous growing neural gas for applications with time constraint: Optimal parameter estimation. *Neural Networks*, *32*, 196–208.

García-Rodríguez, J., & García-Chamizo, J. M. (2011). Surveillance and human–computer interaction applications of self-growing models. *Applied Soft Computing*, *11*, 4413–4431.

Guenther, W. C. (1969). Shortest confidence intervals. *The American Statistician*, *23*(1), 22–25.

Hildreth, E. C. (1984). The computation of the velocity field. *Proceedings of the Royal Society B*, *22*, 189–220.

Kohonen, T. (2001). *Self-organizing maps*. Springer.

Linde, Y., Buzo, A., & Gray, R. M. (1980). Algorithm for vector quantization design. *IEEE Transactions on Communications systems*, *28*(1), 84–95.

Marsland, S., Shapiro, J., & Nehmzow, U. (2002). A self-organising network that grows when required. *Neural Networks*, *15*(8–9), 1041–1058.

Martinez, T. M., & Schulten, K. J. (1994). Topology representing networks. *Neural Networks*, *7*(3), 507–522.

Nakayama, K., & Silvermann, G. H. (1988). The aperture problem-II. Spatial integration of velocity information along contours. *Vision Research*, *28*(6), 747–753.

Patra, B. (2011). Convergence of distributed asynchronous learning vector quantization algorithms. *Journal of Machine Learning Research*, *12*, 3431–3466.

Qin, A., & Suganthan, P. (2004). Robust growing neural gas algorithm with application in cluster analysis. *Neural Networks*, *17*, 1135–1148.

Stergiopoulou, E., & Papamarkos, N. (2009). Hand gesture recognition using a neural network shape fitting technique. *Engineering Applications of Artificial Intelligence*, *22*(8), 1141–1158.

Tence, F., Gaubert, L., Soler, J., De Loor, P., & Buche, C. (2013). Stable growing neural gas: a topology learning algorithm based on player tracking in video games. *Applied Soft Computing*, *13*(10), 4174–4184.

[22] See line 18 in Procedure 5 and line 22 in Procedure 7.