Elsevier Editorial System(tm) for Neural Networks
Manuscript Draft

Manuscript Number: NEUNET-D-14-00125

Title: Online Computing of Non-Stationary Distributions Velocity Fields by an Accuracy Controlled Growing Neural Gas

Article Type: Article

Section/Category: Learning Systems

Keywords: Vector Quantization; Growing Neural Gas; Velocity Field

Corresponding Author: Dr. Herve Frezza-Buet, Ph.D.

Corresponding Author's Institution: Supélec

First Author: Herve Frezza-Buet, Ph.D.

Order of Authors: Herve Frezza-Buet, Ph.D.

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 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.

Online Computing of Non-Stationary Distributions Velocity Fields by an Accuracy Controlled Growing Neural Gas

Hervé Frezza-Buet, PhD, Associate Professor

Information, Multimodality and Signal
Suplec, UMI 2958 Georgia Tech / CNRS,

Suplec
2, rue Édouard Belin,
F-57070 Metz,
France

herve.frezza-buet@supelec.fr

Online Computing of Non-Stationary Distributions Velocity Fields by an Accuracy Controlled Growing Neural Gas

Hervé Frezza-Buet[a,b]

[a] *Supélec, 2, rue Édouard Belin, 57070 Metz, France*
[b] *Georgia Tech Lorraine, UMI 2958, 2-3, rue Marconi, 57070 Metz, France*

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 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.

Keywords: Vector Quantization, Growing Neural Gas, Velocity Field

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 in 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 effortless 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 and Saffiotti (2003), that every robotic engineer as 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. Some 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 concern stationary input distributions, but some modifications of the algorithms have been proposed in order to cope with non-stationarity Fritzke (1997); Frezza-Buet (2008). This consists of the design of algorithms which are robust to some change 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 as well as its variation. Indeed, such temporal variation of the input may contain the relevant cues for understanding the semantics of the input. This is for example what highlight the approaches based on optical flows in computer vision Chao et al. (2014).

The paper is organized as follows. In section 2 are introduced general notations, as well as a way to model of the input data. This model enables a *quantitative interpretation* of the parameters of our algorithm. Fitting some 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 distortion* that measures the quality of some vector quantization process. Relying on the Voronoï distortion 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 distortion 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. 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 for the sake of addressing broader application domains.

2. Notations and properties

2.1. Modeling a non stationary input

Let us denote by X a bounded input set where input samples $\xi \in X$ are taken. 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.

$S_p^N \leftarrow \emptyset$
for $i = 1$ **to** N **do**
  $\xi \sim \mathcal{U}_X,\ u \sim \mathcal{U}_{[0,1[}$
  if $u < p(\xi)$ **then**
    $S_p^N \leftarrow S_p^N \cup \{\xi\}$
  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 et al. (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 modelled 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 distortion

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 some distortion function. Classical definitions of vector quantization concepts can be found in Patra (2011), where densities of probability are concerned. Let us here recall these definitions in the restricted case of a finite input sample set S rather than distributions. Let us name $\Omega_\kappa = \{\omega_1, \omega_2, \cdots \omega_\kappa\} \in X^{|\kappa|}$ 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 $\mathcal{E}_{\Omega_\kappa}^S$ induced by Ω_κ on S is

$$\mathcal{E}_{\Omega_\kappa}^S = \frac{1}{|S|}\sum_{\xi\in S} d\left(\xi, \omega_\xi^\star\right) \tag{1}$$

where d is some similarity measure, typically the squared Euclidan distance, and $\omega_\xi^\star = \operatorname{argmin}_{\omega\in\Omega_\kappa} d(\xi,\omega)$. The goal of most vector quantization algorithms is to find $\Omega_{\kappa,S}^\star = \operatorname{argmin}_{\Omega_\kappa\in X^{|\kappa|}} \mathcal{E}_{\Omega_\kappa}^S$.

Let us note $V_\omega^S = \left\{\xi \in S \,|\, \omega_\xi^\star = \omega\right\}$ the samples of S belonging to the Voronoï region of prototype ω. V_ω^S is a finite set, referred to as a *Voronoï cell* in the following. It is obvious that $\left\{V_\omega^S\right\}_{\omega\in\Omega_\kappa}$ is a partition of S. Let us define the *Voronoï distortion* $\mathcal{V}_\omega^S$, i.e. the contribution to some Voronoï cell to the global distortion $\mathcal{E}_{\Omega_\kappa}^S$, as

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

It is obvious that

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

The Voronoï distortion 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 prototype attain some kind of equilibrium where the errors between samples and prototypes are equally shared. This share is a common Voronoï distortion value, that represents the *quantization accuracy*. In the GNG algorithm by Fritzke Fritzke (1995b), 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 distortion introduced here.

Let us illustrate this on an artificial example. Let $X = [-0.5, 0.5]^2$ and $p(\xi) = \mathbb{1}_{\xi_y \leq 0} \times (0.2 \times \mathbb{1}_{\xi_x \leq 0} + \mathbb{1}_{0 < \xi_x}) + \mathbb{1}_{0 < \xi_y} \times (\xi_x + 0.5)$. Let us set up $S = S_p^{50000}$ 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 et al. (1980). In figure 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 distortion. 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 distortion values among the prototypes. In figure 2, the mean prototype error, i.e $\frac{1}{|V_\omega^S|} \sum_{\xi \in V_\omega^S} d(\xi, \omega)$ is represented. It is obvious that the values are related to the spatial extension of the V_ω^S, and the concentration of prototype at higher sample density regions is visible on the central plot in figure 2 in blue.

As opposed to figure 2, figure 1 shows that the Voronoï distortion 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 figure 1, the value $\mathcal{V}_\omega^S$ 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 figure 1), the value $\mathcal{V}_\omega^S$ is the summation of many small values.

It has been mentioned previously that the optimal prototypes reach an equilibrium where the Voronoï distortion is shared equally, whatever the local value of p around the prototypes. Figure 1 shows that this assertion is idealized, since real Voronoï distortion distribution is made of contiguous

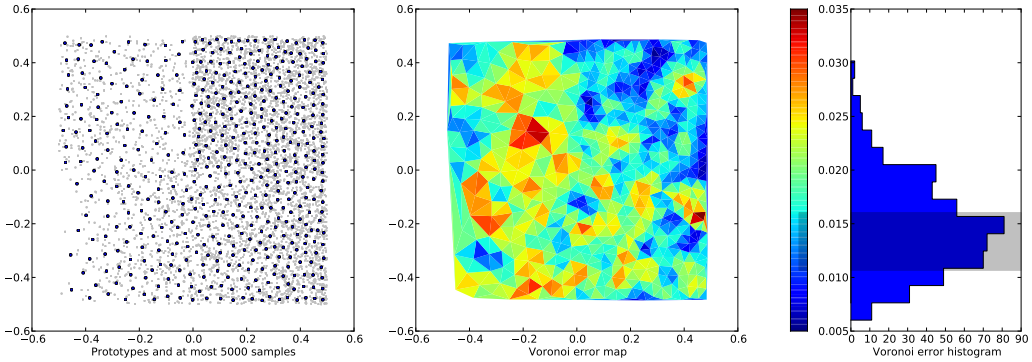

Figure 1: Voronoï distortion distribution over the optimal prototypes. $N = 50000$, $\kappa = 500$, $p(\xi) = \mathbb{1}_{\xi_y \leq 0} \times (0.2 \times \mathbb{1}_{\xi_x \leq 0} + \mathbb{1}_{0 < \xi_x}) + \mathbb{1}_{0 < \xi_y} \times (\xi_x + 0.5)$, $d(\xi, \xi') = \|\xi - \xi'\|^2$. The shortest 50% confidence interval is $[1.066 \times 10^{-2}, 1.603 \times 10^{-2}]$ (see the shaded area on the histogram).

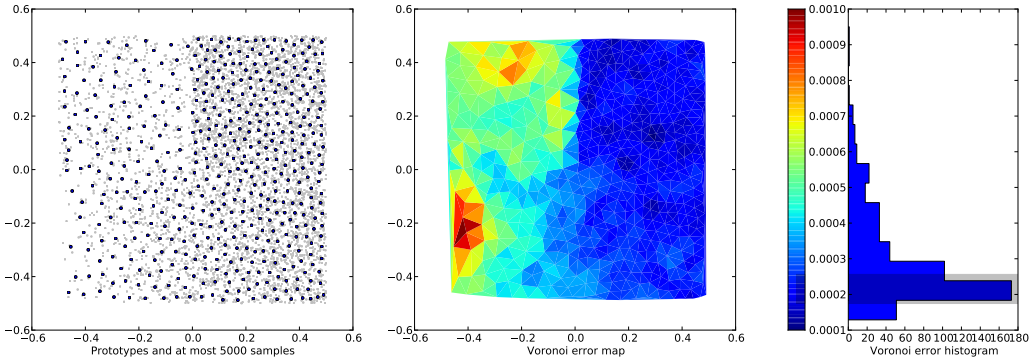

Figure 2: Mean prototype error distribution over the optimal prototypes. $N = 50000$, $\kappa = 500$, $p(\xi) = \mathbb{1}_{\xi_y \leq 0} \times (0.2 \times \mathbb{1}_{\xi_x \leq 0} + \mathbb{1}_{0 < \xi_x}) + \mathbb{1}_{0 < \xi_y} \times (\xi_x + 0.5)$, $d(\xi, \xi') = \|\xi - \xi'\|^2$.

regions of higher or lower values. As far as the author knows, this has not been addressed formally in the vector quantization field. Such distortion variations seem to occur systematically, as suggest figures 3, 4 and 5. In these figures, p delimits 1 and 0-values regions only, and the number of prototypes is determined as $|S|/100$ so that distortions lies in the same range of values in the three figures.

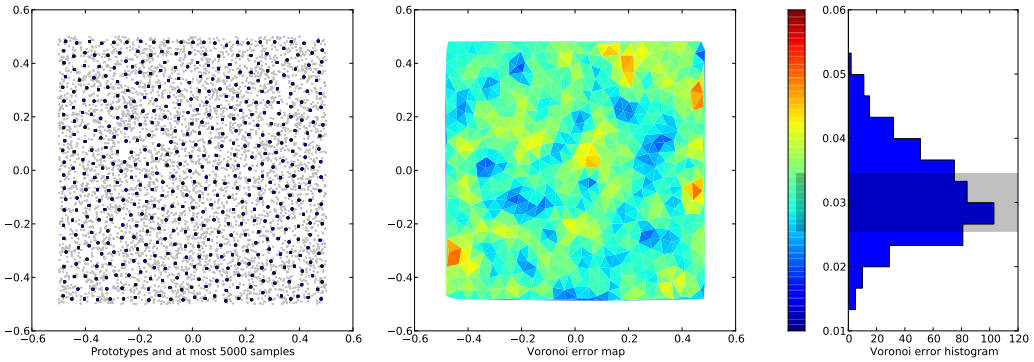

Figure 3: Voronoï distortion distribution over the optimal prototypes. $N = 50000$, $\kappa = 500$, $p(\xi) = 1$, $d(\xi, \xi') = \|\xi - \xi'\|^2$. The shortest 50% confidence interval is $[2.55 \times 10^{-2}, 3.45 \times 10^{-2}]$ (see the shaded area on the histogram).

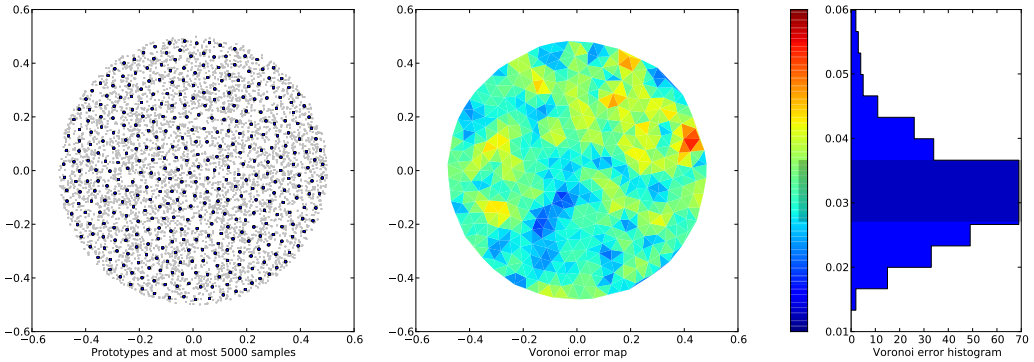

Figure 4: Voronoï distortion distribution over the optimal prototypes. $N = 50000$, $\kappa = 391$, $p(\xi) = \mathbb{1}_{\|\xi\| \leq 0.5}$, $d(\xi, \xi') = \|\xi - \xi'\|^2$. The shortest 50% confidence interval is $[2.715 \times 10^{-2}, 3.655 \times 10^{-2}]$ (see the shaded area on the histogram).

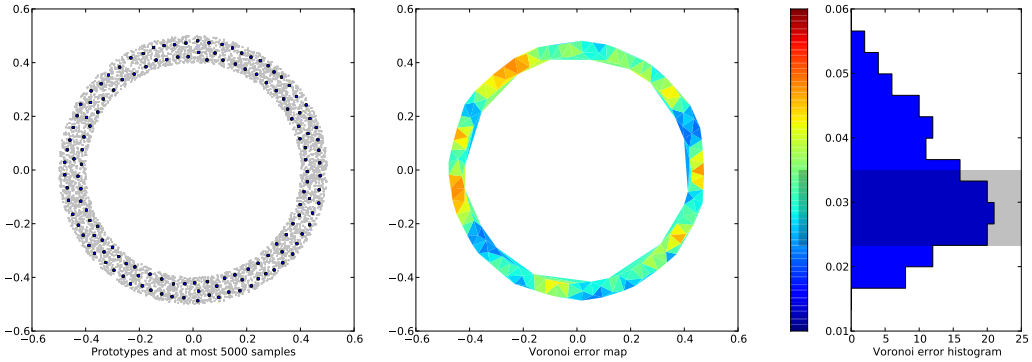

Figure 5: Voronoï distortion distribution over the optimal prototypes. $N = 50000$, $\kappa = 142$, $p(\xi) = \mathbb{1}_{0.4 \leq \|\xi\| \leq 0.5}$, $d(\xi, \xi') = \|\xi - \xi'\|^2$. The shortest 50% confidence interval is $[2.329 \times 10^{-2}, 3.493 \times 10^{-2}]$ (see the shaded area on the histogram).

3. Controlling the quantization accuracy

Usually, in vector quantization algorithms, the number of prototypes is a crucial value that has to be determined by hand. For the κ-means (Linde et al., 1980), choosing the right value of κ when the number of cluster 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, 1995b), Growing Grids (Fritzke, 1995a), etc..., the growth is stopped when some user-defined number of nodes is reached. Several approaches tackle this problem. In Tenceb et al. (2013), a decaying error is maintain for each vertex, and vertices are added until the error for all nodes is kept below some desired threshold. Regulation from information theory principles have also been proposed (Qin and Suganthan, 2004), with some minimum description length argument that is also related to a measure of some error at each vertex. Controlling the number of prototypes κ according to some desired error-based quantization accuracy is the motivation of GNG-T (Frezza-Buet, 2008) as well, and a significant improvement of this control is proposed here, based on a geometrical interpretation of the error value.

3.1. Definition of a scalar accuracy measure

From the input modelling introduced in section 2.1 and the way the input sample set S_p^N is obtained, let us notice that the Voronoï distortion $\mathcal{V}_\omega^{S_p^N}$ in-

troduced in section 2.2, eq. 2, depends on the number N used in algorithm 1. Indeed, for given $k \in \mathbb{N}$, κ, p and Ω_κ, the following obviously stands:

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

and thus

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

Distortion 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 distortion set $\left\{\mathcal{V}_\omega^S\right\}_{\omega\in\Omega}$, where $\Omega = \Omega^\star_{\kappa,S}$ and $S = S_p^N$. We consider the quantization as fitting the target when $N.T$ belongs to this δ-shortest confident interval (δ-SCI, see algorithm 2). The use of $N.T$ makes explicit the influence of N on the distortion values. As the distribution of Voronoï distortions is non symmetrical, the δ-SCI represent 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 the whole 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 some scalar value T requires to choose this value carefully for a given problem. Trial and error adjustment can be done, but the input modelling 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 $\omega \in \Omega^\star = \Omega^\star_{\kappa^\star,S}$ are found, let us consider the induced Voronoï regions $\{\nu_\omega\}_{\omega\in\Omega^\star}$, with $\nu_\omega = \{\xi \in X \mid \operatorname{argmin}_{\omega'\in\Omega^\star} d(\xi,\omega') = \omega\}$ 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 $\int_{\xi\in A} \mathrm{d}\xi$. We can consider roughly that, for any $\omega \in \Omega^\star$, $|\nu_\omega| \approx \frac{|X|}{\kappa^\star}$ since X is tiled by $\kappa^\star$ similar Voronoï regions.

Let us note that $d_\omega = \frac{1}{|\nu_\omega|} \int_{\xi \in \nu_\omega} d(\xi, \omega) \mathrm{d}\xi$ 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 $\mathcal{V}_\omega^S / \left|V_\omega^S\right|$ within the Voronoï cell V_ω^S. The following then stands

$$\frac{\mathcal{V}_\omega^S}{|V_\omega^S|} \approx d_\omega \Rightarrow \mathcal{V}_\omega^S \approx \left|V_\omega^S\right| . d_\omega \Rightarrow \mathcal{V}_\omega^S \approx \frac{N}{\kappa^\star} . d_\omega \tag{6}$$

since S contains N samples that are roughly divided up equally into Voronoï cells, i.e $\left|V_\omega^S\right| \approx N/\kappa^\star$. In section 3.1, the target T as been introduced as the value such as $N.T$ is the most meaningful Voronoï distortion value. Identifying $N.T$ with eq. 6 leads to the interpretation of the target as

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

Let us take the example of figures 3, 4 and 5 to illustrate the interpretation of T. Let us consider that desired accuracy corresponds to the use of $\kappa^\star = 500$ samples when $p = 1$. This is what figure 3 actually shows. As $X = [-0.5, 0.5]^2$, $|X| = 1$ and thus $|\nu_\omega| = 1/\kappa^\star$ for the tiling Voronoï regions. Let us approximate their shape as a disk, whose surface is $1/\kappa^\star$, centered at ω. As $d(\xi, \xi') = \|\xi - \xi'\|^2$, 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 figure 3, $N = 50000$ is used, so $N.T \approx 0.0318$. It actually belongs to the 50%-SCI shaded in the figure 3 histogram. The same stands for figures 4 and 5, since the number of prototypes has been chosen accordingly (see the end of section 2.2), as opposed to figure 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 were to obtain the same quantization accuracy as in figures 3, 4 and 5. Indeed, on the upper right corner of the input space in figures 1, the density of prototypes is locally higher that the density in figure 3, whereas $p(\xi) = 1$ in the upper right corner of both figures.

Let us re-plot figure 1 with a number of prototype determined from the target. The procedure consists in determining κ by dichotomy, until $N.T = N/2\pi\kappa^{\star 2}$ belongs to the 50%-SCI of the Voronoï distorsions. Figure 6 shows the result, where upper right corner is now similar to figure 3.

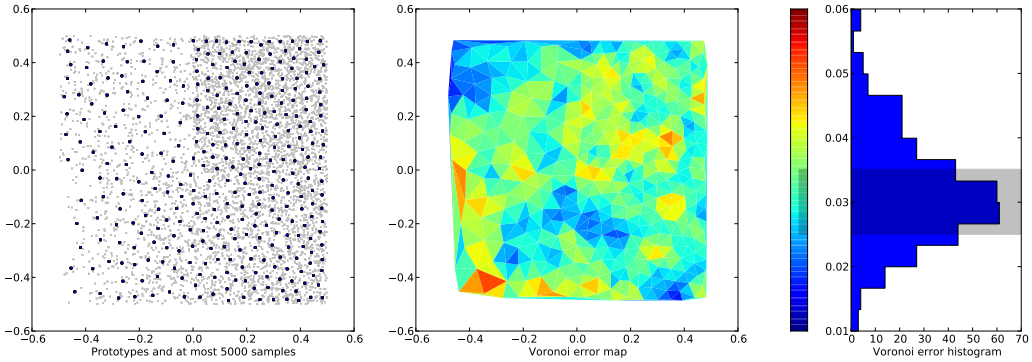

Figure 6: Voronoï distortion distribution over the optimal prototypes. $N = 50000$, $p(\xi) = \mathbb{1}_{\xi_y \leq 0} \times (0.2 \times \mathbb{1}_{\xi_x \leq 0} + \mathbb{1}_{0<\xi_x}) + \mathbb{1}_{0<\xi_y} \times (\xi_x + 0.5)$, $d(\xi, \xi') = \|\xi - \xi'\|^2$. The shortest 50% confidence interval 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 $N.T = N/2\pi\kappa^{\star 2}$ is in the darken range of the histogram. This leads to $\kappa^\star = 343$ in this experiment.

Algorithm 2 `shortest_confidence_interval`(V, δ)

Require: $\delta \in [.5, 1]$ // $V = \{v_i\}_{0 \leq i < N}$ is the set of N values.

```
l ← (int) (δ.N) // (int) (x) stands for the closest integer to x
sort V in an increasing order
r ← v_{N-1} − v_0 + 1
for k = 0 to N − l do
    r' ← v_{k+l-1} − v_k
    if r' < r then
        r ← r', j ← k
    end if
end for
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 distortion 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 on 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 reset of internal values computing statistics. From the input modeling of section 2.1, let us define here an epoch as the setting up of some sample set S_p^N. The connection with real data sampling will be discussed further. The handling of an epoch by GNG-T is described in algorithm 3, where G is the graph of prototypes.

Algorithm 3 gngt_epoch$(G, S, \alpha, \text{dyn_topo})$

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: gngt_open_epoch(G)
5: **repeat**
6: $\quad \xi \sim \mathcal{U}_{S'},\ S' \leftarrow S' \setminus \{\xi\}$
7: $\quad$ gngt_submit$(G, \xi, \alpha, \text{dyn_topo})$
8: **until** $S' = \emptyset$
9: gngt_close_epoch$(G, \text{dyn_topo})$

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, 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 $\mathrm{e}^{v \leftrightarrow v'}$ and edge between v and v', by $\mathrm{E}(v)$ the edges involving the vertex v and $\mathrm{V}(v) = \left\{v' \in G \,|\, \exists \mathrm{e}^{v \leftrightarrow v'} \in \mathrm{E}(v)\right\}$ 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 algorithm 4).

Algorithm 4 gngt_open_epoch(G)

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

After this clear, samples are submitted one by one, the submission procedure being very close to what GNG does (see algorithm 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 some coefficient $\alpha \in [0, 1]$, such as 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 algorithm 6) is used.

Algorithm 5 `gngt_submit`$(G, \xi, \alpha, \text{dyn\_topo})$

if $\kappa^G < 2$ **then**
  if dyn_topo **then**
    Add a new vertex v to G.
    $v_\omega \leftarrow \xi$ // The prototype takes the value of the sample.
    $v_\text{e} \leftarrow 0$
    $v_\text{n} \leftarrow 1$
  end if
  return // Exit the algorithm here.
end if
$v^\star \leftarrow \text{argmin}_{v \in \text{V}(G)} d(v_\omega, \xi)$ // $v^\star$ is the closest vertex.
if dyn_topo **then**
  $v^{\star\star} \leftarrow \text{argmin}_{v \in \text{V}(G) \setminus \{v^\star\}} d(V_\omega, \xi)$ // $v^{\star\star}$ is the second closest vertex.
  create the edge $\text{e}^{v^\star \leftrightarrow v^{\star\star}}$ if it does not exist yet.
  $\text{e}_\text{a}^{v^\star \leftrightarrow v^{\star\star}} \leftarrow 0$ // Set/reset the age of the edge at a 'newborn' value.
  for all $\text{e} \in \text{E}(v^\star) \setminus \left\{\text{e}^{v^\star \leftrightarrow v^{\star\star}}\right\}$ **do**
    $e_\text{a} \leftarrow e_\text{a} + 1$
    if $e_\text{a} > \varphi$ **then**
      remove e // Too old edges are removed.
    end if
  end for
end if
$v^\star_\text{e} \leftarrow v^\star_\text{e} + d(v^\star_\omega, \xi)$ // Update winner statistics.
$v^\star_\text{n} \leftarrow v^\star_\text{n} + 1$
`vq_learn`$(\alpha, v^\star_\omega, \xi)$
for all $v \in \text{V}(v^\star)$ **do**
  `vq_learn`$(\zeta\alpha, v_\omega, \xi)$ // $\zeta \in]0, 1]$ so that the learning of neighbors is weaker.
end for

Algorithm 6 vq_learn(α, ω, ξ)

Require: $\alpha \in [0, 1]$

1: $\omega \leftarrow \omega + \alpha.(\xi - \omega)$

Once all the samples of S_p^N 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 start and the current number of vertices. Here again, considering epochs makes the occurring of the topology evolution more rational. The topology is updated according to algorithm 7.

Algorithm 7 `gngt_close_epoch`$(G, \text{dyn\_topo})$

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: $v_\omega \leftarrow v^\star_\omega$ // An isolated 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 neighbour of $v^\star$ with the highest accumulated error.
26: `vq_learn`$(\lambda, v_\omega, v^{\star\star}_\omega)$ // 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**

There are slight modifications from GNG in the network evolution procedure described by algorithm 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 discussed later), while GNG essentially adds neurons. A second change is line 26. In GNG (and our previous version of GNG-T), $v_\omega \leftarrow \frac{v^\star_\omega + v^{\star\star}_\omega}{2}$ was used, i.e. $\lambda = 0.5$. Nevertheless, with this rule, the newly created neuron can be isolated between two cluster of samples, and then immediately deleted at next epoch at line 6... and then created again, etc. 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 algorithm 7. In our previous implementation (Frezza-Buet, 2008), this control was straightforward. It is recalled in algorithm 8.

Algorithm 8 `compute_evolution`(G)

1: $\mu = \frac{1}{\kappa^G} \sum_{v \in G} v_\mathrm{e}$ // μ is the average of accumulated errors.
2: **if** $N.T < \mu$ **then**
3: **return** `Add`
4: **else**
5: **return** `Remove`
6: **end if**

Let us 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 $t, t+1, \cdots$ are run with a constant density function $\forall \xi \in X,\ p^t(\xi) = d^t$, 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 the set to another value for the next 100 epochs, etc. 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 algorithm 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).

Figure 7 shows that GNG-T actually adjusts the number of nodes according to $N.T$, but this number of node 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 re-adjust their position according to their current number so that X is fairly shared[1].

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

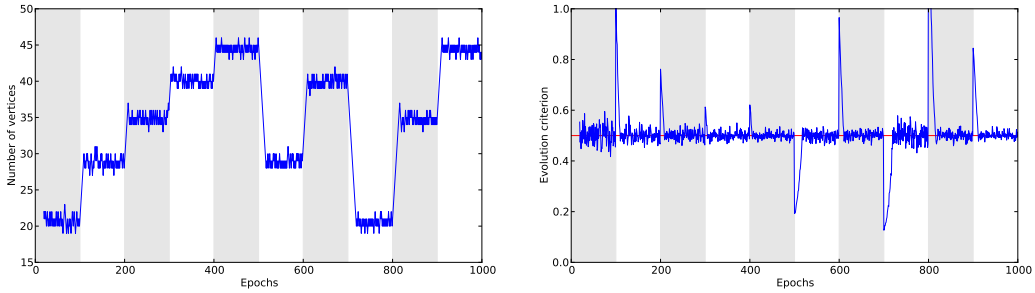

Figure 7: Evolution of the number of nodes computed by GNG-T for successive epochs. The distribution $p^t(\xi) = \text{const}$ at each epoch, the constant value differs from one chunk of 100 epochs to the following. Chunks are marked as stripes in the plots, and const is taken in $\{0.2, 0.4, 0.6, 0.8,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

This spurious motion can be significantly reduced by considering δ-SCI introduced in section 3.1. Indeed, algorithm 9 can be used instead of algorithm 8. This leads to the behavior depicted in figure 8. It can be seen that the number of prototypes is much more stable.

Algorithm 9 `compute_evolution`(G)

$(\min, \max) \leftarrow$ `shortest_confidence_interval`$\left(\{v_e\}_{v \in G}, \delta\right)$
$\Delta \leftarrow \max - \min$
$a \leftarrow \min + \mu\Delta$ // $\mu \in [0, .5[$, $[\min, \max]$ 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 algorithm 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 algorithm 10). Figure 9 shows

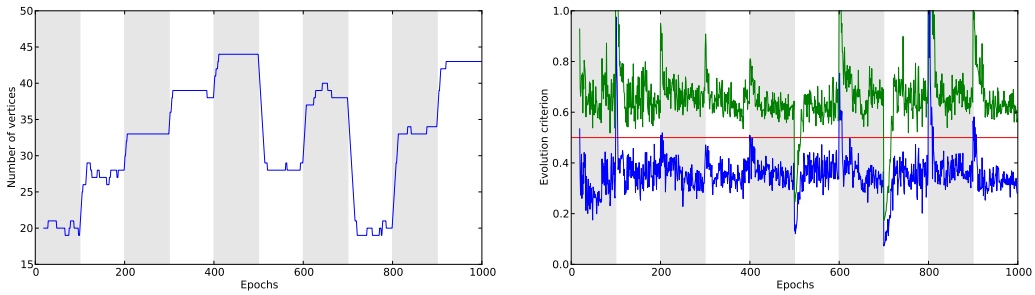

Figure 8: Experiments and graphical convention are similar to figure 7. On the right plot, min and max values computed by algorithm 9 are plotted. The algorithm 9 parameters are $\delta = 75\%$, $\mu = 0.2$.

the behavior of GNG-T in this case[2].

Algorithm 10 `compute_evolution`(G)

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

$(k, l) \leftarrow$ `shortest_confidence_interval`$\Big(\{v_e\}_{v \in G}, \delta\Big)$
$\min \leftarrow \min + \nu(k - \min)$ // $\nu \in [0, 1]$.
$\max \leftarrow \max + \nu(l - \max)$
$\Delta \leftarrow \max - \min$
$a \leftarrow \min + \mu\Delta$ // $\mu \in [0, .5[$, $[\min, \max]$ 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

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

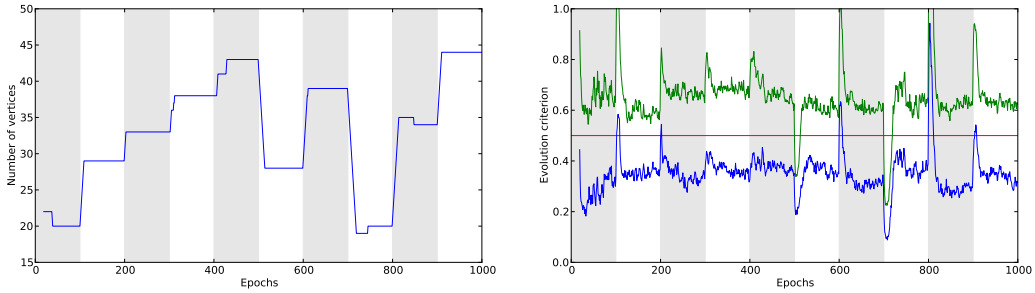

Figure 9: Experiments and graphical convention are similar to figure 7. On the right plot, min and max values computed by algorithm 10 are plotted. The algorithm 10 parameters are $\delta = 75\%$, $\mu = 0.2$, $\nu = 0.4$.

5. Velocity field

The motivation for the stabilization of GNG-T presented in this paper is the computation of the velocity field of some non-stationary distribution. Indeed, the stabilization proposed in section 4 removes spurious motions of the prototype, 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 that samples 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 the 2D image[3]. Even when only the optical flow it 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 and Silvermann, 1988), detailed further. To cope with that ambiguity, most optical flow techniques add extra hypotheses on the velocity field (smooth variation over space, etc...) so that the processing consists of finding a flow that fits the best both the data and the constraints

[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.

(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 that 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 paragraph, and then the velocity field computation is introduced and illustrated on artificial data.

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.

Let us consider 2D input set as in 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 figure 10. 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 algorithm 3. GNG-T the 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 figure 10 shows[5]. This ef-

[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`

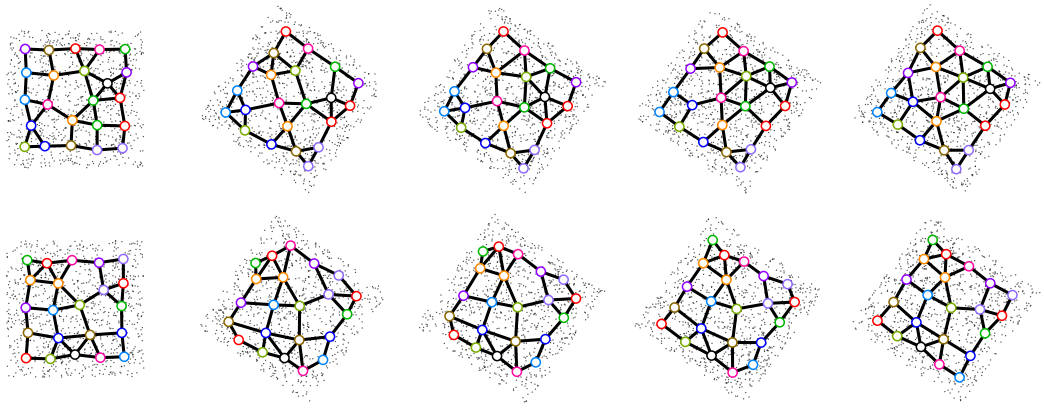

Figure 10: Evolution of GNG-T prototypes when p^t rotates abruptly. The target T is kept constant during the transition. The top line shows the graph adaptation when topology adaptation is enabled. It is disabled in the bottom line.

fect is an emerging as well as 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), the preservation of the structure (section 5.1), the velocity field can be computed from some data stream. The quality of the speed measures relies on the efforts for stabilizing GNG-T (section 4.2).

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 epoch are required for some time instance t in order to apply a structure-based temporal smoothing. It means that algorithm 3 has to be executed several time for each instance of p^t. Indeed, 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 about prototypes positions. This leads to the update for each time step detailed in algorithm 11.

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

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

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 algorithm 12 (see line 11 in algorithm 11). This prototype position averaging is more reliable to estimate the speed that the current position. This is why, for each time, and for each 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 algorithm 13 does at the start of each time computation (see line 2 in algorithm 11).

Algorithm 12 `prototype_mean`(G)

// For a newly created vertex v, $v_\mathrm{s} = \texttt{New}$, $v_\Sigma = 0$ and $v_\# = 0$.
for all $v \in \mathrm{V}(G)$ **do**
  if $v_\mathrm{s} \neq \texttt{New}$ **then**
    $v_\Sigma \leftarrow v_\Sigma + v_\omega$
    $v_\# \leftarrow v_\# + 1$
  end if
end for

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

for all $v \in \mathrm{V}(G)$ **do**
  if $v_\mathrm{s} \neq \texttt{Ok}$ **then**
    $N \leftarrow \{v' \in \mathrm{V}(v) \text{ and } v'_\mathrm{s} = \texttt{Ok}\}$ // N gathers the neighbors of v for which some speed is computed.
    $\overline{\mathrm{d}\omega} \leftarrow 1/|N| \sum_{v' \in N} v'_{\mathrm{d}\omega}$
  end if
  if $v_\mathrm{s} = \texttt{New}$ **then**
    $v_\mathrm{s} \leftarrow \texttt{NoPrev}$
    $v_{\mathrm{d}\omega} \leftarrow \overline{\mathrm{d}\omega}$ // The speed is obtained from neighboring vertices.
    $v_{-1} \leftarrow 0$
  else if $v_\mathrm{s} = \texttt{NoPrev}$ **then**
    $v_\mathrm{s} \leftarrow \texttt{Ok}$
    $v_{\mathrm{d}\omega} \leftarrow \overline{\mathrm{d}\omega}$ // The speed is obtained from neighboring vertices.
    $v_{-1} \leftarrow v_\Sigma / v_\#$ // $v_\Sigma / v_\#$ is the average position of v_ω during last non-transient epochs.
  else if $v_\mathrm{s} = \texttt{Ok}$ **then**
    $v_{\mathrm{d}\omega} = \left(v_\Sigma / v_\# - v_{-1}\right) / \Delta t$
    $v_{-1} \leftarrow v_\Sigma / v_\#$
  end if
  $v_\Sigma \leftarrow 0$, $v_\# \leftarrow 0$
end for

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 cycle is repeated three times[6]. Snapshots of this dynamical process are shown in figures 11, 12 ,13 , 14 and 15. In these figures, it can be noticed that the velocity field is retrieved for all nodes, even for the ones lying at the center of the distribution, where no local change is visible. For example, figure 13 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.

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 [7]. 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 (García-Rodríguez et al., 2012; Fišer et al., 2013). Only plots of time consumption are given here, without details about implementation issues.

Let us implement a color-based pixel selection (see yellow pixels on the left frames in figure 16). 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

[6] See the movie `http://malis.metz.supelec.fr/depot/Vq/Paper/expe-005-003.avi`

[7] `http://malis.metz.supelec.fr/spip.php?article182`

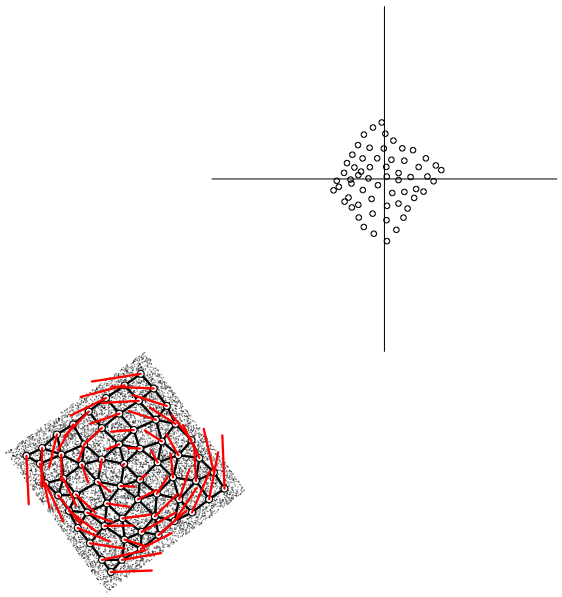

Figure 11: Input space is $X = [-1, 4] \times [-0.9, 0.9]$, where the square area for which $p^t(\xi) = 1$ has a 1-length side. The bottom of the figure shows the nodes as well as the velocity field (red lines). The opposite of the node speed is rather represented by the red lines for the sake of clarity. 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 (clock-wise). The velocity field is computed over the whole distribution, in accordance with the rotation. Other parameters are $N = 200000$, $T = 5 \times 10^{-6}$, $n_{\text{struct}} = 10$, $n_{\text{evol}} = 5$, $n_{\text{mean}} = 100$, $\delta = 75$, $\nu = 0.4$, $\mu = 0.2$. 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.

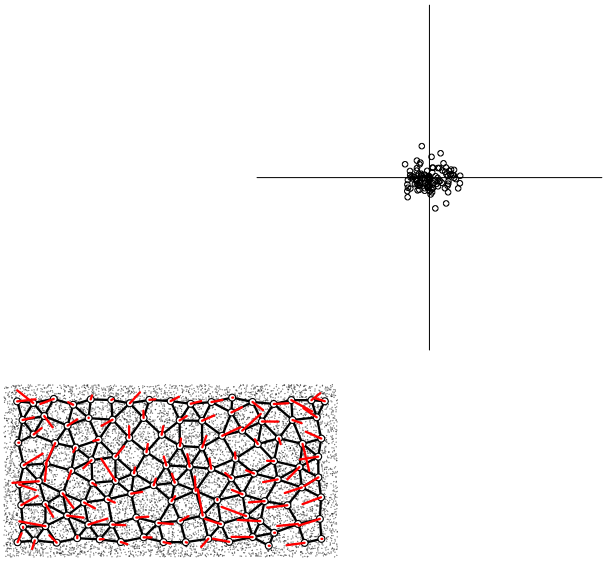

Figure 12: Same experiment as figure 11, but the snapshot is taken at the end of the expanding stage. The speeds are distributed along the horizontal axis, in accordance with the horizontal expansion velocity field.

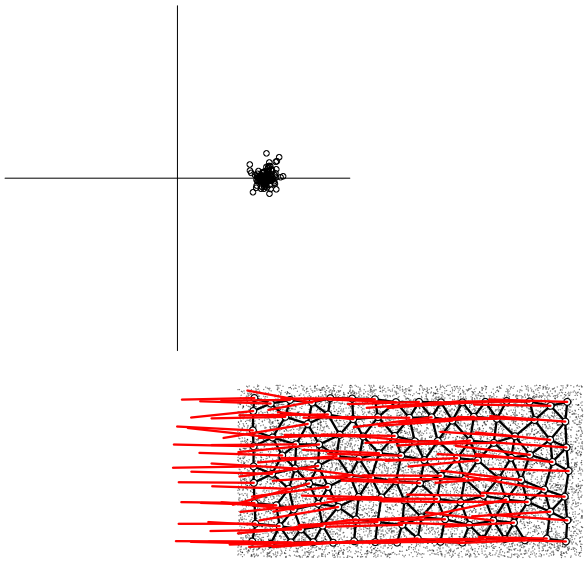

Figure 13: Same experiment as figure 11, but the snapshot is taken during the translation to the right. The velocities are the same for all nodes, in accordance with the global translation.

Figure 14: Same experiment as figure 11, but the snapshot is taken during the translation to the left.

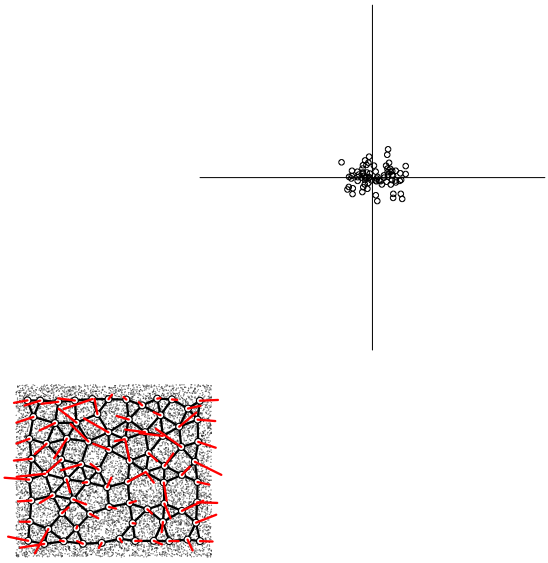

Figure 15: Same experiment as figure 11, but the snapshot is taken at the end of the shrinking stage. The speeds are linearly distributed along the horizontal axis, in accordance with the horizontal shrinking velocity field.

$N = w \times h$ 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 some efficiency reasons, it is as if $N = \alpha.(w \times h)$ were used in algorithm 1. Thus, one need 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[8] 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 algorithm 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 actually used in algorithm 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 influence on both the computation time and the qualitaty of the result is measured from a recorded movie and not 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` $= 250$[9] and `maxsamples` $= 2000$[10] are qualitatively the same, as figures 16 and 17 show. On the computer used for the test, a Intel Core i5 CPU 650 4×3.20GHz with a 3.8Go RAM running under a Fedora 18 Linux distribution, the performances of our implementation are plotted on figure 18. 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 algorithm 11, $\delta = 0.75, \nu = 0.4, \mu = 0.2, T = 0.1$ for algorithm 10, N is determined from

[8] `www.gstreamer.net`

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

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

equation 8 for each frame, $\varphi = 50, \zeta = 0.2, \alpha_{\text{fast}} = 0.005, \alpha_{\text{slow}} = 0.001, \lambda = 0.001$ for algorithms 5 and 7.

7. Conclusion

This paper promotes the topology preserving vector quantization as a promising tool for tackling the problem of anchoring symbolic structures in analogous input data flows. It extends existing approaches in two directions. First, a quantitative measure of the accuracy of the quantification itself is exhibited, thus enabling the control of the appropriate number of prototypes as well as a stabilization of their positions. Second, it adds to the prototypes a velocity information, inferred from successive variations of the sample densities. Labelling the vertices of the graph with velocity information goes beyond the velocity field computation itself, since the velocities in the field can be labelled according to the connected components of the graph, bringing supplementary semantics to the flow analysis process, as rightmost frames in figure 16 show.

For the sake of clarity, the experiments in this paper address bi-dimensional distributions, in order to offer a qualitative comparison with optical flows, which are the most common velocity field processing techniques. As opposed to optical flows, the algorithm formulation does not use any Taylor approximation of some temporal differential operator, since an emerging effect of Kohonen Self-Organizing Maps is exploited to apply continuity constraints from one frame to the next. Therefore, low frame rate temporal sampling, for which significant changes between consecutive frames are observed, can be handled, as figure 10 shows.

What has been shown on bi-dimensional distributions in this paper can be generalized to any kind of distributions, since no steps of the algorithms presented is specific to bi-dimensional data. For these general cases as well, a clever use of the N parameter of algorithm 1 could help to control the time consumption, as it has been done with the `maxsamples` parameter in our video experiment.

Having at our disposal a topology preserving vector quantization tool where the dynamics is controlled, from which velocity can be extracted reliably, opens perspective in the processing of complex signals, as for examples those received from sensors of robotic devices, where a semantic interpretation is crucial for taking the right behavioral decisions. This is what ongoing work is addressing.

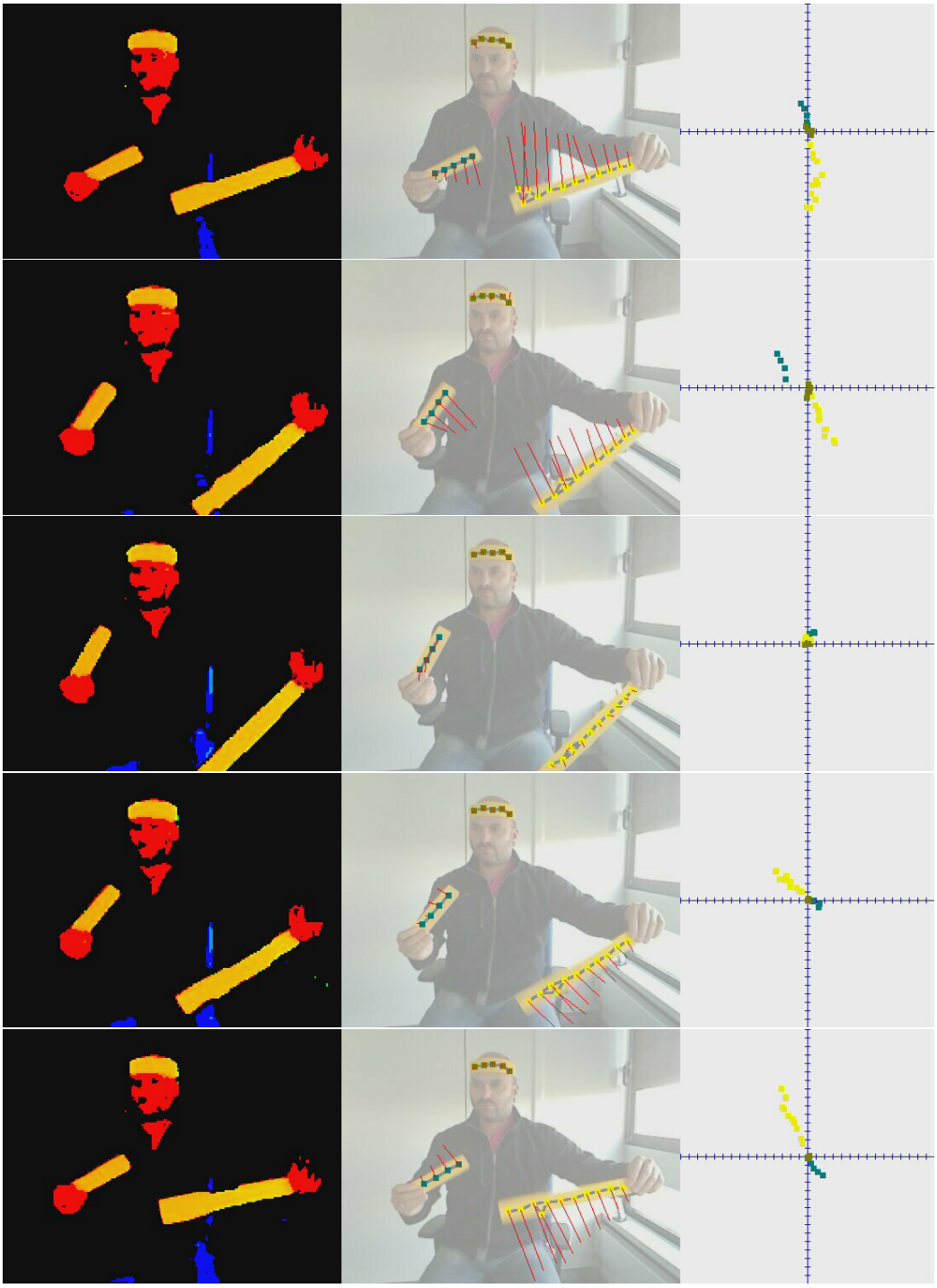

Figure 16: Consecutive snapshots of our gstreamer plugin during a 'drum roll' gesture, taken every five frames. On the left, the color-base selection filter is shown. Only yellow pixels are submitted to the algorithm, with a maximum of `maxsamples` = 250. The center image shows the graph computed by 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, are 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.

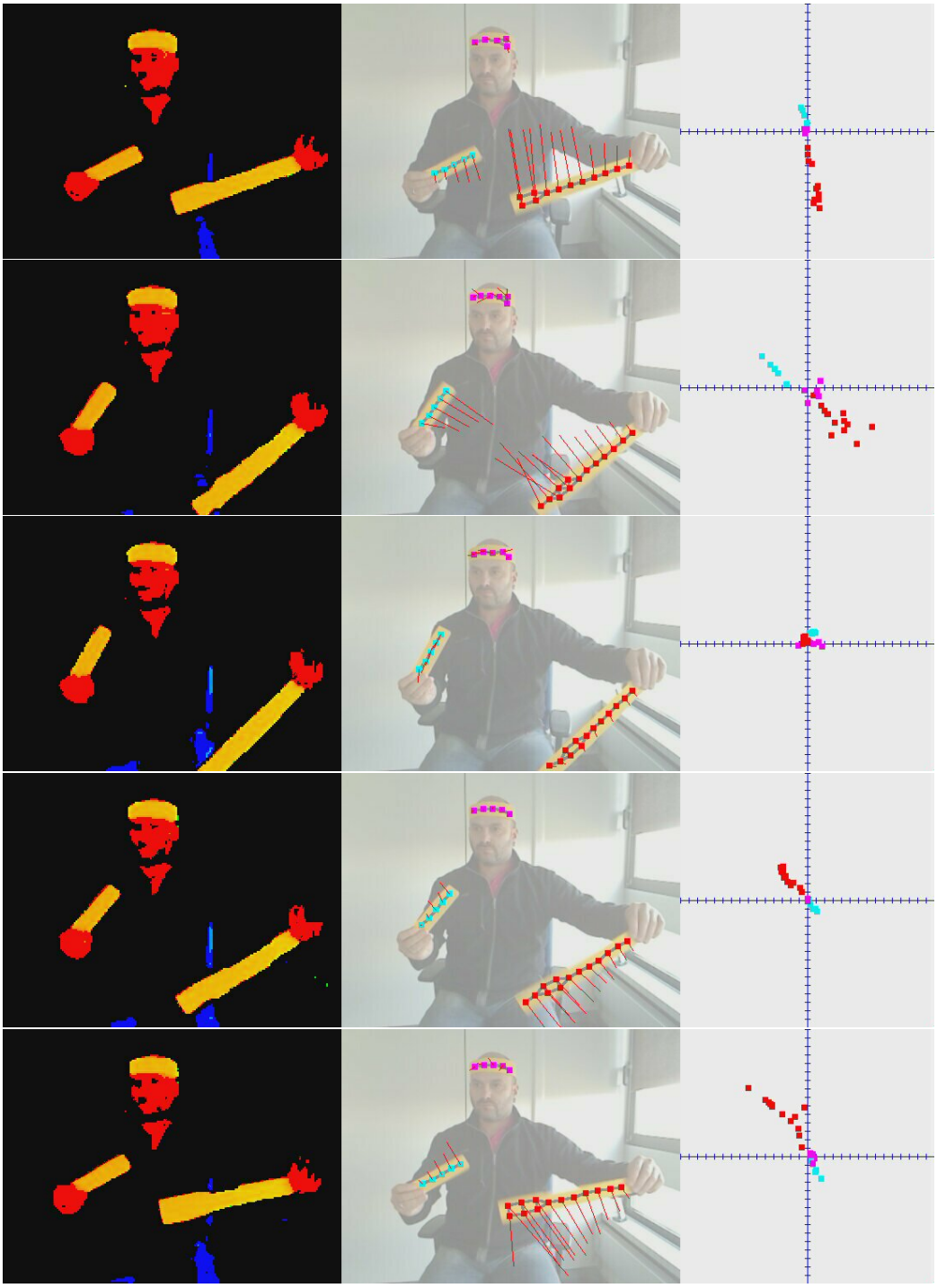

Figure 17: Same as figure 16, but with `maxsamples` = 2000.

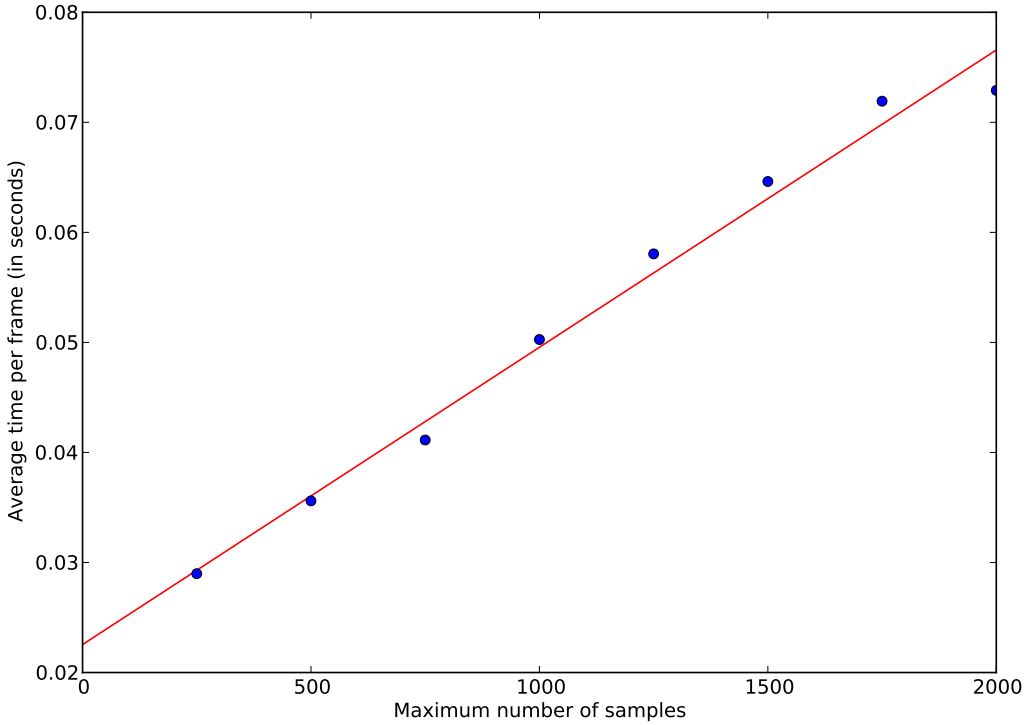

Figure 18: Average frame computation time of our gstreamer plugin, when included in a pipeline and fed with the benchmark movie used in figures 16 and 17, according to the `maxsamples` parameter.

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.

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.

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. 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, pp. 625–632.

Fritzke, B., 1997. A self-organizing network that can follow non-stationary distributions. In: ICANN'97: International Conference on Artificial Neural Networks. Springer, pp. 613–618.

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.

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.

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.

Tenceb, F., Gauberta, L., Solera, J., Loora, P. D., Buchea, C., 2013. Stable growing neural gas: A topology learning algorithm based on player tracking in video games. Applied Soft Computing 13, 4174–4184.