Incremental learning for visual classification using Neural Gas

Ignazio Aleo, Paolo Arena and Luca Patané.

Abstract— **In this paper we investigate a novel algorithm for solving classification problems in an action-oriented perception framework supported by visual feedback. The approach is based on an extension of the Neural Gas with local Principal Component Analysis (NGPCA) algorithm. As an abstract Recurrent Neural Network (RNN) this model is able to complete a partially given pattern. Under this point of view it is possible to generalize the model as a supervised classifier in which for a given segmented object (i.e. with particular visual cues) the class variable is retrieved as the network outputs. An incremental version of the algorithm is also presented and applied in a robotic platform for object manipulation tasks.**

I. Introduction

In robotic applications data classification is extremely important and, as far as the visual input is considered, the number of features and the complexity in class discrimination increase enormously. For these reasons it is important to identify algorithms for classification that show characteristics like robustness, low computational needs and flexibility, in order to be adopted in different situations and others.

The Neural Gas (NG) [1] is a vector quantization technique that tries to approximate a given manifold with a reduced number of points.

With the local Principal Components it is possible to represent sensor distributions locally constrained to subspaces with fewer dimensions than the space of the training data. For this reason, as described in [2], Principal Component Analysis (PCA) is able to model distributions in which directions with almost zero variance exist. This reveals to be a key point in a visual classification task in which the number of dimensions in the feature space is easily very high.

With the introduction of PCA in the Neural Gas algorithm (NGPCA), each of the data-representative points (hereafter indifferently referred to as code-book vectors or particles) is extended into hyper-ellipsoids to better match the given data distribution [3]. This technique has been applied in different fields as redundant inverse and forward kinematic (IK and FK) for serial manipulator and roving robot path planning [4]. Though in this first implementation the algorithm exploits a supervised learning mechanism, our final aim is to further develop a fully unsupervised classifier for real robotic applications. Under this point of view our algorithm has been compared, from the beginning, with a Self-Organizing Map (SOM) [5] which is at the same time flexible and simple.

As a key point for a real time application, the data set is incrementally acquired during the experiment and not completely available at the beginning as it happens in traditional structures. Under this point of view, it is important to envisage a growing mechanism that allows the algorithm to cope with sample collection typical of a moving robot in an unknown environment (i.e. dynamically changing training set).

The model architecture can be easily divided in a learning phase, in which the data distribution is approximated and in a recall phase in which an uncomplete pattern is presented to the network in order to retrieve the output (complete pattern). As already discussed in [6] [7], similarly to other classes of Recurrent Neural Networks (RNNs), this model is able to cope with multiple solution tasks providing one of the possible solutions and it is also possible to choose the role of input and output neurons, after the training, simply modifying the recall phase.

In the next sections an extension to a supervised classification of the NGPCA algorithm, with visual cues variables used as part of the input pattern, is discussed. Moreover an incremental on-line version of the architecture gives an added value action-oriented classification capability for object manipulation on a roving robot applications in dynamically changing environment [8].

II. Model description

The Neural Gas algorithm is a variant of the soft-clustering vector quantization with the addition of annealing. For a given pattern space $P \subseteq \Re^d$, the algorithm starts choosing m points c_j (with $j = 1, 2, ..., m$), in this hyperspace, from the N training set elements.

During each step, a random pattern $\mathbf{x}$ is chosen from the training set. Then each j-point position $\mathbf{c_j}$ is updated for the next iteration step relating to its rank r_j function of the distance from the selected pattern through the learning rate ε and the neighborhood range ϱ as follows:

$$\mathbf{c}_j(t+1) = \mathbf{c}_j(t) + \alpha_j \cdot [\mathbf{x} - \mathbf{c}_j(t)], \quad (1)$$

where α_j is defined by $\alpha_j = \varepsilon \cdot e^{-r_j/\varrho}$.

In order to force algorithm convergence through iterations, both ε and ϱ exponentially decrease from ε_{init} (ϱ_{init}) to ε_f (ϱ_f) as in the following:

$$\begin{aligned} \varepsilon(t) &= \varepsilon_{init}(\frac{\varepsilon_f}{\varepsilon_{init}})^{\frac{t}{T}} \quad with \;\; \varepsilon_{init} \geq \varepsilon_f, \quad (2) \\ \varrho(t) &= \varrho_{init}(\frac{\varrho_f}{\varrho_{init}})^{\frac{t}{T}} \quad with \;\; \varrho_{init} \geq \varrho_f, \end{aligned}$$

where t is the current iteration time and T is the last iteration time.

The local PCA extension of the NG considers hyper-ellipsoid units, with q principal components, instead of simple points and therefore the ranking of the units cannot

Ignazio Aleo, Paolo Arena, and Luca Patané are with the Dipartimento di Ingegneria Elettrica, Elettronica e dei Sistemi (DIEES), Universitá degli Studi di Catania, Italy (email: {ialeo,parena,lpatane}@diees.unict.it)

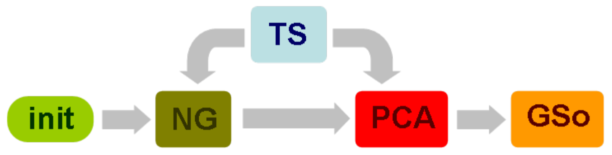

Fig. 1. Algorithm block diagram for the learning phase of the Neural Gas with local Principal Component Analysis (NGPCA). The NG block performs center updating for a given training vector, extracted by the Training Selector (TS), the PCA block executes one step of local principal components algorithm and GSo (Graham-Schmidt ortho-gonalization) is able to give eigenvectors orthogonality property.

TABLE I
SUMMARY TABLE

m	number of particles
N	number of training patterns
d	pattern space dimension
q	number of principal components
$\mathbf{c}_j$	center of particle j
r_j	ranking of particle j
v_{res}	total residual variance
Λ_j	eigenvalues matrix of particle j
W_j	eigenvectors matrix of particle j

depend on an Euclidean distance. One of the possible distance measure is the normalized Mahalanobis distance [4] [9], an elliptical distance that can be computed, for each j particle, as:

$$E(x) = \xi^T W\Lambda^{-1}W^T\xi + \tfrac{1}{\sigma^2}(\xi^T\xi - \xi^T WW^T\xi) + \\ + ln(det\Lambda) + (d-q)ln\sigma^2, \qquad (3)$$

where $\xi = \mathbf{x} - \mathbf{c}$ is the deviation of vector x from the center unit, W is the eigenvector matrix, Λ is a diagonal matrix containing the eigenvalues. The second term of equation (3) is the reconstruction error divided by σ^2 that depends on the total residual variance v_{res}, among all $d-q$ minor dimensions ($d \geq q$), as in

$$\sigma^2 = \frac{v_{res}}{d-q}. \qquad (4)$$

The total residual variance is updated according to:

$$v_{res}(t+1) = v_{res}(t) + \\ + \alpha \cdot (\xi^T W\Lambda^{-1}W^T\xi - \xi^T WW^T\xi - v_{res}(t)). \qquad (5)$$

To modify principal components of existing ellipsoids, one step of the Robust Recursive Least Square Algorithm (RRLSA), described in [10] [11] is performed.

$$\{W(t+1), \Lambda(t+1)\} = \\ \mathbf{PCA}\{W(t), \Lambda(t), \xi(t), \alpha(t)\} \qquad (6)$$

Since the orthogonality of W is not preserved after each step, the Gram-Schmidt orthogonalization method has been introduced [4]. The algorithm overall block diagram is shown in Fig. 1 where the already introduced main blocks are: an initialization block (init) that initializes all learning parameters and variables; a Neural Gas block (NG) that updates the position of the centers of the ellipsoids for a given pattern x chosen by the Training Selector (**TS**) from the training set $\mathbf{T} = T_1, T_2, ..., T_N$ as in equation (1); a Principal Component Analysis block (PCA) that updates hyper-ellipsoids axes with RRLSA as in equation (6) and a orthogonalization block (GSo) that performs the so called Gram-Schmidt algorithm. Relevant parameters and variables used in the model are summarized in Table I. In order to reduce the dependence of the error in equation (3) from the volume of the considered ellipsoid and to avoid useless particles (with weight α almost zero) in the pattern space, it is possible, as in [4], to modify the distance measure as follows:

$$\widetilde{E}(x) = (\xi^T W\Lambda^{-1}W^T\xi + \\ + \tfrac{1}{\sigma^2}(\xi^T\xi - \xi^T WW^T\xi))V^{2/d}, \qquad (7)$$

where V is the volume of the considered ellipsoid unit and can be computed according to

$$V = \sigma^{d-q}\sqrt{det\Lambda}. \qquad (8)$$

A. Testing phase

After the learning phase, the data distribution is represented by m hyper-ellipsoids with center $\mathbf{c}_j$ (with $j = 1, 2, ..., m$), semi-axes lengths $\sqrt{\lambda_j^k}$ (with $k = 1, 2, ..., q$), $\mathbf{w_j}^k$ principal component eigenvectors and a residual variance σ_j^2.

In the recall phase an incomplete pattern $\mathbf{p}^* \subseteq \Re^{d-s}$, with an s number of laking dimensions ($s \leq d$), is given as input and the algorithm would rebuild the s free dimensions and give the optimum complete pattern $\widehat{z}_{j^*}$ as output.

In order to perform the recall, the input to the network is given in form of an offset $\mathbf{p}$ (i.e. fulfilment of $\mathbf{p}^*$ in R^d with zeros among lacking dimensions) in the constrained space $\mathbf{z}(\eta) \subseteq \Re^d$ as follows:

$$\mathbf{z}(\eta) = \mathbf{M}\eta + \mathbf{p}, \qquad (9)$$

where $\mathbf{M}$ matrix aligns the constrained space to a particular parameter space while $\eta \in \Re^s$ is a vector of free parameters.

For instance, if $d = 6$ and $s = 3$ and the three free variables are in the last part of the training patterns (i.e. $\mathbf{p}$ of the form $\mathbf{p} = [p_1, p_2, p_3, 0, 0, 0]$) the $\mathbf{M}$ matrix has the following form:

$$\mathbf{M} = \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix}, \quad \mathbf{0} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \qquad (10)$$

A potential function in the constrained subspace $E_j(\mathbf{z})$ can be computed for each ellipsoid as:

$$E_j(\mathbf{z}) = \mathbf{y}_j^T\Lambda_j^{-1}\mathbf{y}_j + \tfrac{1}{\sigma_j^2}(\xi_j^T\xi_j - \mathbf{y}_j^T\mathbf{y}_j) + \\ + ln(det\Lambda_j) + (d-q)ln\sigma_j^2, \qquad (11)$$

where ξ_j is the displacement from the center $\xi_j = \mathbf{z} - \mathbf{c}_j$ and $\mathbf{y}_j = W_j^T\xi_j$ is the representation in the local coordinate system of the ellipsoid.

For each unit j, the point of constrained space with smallest potential $\widehat{z}_j$ is determined, according to equation (11), and then the unit j^* that has the minimal potential among all $E_j(\widehat{\mathbf{z}}_j)$ is chosen as complete pattern.

As shown in [4], the function $E(\eta)$ is convex and it is possible to determine analytically the only minimum $\widehat{\eta}_j$ computing:

$$\widehat{\eta}_j = \mathbf{A}_j(\mathbf{p} - \mathbf{c}_j), \tag{12}$$

with

$$\begin{aligned} \mathbf{A}_j &= -(\mathbf{M}^T D_j \mathbf{M})^{-1}\mathbf{M}^T\mathbf{D}_j, \\ \mathbf{D}_j &= \mathbf{W}_j \mathbf{\Lambda}_j^{-1}\mathbf{W}_j^T + \tfrac{1}{\sigma_j^2}(\mathbf{I} - \mathbf{W}_j\mathbf{W}_j^T). \end{aligned} \tag{13}$$

III. Classification problem

A simple supervised classification problem is one of the straightforward applications of this kind of approach.

The proposed strategy is oriented to object classification through visual features acquired by a moving robot (e.g. robotic arms or rovers with manipulation capabilities) equipped with a camera. The training pattern is divided in a feature hyper-space $\mathbf{f}$ and a class identification parameter n.

$$\mathbf{x} = [\mathbf{f}, n]. \tag{14}$$

During the training phase the complete pattern is presented as input to the algorithm. In the testing phase the class number portion of the vector is left free and is retrieved as output.

A. *Patterns*

Simple visual cues have been chosen to build the feature part f of the input training pattern. Both a space-color representation and geometrical features extraction have been performed in a segmented image. The most present hue, saturation and value (respectively c_H, c_S and c_V) and some geometrical features p, A, e_m and e_M (respectively the perimeter, the area, the shorter and the longer edge of the minimum rectangle that contains the object) have been extracted from a preprocessed image obtained with the Canny segmentation algorithm [12] [13].

The feature part of the input pattern is here described:

$$\mathbf{f} = [c_H, c_S, c_V, p, A, e_m, e_M]. \tag{15}$$

As often happens in real-time robotics, in our application, the training data-set is not fully available at the beginning of the learning: we suppose the robot to collect training images (i.e. complete training patterns) during its mission (i.e. iteration after iteration) and at the same time try to correctly classify them to increase its performance in a foraging task-oriented scenario.

B. *Incremental Learning*

In order to achieve incremental learning, several problems need to be overcome. To this end some key modifications to the presented classification algorithm were added.

First of all, the number of particles vectors should be variable. Then the annealing of each ellipsoid (i.e. the capability to move) must be function of its own age (i.e. number of steps present in the training).

The on-line supervised nature of the algorithm needs both recall and training phase to be achieved in each iteration as explained below.

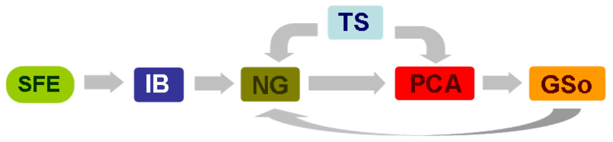

Fig. 2. Algorithm block diagram for on-line incremental learning that uses pre-processed input vectors. The Segmentation and Feature Extraction block (**SFE**) performs a frame segmentation and extracts the feature vector **f**. The Incremental Block (**IB**) regulates the increase of the number of gas particles used in the classification.

A block diagram of the modified algorithm is shown in Fig. 2. When a new input is presented (i.e. a vector of features $\mathbf{f}$ with a class n from the Segmentation and Feature Extraction (**SFE**), pre-processing block) the Incremental Block (**IB**) performs a testing phase and then checks the distances $\widetilde{E}(x)$ for all the ellipsoids already instantiated. Therefore, when a pattern is chosen for learning purposes, a test on the error is performed:

$$min(\widetilde{E}(x)) \leq E_{th}, \tag{16}$$

where E_{th} is a distance threshold used as a trade-off between accuracy and computational complexity. If inequality (16) is not satisfied a new instance in the code-book is produced. Moreover, the new ellipsoid is initialized and its Mahalanobis distance from the current pattern is set to zero.

The time variable is then no more a global iteration counter but becomes a variable associated to each particle. Therefore, the ellipsoid j at time t would have age $a_j(t)$ and its specific learning parameter are ε_j and ϱ_j as in the following equations:

$$\begin{aligned} \varepsilon_j(t) &= \varepsilon_{init}(\frac{\varepsilon_f}{\varepsilon_{init}})^{\frac{a_j(t)}{a_{max}}} \quad with \quad \varepsilon_{init} \geq \varepsilon_f, \\ \varrho_j(t) &= \varrho_{init}(\frac{\varrho_f}{\varrho_{init}})^{\frac{a_j(t)}{a_{max}}} \quad with \quad \varrho_{init} \geq \varrho_f, \end{aligned} \tag{17}$$

where a_{max} is the maximum ellipsoid age.

This mechanism allows to selectively force the convergence of each code-book ellipsoid and therefore it is possible to cope with a dynamically changing training set.

C. *Pruning*

One of the problems introduced with the incremental learning mechanism is the possible growing of the number of particles that can appear during learning but that cannot be eliminated.

In order to achieve a fast convergence maintaining the forgetting capability and performances increasing over long time simulation, a pruning strategy has been implemented.

Thanks to the supervised classification process, each time one particle is selected as the nearest to the presented input pattern (its distance is under the threshold E_{th}), a recall phase is performed and the resulting class is compared with the real one contained in the input pattern. Considering that the class number is a discrete variable while algorithmic outcomes are continuous, a classification error means that the distance

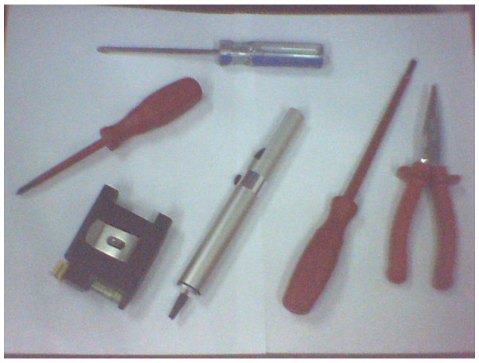

Fig. 3. Image of all laboratory tools used to create the input data set. Image has been acquired at VGA resolution ($640px$x$480px$)

between the reconstructed class number n_z and the pattern class number n_t exceeds a user defined threshold ($c_{th} \geq 0$)

$$|n_z - n_t| \geq c_{th} \tag{18}$$

In this way for each particle j, at time t, it is possible to compute an error rate $e_j(t)$ as follows:

$$e_j(t) = \frac{n_{err,j}(t)}{n_{wins,j}(t)}, \tag{19}$$

where $n_{err,j}$ is the number of times the particle j was the nearest during recalls and test phase led to a classification error; $n_{wins,j}$ is the total number of times the particle j was the first ranked among the others.

After code-book j has expired its life (i.e. $a_j \geq a_{max}$), if its error rate (e_j) is above a tolerance threshold (r_{th}) the corresponding particle is erased. In the other cases it is held frozen with a minimum learning parameter (i.e. $\varepsilon = \varepsilon_f$ and $\rho = \rho_f$).

IV. Experimental Setup

Experimental images have been acquired to test the algorithm performances. In order to consider a realistic visual flow on autonomous robot, both resolution and overall quality of the images have been kept low (i.e. no particular light conditions nor lens quality have been considered to ensure images with high saturation, sharpness and contrast). Testing objects have been chosen from common laboratory tools (see an example in Fig. 3). Visual cues have been extracted with OpenCV library [12] and a custom made simple segmentation algorithm through a host PC with low-resolution webcam ($640px$ x $480px$). It must be remarked that considering a frame rate of $30fps$, at the chosen resolution, simple segmentation routines can be easily performed in less than $t_s = \frac{1}{60}s$ in modern PC leaving at least $t_l = \frac{1}{60}s$ ($t_l = t_{fps} - t_s$) for classification learning. A reduced feature-space of the outcomes of the segmentation process is shown in Fig. 4 with one grey-scale value per each class.

V. Experimental Results and Comparisons

The input data set $\mathbf{T} = \{i_1, i_2, ..., i_N\}$ has been built of $N = 1300$ different images where only one object is present each time. Six different classes were chosen, one for each

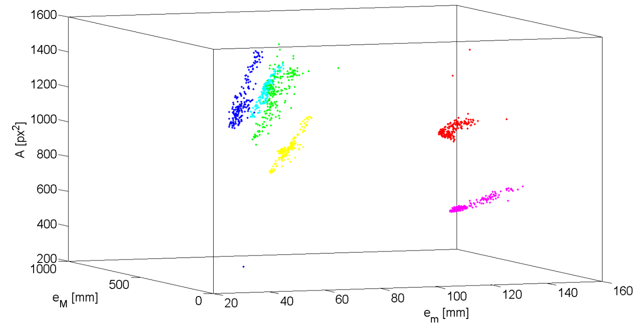

Fig. 4. Reduced feature-space obtained after segmentation. Depicted dimensions have been reduced from six to three (i.e. e_m, e_M and A). Each grey-scale value stands for a different class. It can be noticed that patterns from all three scree-drivers have reduced Euclidean distance in this feature sub-space (on the left).

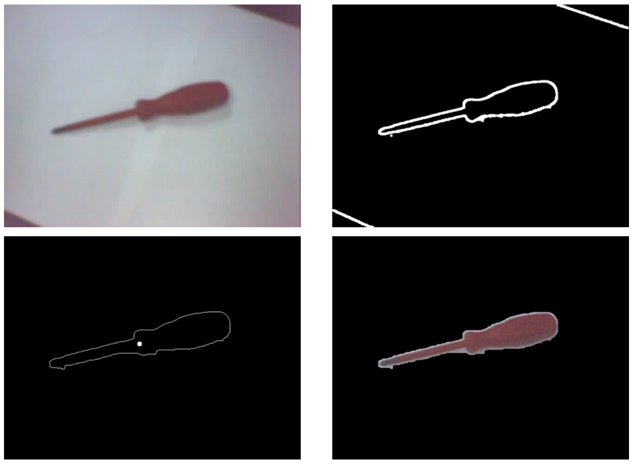

Fig. 5. Flow of the segmentation process. First a gaussian filter is applied (top-left), then the Canny algorithm generates a binary image (top-right) and finally meaningful contours are isolated (bottom-left). Segmented object area is determined and the same portion of the original image is extracted for color space based statistical analysis (bottom-right).

object present in Fig. 3. It is important to be noticed that three similar screw-driver were selected. As it is possible to see in Fig. 4, correspondent feature pattern clusters are very near (e.g. with and Euclidian distance definition) therefore the classification task is not trivial.

The typical segmentation flow performed by the PC is depicted in Fig. 5. The camera image is preprocessed with a gaussian filter, then a hue-based threshold and Canny segmentation [13] is applied and then meaningful objects are extracted through contours selection. During a testing phase, at the time t, we considered the error index $e_T(t)$ that can be computed basing on classification errors as follows:

$$e_T(t) = \frac{n_{err,T}(t)}{a_{max}}, \tag{20}$$

where $n_{err,T}$ is the number of classification errors occurred in the time window $[t - a_{max}, t]$.

The parameters used in this experiment are: $\varepsilon_{init} = 0.5$, $\varepsilon_f = 0.02 \cdot \varepsilon_{init}$, $\rho_{init} = 0.01$, $\rho_f = 0.02 \cdot \rho_{init}$ and $a_{max} = 3500$. Considering that in each frame a meaningful

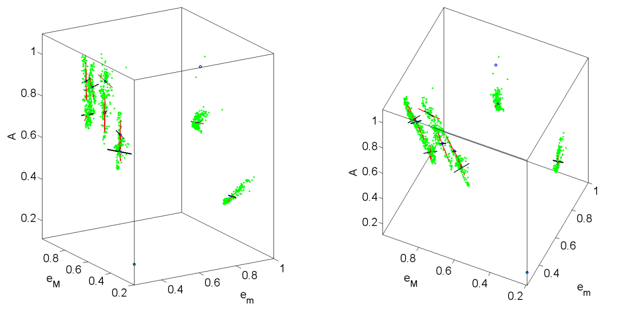

Fig. 6. Two different views of the reduced normalized features space (e_m, e_M and A) together with code-book ellipsoids after learning.

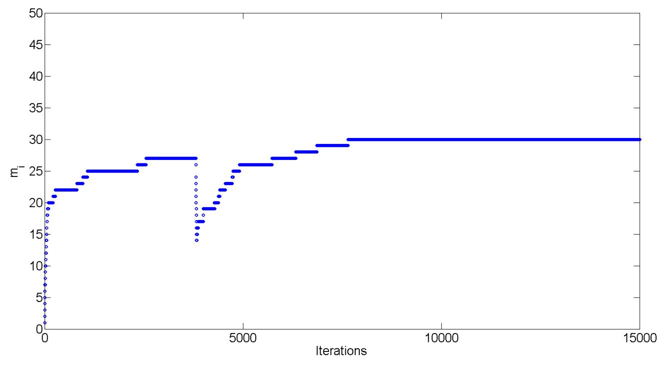

Fig. 7. Number of code-book ellipsoids (m) through learning with respect to the time (algorithm iteration number). The number of particles decrease for a while, around iteration 3500, due to the pruning strategy that allows each j particle to be erased when $a_j \geq a_{max}$.

pattern is present, the algorithm converges in less than 500 iteration to an error $e_T \leq 0.07$ with an average computational time for each iteration cycle (including learning and recall) of $t_l = 5\ ms$. Calculations were performed on a laptop double core 2.2GHz with 2GB RAM. An example of how particles look like after learning is shown in Fig. 6. As briefly introduced in the previous sections starting from a number of ellipsoid $m(0) = 0$ as initial condition, $m(t)$ increases through iteration as shown in Fig. 7. As it is possible to see in Fig. 7, the maximum number of particles used to classify the objects is about $m = 30$. This number is quite low with respect to the total number of different training patterns presented to the algorithm ($N = 1300$). Considering one learning algorithm iteration at each acquired frame with an image frame rate of $30fps$ ($t_{fps} = \frac{1}{30}\ s = 33.3\ ms$), an elaboration time $t_s \leq 28\ ms$ ($t_s = t_{fps} - t_l$) can be used for segmentation and other control algorithm. Moreover, the same parameters lead toward a $e_T \leq 0.015$ in $2500\ iterations$ ($t = 12.5s$ of computational time without image acquisition and segmentation in our hardware setup) and $e_T \leq 0.005$ in $15000\ iterations$ (i.e. $t = 75s$).

In order to be comparable even in supervised and semi-supervised problem solving, the classic SOM has been trained with a discrete class parameter and a test phase has been introduced at each iteration.

For a given pattern $\mathbf{x}$, the Best Matching Unit j^* (BMU, i.e. the nearest unit under Euclidian metric) is chosen just using distance on the features part $\mathbf{f}$ of the $\mathbf{x}$ vector as follows.

$$j^* = argmin_j(\mathbf{f}_j - \mathbf{f}_x) \quad with\ j = 1, 2, ..., m_S, \tag{21}$$

with m_s number of neurons in the Self-Organizing Map. The classification error is then evaluated on the class part of the vector (i.e. n parameter) between the BMU and the presented pattern as in inequality (18).

SOM with a relatively large number of neurons (i.e. SOM50 with 50×50 neurons and SOM200 with 200×200) have been tested.

The same error value e_T has been calculated, according to equation (20), during learning in supervised classification with SOMs.

It is possible to observe that, in our simulations, the larger is the number of neurons of the map the smaller is the overall classification error. SOM with 40000 neurons performs largely worse than NGPCA. The error decrease in the first part of the learning for the considered algorithm is depicted in Fig. 8. Table II summarizes results comparison between different structures through learning process,

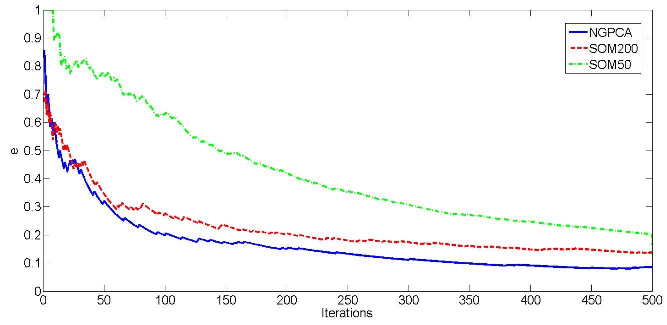

Fig. 8. Error indexes e_T comparison between implemented NGPCA (solid line) and Self-Organizing Map approaches with 50 by 50 neurons (dashed-line) and 200 by 200 neurons (dotted-lint) through learning iteration.

TABLE II
PERFORMANCE ANALYSIS TABLE

	NGPCA		SOM	
i	m_i	e_T [%]	$e_{T,S50}$ [%]	$e_{T,S200}$ [%]
500	22	7	21	14
2500	26	1.5	12	7
15000	30	0.5	4.5	2

where $e_{T,S50}$ and $e_{T,S200}$ are respectively the classification error rate calculated in SOM50 and in SOM200 maps.

It must be noticed that for a correct algorithm convergence in SOM many parameters have to be chosen. This implies that this kind of approach needs a temporal horizon that fixes the period of time in which different meaningful pattern have to be presented. In this way, comparisons are performed in the NGPCA worst case: all patterns are presented long before the SOM starts freezing. In real case in a robotic application the converging time could not be determined.

The proposed incremental on-line version of NGPCA with growing and pruning mechanisms solves this problem completely.

The complexity of the NGPCA algorithm is $O(m \cdot q)$ with m number of code-book ellipsoids and q is the number of principal components.

VI. Conclusions

In this work an on-line classification application of the Neural Gas with Principal Component Analysis is presented. Both the algorithm and the whole experimental setup have been chosen for straightforward porting of the architecture to a PC-based object manipulation in a roving platform. Under this point of view, the error-to-performances trade-off of the learning process is selectable through parameter tuning and the overall computational cost of the algorithm can be kept low, even for a embedded version of the classifier. The model input-output variables can be changed at any time for a given learning. A first comparison with standard structures, like Self-Organizing Maps, shows a higher level of flexibility without error nor complexity increase. Nevertheless, for a complete classification capability, focused on robot action-perception closed loop application, the proposed algorithm should be further developed in a fully unsupervised classifier.

Acknowledgements

The authors acknowledge the support of the European Commission under the project ICT 216227 - SPARK II "Spatial-temporal patterns for action-oriented perception in roving robots II: an insect brain computational model".

References

[1] Martinetz, T. M., Berkovich, S. G., and Schulten, K. J., "Neural-Gas network for vector quantization and its application to time-series prediction", *IEEE Transactions on Neural Networks*, 4, pp. 558-569, 1993.

[2] Tipping, M. E., Bishop, C. M. , "Mixtures of probabilistic principal component analyzers", *Neural Computation*, vol. 11, pp. 443-482, 1999.

[3] Möller R. and Hoffmann, H. , "An extension of neural gas to local PCA", *Neurocomputing*, vol. 62, 305-326, 2004.

[4] Hoffmann, H., Möller, R. , "Unsupervised learning of a kinematic arm model", *Artificial Neural Networks and Neural Information Processing*, vol. 2714, (ICANN/ICONIP), LNCS, Kaynak O, Alpaydin E, Oja E, Xu L, Springer, Berlin, pp. 463-470, 2003.

[5] Kohonen, T. , "Self-Organizing Maps" *Springer*, Berlin, 1995.

[6] H. Cruse and U. Steinkuhler, "Solution of the direct and inverse kinematics problems by a common algorithm based on the mean of multiple computations", *Biol. Cybernetics* 69, pp. 345-351 2, 1993.

[7] P. Arena and L. Patané, "Spatial Temporal Patterns for Action Oriented Perception in Roving Robots", Springer, Series: *Cognitive Systems Monographs*, vol. 1, 2009.

[8] EU Project SPARK II, website online at *www.spark2.diees.unict.it*

[9] Hinton, G. E., Dayan, P., and Revow, M. "Modeling the manifolds of images of handwritten digits", *IEEE Transactions on Neural Networks*, vol. 8, pp. 65-74, 1997.

[10] Möller, R. "Interlocking of learning and orthonormalization in RRLSA". *Neurocomputing*, vol. 49, pp. 429-433, 2002.

[11] Ouyang, S., Bao, Z., Liao, G.S. , "Robust recursive least squares learning algorithm for principal component analysis", *IEEE Transactions on Neural Networks*, vol. 11(1), pp. 215-221, 2000.

[12] OpenCV website homepage online at *http://opencv.willowgarage.com*

[13] Bill Green, "Canny Edge Detection Tutorial", 2002 online at *www.pages.drexel.edu*