ANNIIP 2010

Kurosh Madani (Ed.)

Artificial Neural Networks and Intelligent Information Processing

Proceedings of ANNIIP 2010
6th International Workshop on
Artificial Neural Networks and Intelligent Information Processing
In conjunction with ICINCO 2010
Funchal, Madeira - Portugal, June 2010

Kurosh Madani (Ed.)

Artificial Neural Networks and Intelligent Information Processing

Proceedings of the
6th International Workshop on
Artificial Neural Networks and Intelligent Information Processing
Workshop ANNIIP 2010

In conjunction with ICINCO 2010
Funchal, Madeira, Portugal, June 2010

SciTePress
Portugal

Volume Editors

Kurosh Madani
University PARIS-EST / PARIS 12
France

6th International Workshop on
Artificial Neural Networks and Intelligent Information Processing
Funchal, Madeira, Portugal, June 2010

Printed in Portugal

ISBN: 978-989-8425-03-4
Depósito Legal: 311151/10

Foreword

Theoretical, applicative and technological challenges, emanating from nowadays' industrial, socioeconomic or environment needs, open every day new dilemmas to solved and new challenges to defeat. Bio-inspired Artificial Intelligence and related topic have shown its astounding potential in overcoming the above-mentioned needs. It is a fact and at the same time a great pleasure to notice that the ever-increasing interest of both confirmed and young researchers on this relatively juvenile science, upholds a reach multidisciplinary synergy between a large number of scientific communities making conceivable a forthcoming emergence of viable solutions to these real-world complex challenges.

Since its first edition in 2005, ANNIIP international workshop, within the frame of the prestigious ICINCO International Conference, takes part in appealing intellectual dynamics created around bio-inspired Artificial Intelligence by offering a privileged space to refit and exchange the knowledge about further theoretical advances, new experimental discoveries and novel technological improvements in this promising area. The present book is the outcome of the sixth edition of this annual event.

Within this inveterate philosophy and around a deliberately limited number of papers, the objective of this fifth volume is to convene once more relevant recent works focusing this exciting topic, related fields and issued applications. Conformably to our values, the choice of publishing a restricted number of papers is persistently motivated on the one hand by the premeditated desire to give a large space to exchanges and discussions during the workshop, and on the other hand by the strong principle of the presentation of each accepted article by its authors. If "Bio-inspired Artificial Intelligence" and its real-world applications remain, as in the previous editions of this international workshop, the foremost premises of this sixth volume, a special attention has been devoted to the balance between theoretical and applicative aspects.

It is important to remind that scientific relevance and technical excellence of a collective volume emerge from quality of its contributors: those who have contributed by the high quality of their manuscripts and those who have taken part in reviewing of submitted papers ensuring, by their valuable expertise, the distinction of the present book. I would like to express again my acknowledge-

ments to contributors of all accepted papers: You are the central reason of the nobles of this tome. Also, I would like to reedit my gratitude to Reviewing Board and Program Committee for the valuable work that they accomplished: My heartfelt recognition to those who already were members of ANNIIP Program Committee as well as my sincere thanks to those who kindly accepted to enlarge the Reviewing Board of this sixth workshop's edition.

It is also essential to be reminiscent that creative dynamics is frequently the result of fruitful humans' contacts within a same scientific field or the consequence of humans' interactions from different scientific communities and since 2004, the date of the its first edition, the ICINCO multi-conference has been an outstanding bench of such creative synergies. For that, again, I would like to express my warm appreciation and my particular gratitude to my friend Prof. Joaquim Filipe, ICINCO 2010 Conference's Chair, for his faith in young science of "Bio-inspired Artificial Intelligence" and for his reliance on devoting once more this privileged space to the ANNIIP workshop within his valuable conference.

Finally, if ICINCO Organizing Committee's professionalism became an obvious skill of this international event's organization in so accurate way, it should never be forgotten that the organization of a prestigious conference remains a challenging undertake requiring a reliable and a solid team. So, I would like to acknowledge whole the organizing team, with a special word for Marina Carvalho and Vitor Pedrosa from Workshops Secretariat who, during the six revolved years, have proved their irreplaceable merit as key persons in ANNIIP workshop's organization.

June 2010,

Kurosh Madani
University PARIS-EST / PARIS 12, France

Workshop Chairs

Kurosh Madani
University PARIS-EST / PARIS 12
France

Program Committee

Veronique Amarger, PARIS-EST University, France
Ezzedine Ben Braiek, Ministery of High Studies, Tunisia
Abdennasser Chebira, LISSI, France
Amine Chohra, Images, Signals, and Intelligent Systems Laboratory (LISSI / EA 3956), Paris-East University, France
Suash Deb, C. V. Raman College of Engineering, India
Khalifa Djemal, University of Evry Val d'Essonne, France
Vladimir Golovko, Brest State Technical University, Russian Federation
Maria Del Carmen Hernandez Gomez, Grupo de Inteligencia Computacional Computational / UPV/EHU, Spain
Manuel Grana, Universidad del Pais Vasco, Spain
Robert Hiromoto, University of Idaho, U.S.A.
Dattatraya Kodavade, DKTE Society's Textile & Engineering Institute Ichalkaranji, India
Thomas Laengle, Fraunhofer Iitb, Germany
Kurosh Madani, University PARIS-EST Creteil (UPEC), France
Georgy Panev, SPIIRAN - Russian Academy of Science, Russian Federation
Christophe Sabourin, Signals, Images, and Intelligent Systems Laboratory, France
María Teresa García Sebastián, University of the Basque Country (UPV/EHU), Spain
Moustapha Séne, Université Gaston Berger de Saint-Louis (Sénégal), Senegal
Eva Volna, University of Ostrava, Czech Republic
Qiangfu ZHAO, Aizu, University, Japan

Table of Contents

Papers

PAPERS

Finding Fuzzy Communities in Directed Networks

Kun Zhao, Shao-Wu Zhang and Quan Pan

School of Automation, Northwestern Polytechnical University, Xi'an, China
zz_kk@126.com, {zhangsw, quanpan}@nwpu.edu.cn

Abstract. To comprehend the directed networks in a fuzzy view, we introduce a new matrix decomposition approach that reveals overlapping community structure in weighted and directed networks. This method decomposes a directed network into modules by optimally decomposing the asymmetric feature matrix of the directed network into two matrices separately representing the closeness degree from node to community and the closeness degree from community to node. Their combined result uncovers the community structures in a fuzzy sense in the directed networks. The illustrations on an artificial network and a word association network give reasonable results.

1 Introduction

A cogent module representation of a network will retain the important information about the network and highlight the underlying structures and the relationships in the network. Many researches have been devoted to the development of algorithmic tools for discovering communities [1]. Nearly all of these methods, however, are not intended for the analysis of directed network. Yet, directedness is an essential feature of many real networks. Ignoring direction may reduce considerably the information that one can extract from the network structure. In particular, neglecting link directedness when looking for communities may lead to partial, or even misleading, results. Very few algorithms [2], [3], [4] currently available are able to handle directed graphs, since the presence of directed links places a serious obstacle towards community detection problems.

Another subject that attracts much attention in network studies is the detection of overlapping communities, or fuzzy clustering. Specifically, many real world networks exhibit an overlapping community structure, which is hard to grasp with the classical graph clustering methods [5], [6], [7] where every node of the graph belongs to exactly one community. Up to now, only a small number of studies [8], [9], [10] have addressed the problem of overlapping community. Typically, there is an algorithm takes symmetrical non-negative matrix factorization (s-NFM) [10] into optimization framework and achieves explicit physical meaning for the clustering results, which are helpful for the network analysis after clustering. As for directed networks, however, symmetrical factorization could not treat the asymmetry which is resulted from the directedness in edges. To solve this problem, we constructed a new optimization framework based on the approximation to the directed feature matrix with matrices of two types of directed paths. In order to complete the framework, we also proposed a directfied and fuzzified variant of the modularity function first

introduced by Newman [11]. New function provides a reasonable basis for the determination of the optimal number of communities. The clustering results contain abundant information and equally possess explicit physical meaning. We tested our method on a computer-generated graph and a real-world graph and gained significant and informative community divisions in both cases.

2 The Algorithm

2.1 Optimization Scheme for Directed Graphs

Consider a directed and weighted network $G(N,E)$, which can be described by the weighted adjacency matrix $A=[A_{ij}]_{n\times n}$ where n is the number of nodes, and $A_{ij}>0$ if and only if $(i,j)\in E$ and 0 otherwise. Let the feature matrix of G be $Y=[Y_{ij}]_{n\times n}$ where Y_{ij} denotes the similarity from node i to node j. Note that the relationship between a pair of nodes is easy to grasp in the sense of connecting path. As the path linking a pair of nodes increases, the relationship of the pair is enhanced. Then, in this paper, we make the path number as the central metric of various relationships in network.

Undirected graph is defined as a graph in which edges have no orientation. It is, therefore, no need to distinguish between the paths that start from a given node and the paths that arrive in it because they are essentially the same in undirected graphs. However, in directed graphs, these two types of paths are usually not equivalent since not all edges are bidirectional. Suppose that n nodes can be grouped into r overlapping communities. Here we introduce the concept of node-community similarity matrix $U^{\circ}=[U^{\circ}_{ik}]_{n\times r}$, which is non-negative, to represents the number of paths (or the similarity degree) from nodes to communities, and the concept of community-node similarity matrix $V^{\circ}=[V^{\circ}_{ik}]_{n\times r}$, which is non-negative, to represents the number of paths (or the similarity degree) from communities to nodes. Generally, U° concerns the outgoing edges of node, and V° concerns the incoming edges of node. Fig. 1 illustrates the difference between the two types of paths in directed network.

Fig. 1. Schematic illustrations of the two types of paths in directed graph (community → node and node → community).

Since U° and V° respectively denote the number of paths from node to community and the number of paths from community to node, $U^{\circ}V^{\circ T}$ could further be an approximation of similarity between nodes. That is, we can use U° and V° to reconstruct Y:

$$U^{\circ}V^{\circ T} \to Y \tag{1}$$

For convenience, we hope to have the following approximation form:

$$\overline{U}\overline{S}\overline{V}^{T} \to Y \tag{2}$$

where $\overline{U}$ and $\overline{V}$ are non-negative matrix which are separately the column Frobenius normalization form of U° and V°, and the $r \times r$ non-negative diagonal matrix $\overline{S}$ stores the weights of the columns of $\overline{U}$ and $\overline{V}$. Note that Equation (1) is essentially equal to Equation (2), then

$$\overline{U}\overline{S}\overline{V}^{T} = U^{\circ}V^{\circ T} \tag{3}$$

Equation (2) leads us to the following Frobenius norm (Euclidean distance equation), which measures the fitness of the given matrices $\overline{U}$, $\overline{V}$ and $\overline{S}$ of graph *G(V,E)* by quantifying how precisely they approximate the network structural information *Y*:

$$\min_{W \geq 0} F_G(Y, \overline{U}, \overline{S}, \overline{V}) = \left\| Y - \overline{U}\overline{S}\overline{V}^{T} \right\|_{Fro}^{2} = \frac{1}{2}\sum_{ij}[(Y - \overline{U}\overline{S}\overline{V}^{T}) \circ (Y - \overline{U}\overline{S}\overline{V}^{T})]_{ij} \tag{4}$$

where $A \circ B$ means the Hadamard product (or element-by-element product) of matrices *A* and *B*.

Now the community detection problem is reduced to the optimization of F_G. In other words, we must find the optimal $\overline{U}$, $\overline{V}$ and $\overline{S}$ to minimize F_G. To solve this optimization problem, we will develope a modified Non-negative Singular Value Decomposition.

2.2 Method of Compressed Non-negative Singular Value Decomposition

Matrix factorization plays an important role in scientific computation. The commonly used one is singular value decomposition (SVD) [12]. It approximates one matrix with three lower rank matrices (including one rectangular matrix and two square matrices) with orthogonality constraints, in which the left and right singular vectors correspond to the column and row spaces of the original matrix. SVD has been successfully applied in both science and engineer areas [13]. However the results by SVD on real data always lose physical meaning because they usually contain negative values, and this can not be interpretable easily from intuitive insight. To make the results more interpretable, Liu [14] took the non-negativity constraints into SVD and developed a non-negative SVD (NNSVD).

NNSVD is very help for our theme. We only need to make appropriate modification on it. Firstly, both of $\overline{U}$ and $\overline{V}$ are not square matrices and they do not have orthogonality constraints. Secondly, $\overline{U}$ and $\overline{V}$ must be achieved by the normalization after matrix factorization. We take the above conditions in NNSVD and propose the iteratively update rules of a compressed Non-negative Singular Value Decomposition (c-NNSVD):

$$\begin{cases} U_{k+1} = U_k \circ \dfrac{[YV_kS_k]}{[U_kS_kV_k^TV_kS_k]} \\ S_{k+1} = S_k \circ \dfrac{[U_kYV_k]}{[U_k^TU_kS_kV_k^TV_k]} \\ V_{k+1} = V_k \circ \dfrac{[Y^TU_kS_k]}{[V_kS_kU_k^TU_kS_k]} \end{cases} \tag{5}$$

where U_k and V_k are $n \times r$ non-negative matrix and S_k is $r \times r$ non-negative diagonal matrix. The iteration starts from random matrices which are chosen from a normal distribution with mean 0, variance 1. $\overline{U}$ and $\overline{V}$ are obtained by the column normalization of *U* and *V* which are the optimal solution of iteration rules in Equation (4); and the non-negative diagonal matrix $\overline{S}$ stores the weights of the columns of $\overline{U}$ and $\overline{V}$. The *i*'th diagonal element of $\overline{S}$ corresponds to the *i*'th community, or the *i*'th column of matrix $\overline{U}$ and $\overline{V}$.

According to Equation (3), to gain the number of paths from nodes to communities (U°) and the number of paths from communities to nodes (V°), the weights in $\overline{S}$ should be properly assigned to U° and V°. Intuitively, the weight of the input paths of node should be quantitively equated with the weight of the output paths of node. Therefore, the weights in $\overline{S}$ are equally distributed by:

$$\begin{cases} U^\circ = \overline{U}\overline{S}^{\frac{1}{2}} \\ V^\circ = \overline{V}\overline{S}^{\frac{1}{2}} \end{cases} \tag{6}$$

By Equation (6), we gain the number of paths from nodes to communities and the number of paths from communities to nodes. Note that their sum can produce an integrated closeness degree between nodes and communities, which is necessary for the directed network analysis:

$$W^\circ = U^\circ V^\circ = (\overline{U} + \overline{V})\overline{S}^{\frac{1}{2}} \tag{7}$$

If one do not want to separately consider U° and V°, the integrated quantity, W°, would give a consolidated result which combines the two directions. To specifically illustrate the difference among U°, V° and W°, we apply our method on a 11-nodes network studied in [3], as follows:

Let *r* have a value of 2; the output of Equations (5),(6) and (7) is:

$$U^{\circ T} = \begin{bmatrix} 0.533 & 0.533 & 0.533 & 0.550 & 0.550 & 0.109 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.004 & 0.004 & 0.163 & 0.543 & 0.543 & 0.533 & 0.533 & 0.533 \end{bmatrix}$$

$$V^{\circ T} = \begin{bmatrix} 0.533 & 0.533 & 0.533 & 0.543 & 0.543 & 0.163 & 0.004 & 0.004 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.109 & 0.550 & 0.550 & 0.533 & 0.533 & 0.533 \end{bmatrix}$$

$$W^{\circ T} = \begin{bmatrix} 1.066 & 1.066 & 1.066 & 1.093 & 1.093 & 0.272 & 0.004 & 0.004 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.004 & 0.004 & 0.272 & 1.093 & 1.093 & 1.066 & 1.066 & 1.066 \end{bmatrix}$$

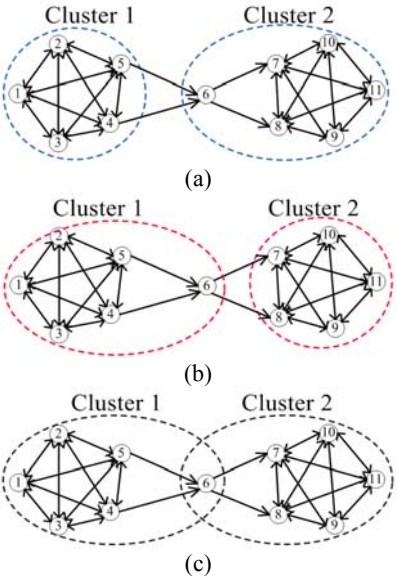


Fig. 2. Community assignments for 11-nodes network from the results of (a) U°, (b) V° and (c) W°.

Based on the above results, it is not difficult to find that the status of node 6 in U° and V° are the exact opposite of each other. In the result from U°, since there is more number of paths from node 6 to community 2 than that to community 1, node 6 is assigned to community 1, as Fig. 2(a) shows. In the result from V°, since there is far less number of paths from community 2 to node 6 than that from community 1 to node 6, node 6 is assigned to the community 2, as Fig. 2(b) shows. However, in the result from W°, the closeness degrees of node 6 to both communities are exactly the same, as Fig. 2(c) shows. Therefore, W° is the integrated similarity degree which combines the degrees on two directions. In general cases, we use W° to provide the clustering result in our framework.

2.3 Feature Matrix

Choosing a feature matrix to store the topological information of a network is a fundamental problem. Here, we select diffusion kernel [15] as the feature matrix. Some graph kernels have been developed [16] to decipher the topological relationships that are implicit in the graph data and make them explicit. One of these is known as diffusion kernel which captures the long-range relationship between nodes through enumerating the number of paths connecting them [16].

Firstly, we review the diffusion kernel of a common undirected network. The Laplacian of undirected network is the following matrix:

$$L_{ij} = \begin{cases} A_{ij}, & i \neq j \\ -d_i, & i = j \end{cases} \tag{8}$$

where d_i is the degree of node i. Diffusion kernel , the exponential of matrix L, is defined as:

$$K = \exp(\beta L) = \lim_{n\to\infty}\left(1+\frac{\beta L}{n}\right)^n = 1+\beta L+\frac{\beta^2}{2}L^2+\frac{\beta^3}{3!}L^3+\cdots \tag{9}$$

where β is a positive constant to control the degree of diffusion. In undirected networks, the resulting matrix K is symmetric and positive definite. It is a valid kernel. A similarity matrix Y can be obtained by normalizing the kernel matrix K in such a way:

$$Y_{ij} = \frac{K_{ij}}{\sqrt{K_{ii}K_{jj}}} \tag{10}$$

Note that, the diffusion kernel of undirected network starts with a symmetric adjacency matrix. However, in directed network, the adjacency matrix A is not symmetric. Therefore, directed networks should have an alternative form of kernel which could be traced back to a different Laplacian. The Laplacian of undirected and weighted network is the following matrix:

$$L_{ij}^{d} = \begin{cases} A_{ij}, & i \neq j \\ -d_i^{out}, & i = j \end{cases} \tag{11}$$

where d_i^{out} is the out degree (weighted) of node i. It is naturally an asymmetric matrix. So, its kernel matrix K^d=exp(βL^d) and the resulted feature matrix are also asymmetric. Frankly speaking, K^d do reflect the number of directed paths from one node to another in an asymmetric manner. In this paper, we choose β= 0.1 in the feature matrices in our study.

2.4 Directfied and Fuzzified Variant of the Modularity Function

If a priori knowledge of the community number is absent, the optimal number of communities should be determined by some computational methods in a self-consistent way without human intervention. Recently, a concept of modularity function Q introduced by Newman and Girvan [11] has been broadly used as a valid measure for community structure. It comes from the notion that: only if the number of edges within communities is significantly higher than would be expected purely by chance can we justifiably claim to have found significant community structure. The original modularity of a network is then defined as:

$$Q = \frac{1}{2m}\sum_{i,j}\left[A_{ij} - \frac{d_i d_j}{2m}\right]\cdot\delta_{C_iC_j} \tag{12}$$

where A_{ij} is an element of the adjacency matrix, δ_{ij} is the Kronecker delta symbol, and c_i is the label of the community to which vertex i is assigned.

Then one maximizes Q over possible divisions of the network into communities, the maximum being taken as the best estimate of the true communities in the network. So, the optimal number of fuzzy communities can be determined by the modularity function Q which gains its maximum on a certain value of r.

In the fuzzy clustering method of Nepusz [8], a fuzzified variant of the modularity Q is presented as:

$$Q_f = \frac{1}{2m}\sum_{i,j}\left[A_{ij} - \frac{d_i d_j}{2m}\right]\cdot s_{ij} \tag{13}$$

where $s_{ij} = \sum_{k=1}^{r} H_{ki}H_{kj}$ and H_{ki} is the fuzzy membership degree of node i to the community k. The probability of the event that vertex i belongs to the same community as vertex j becomes the dot product of their membership vectors, resulting in the similarity measure s_{ij}, which can be used in place of $\delta_{C_iC_j}$ to obtain a fuzzified variant of the modularity.

As for directed network, Newman [2] presented a new modularity function Q_d, which is generally applicable for directed networks:

$$Q_d = \frac{1}{M}\sum_{i,j}\left[A_{ij} - \frac{d_i^{out} d_j^{in}}{M}\right]\cdot \delta_{C_iC_j} \tag{14}$$

where A_{ij} is defined in the conventional manner to be 1 if there is an edge from j to i and zero otherwise, and d_i^{out} is the out-degree of node i and d_j^{in} is the in-degree of node j and M is the total number of directed edges in the network. Indeed edge i-j make larger contributions to this expression if d_j^{in} and/or d_i^{out} is small.

Each of the above two modularity, Q_f and Q_d, has its advantages which the other one does not has. To combine their advantages, we propose another variant of the modularity Q as:

$$Q_{df} = \frac{1}{M}\sum_{i,j}\left[A_{ij} - \frac{d_i^{out} d_j^{in}}{M}\right]\cdot s_{ij} \tag{15}$$

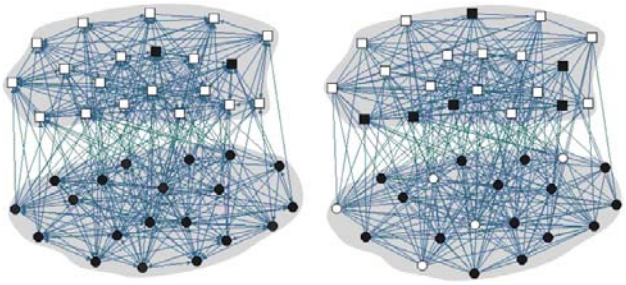
which can be applied to fuzzy clustering in directed networks.

The modularity can be either positive or negative, with positive values indicating the possible presence of community structure. One can search for community structure precisely by looking for the divisions of a network that have positive, and preferably large, values of the modularity. In order to determine the optimal number of fuzzy communities in directed networks, we iteratively increase r and choose the one which results in the highest modularity Q_{df}.

3 Test of the Method

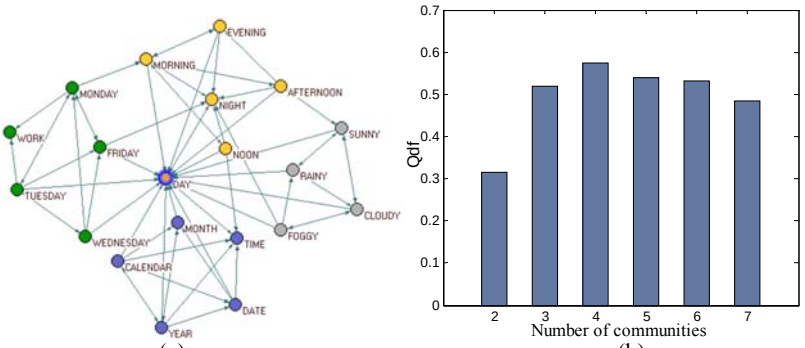
3.1 Random Graph

For illustrative purposes, we consider an artificial computer-generated network, designed specifically to test the performance of the algorithm. As Fig. 3a shows, this network is generated with N = 40 nodes, split into two communities containing 20 nodes each. We put 120 directed edges in each community at random and 120 directed edges between the two communities at random. The edges that fall within groups are biasedly assigned directions so that they are more likely to point from one group to another. As we apply c-NNSVD on this network, the two communities are detected almost perfectly: just two nodes out of 40 are misclassified. This is confirmed in Fig. 3(a), which shows the results of the application of our method. If we ignore the directions, however, using the algorithm presented in [10], there is nearly no community structure to be found in this network, as Fig. 3(b) shows.



(a) (b)

Fig. 3. Community assignments for the two-community random network described in the text from (a) the algorithm of this paper and (b) an undirected clustering algorithm in [10]. The true community assignments are denoted by vertex shape or shaded region. The different colors represent different communities obtained by the algorithms.



(a) (b)

Fig. 4. The communities of the word *DAY* in the South Florida Free Association coupled with the determination of the optimal number of communities. (a) By the method presented in this paper, the word DAY is discovered to be the overlapping node which has the largest membership degree to the yellow group and the second-largest membership degree to the blue group. (b) Histogram of Q_{df} for different choices on number of communities.

3.2 Word Association Graph

We examined a directed network obtained from the South Florida Free Association norms list [17] (containing 10617 nodes and 63788 links), where the weight of a directed link from one word to another indicates the frequency that the people in the survey associated the end point of the link with its start point. We picked a sub-network with 20 nodes from the list and chose 4 as the optimal number of clusters, see details in Fig. 4(b) which indicates that the peak for Q_{df} of 0.5745 is achieved at $r = 4$. For illustration in Fig. 4(a), we showed the (colour coded) modules of the word *DAY* obtained by c-NNSVD, with the overlap emphasized in nested color. According to its different meanings, this word participates in four, strongly internally connected modules. The green community can be associated with work days. The yellow community consists of day times, the gray community contains common adjectives of day related to weather, and the blue community can be associated with the calendar. Separately, the closeness degrees of the word *DAY* to the four communities is 0.018, 1.355, 0.019 and 0.059, which indicate that the yellow group is the dominant community of node *DAY* and the blue group follows.

4 Conclusions

In this paper we presented a new algorithm for identifying overlapping communities in directed networks based on two matrices of similarity between node and community. An integrated quantity was proposed to give a consolidated result and it was shown, through several examples that this leads to detection of the overlapping community structure of the directed network.

Acknowledgements

We thank Liu Weixiang for the useful discussion. The work was supported in part by the National Natural Science Foundation of China under Grant NO. 60775012 and NO. 60634030.

References

1. Danon, L., Duch, J., Diaz-Guilera, A. and Arenas, A. Comparing Community Structure Identification. Stat. Mech. (2005) P09008.
2. Leicht, E. A. and Newman, M. E. J. Community Structure in Directed Networks. Phys. Rev. Lett. (2008) 100, 118703.
3. Nepusz, T. and Bazsó, F. Likelihood-based Clustering of Directed Graphs. In: IEEE 3rd International Symposium on Computational Intelligence and Intelligent Informatics. (2007) Agadir, Marokkó, 28.
4. Palla, G., Farkas, I., Pollner, P., Derenyi, I. and Vicsek, T. Directed Network Modules. Phys. New. J. (2007) 186.

5. Girvan, M. and Newman, M. E. J. Community Structure in Social and Biological Networks. Proc. Natl. Acad. Sci. USA, (2002) 99, 7821−7826.
6. Ravasz, E., Somera, A. L. and Mongru. D. A. Hierarchical Organization of Modularity in Metabolic Networks. Science, (2002) 297, 1551−1555.
7. Capocci, A., Servedio, V. D. P., Caldarelli, G. and Colaiori, F. Detecting Communities in Large Networks. Physica A, (2005) 352, 669−676.
8. Nepusz, T., Petróczi, A., Négyessy, L. and Bazsó, F.. Fuzzy Communities and the Concept of Bridgeness in Complex Networks. Phys. Rev. E, (2008) 77, 1539-3755.
9. Palla, G., Derenyi, I., Farkas, I. and Vicsek, T. Uncovering the Overlapping Community Structure of Complex Networks in Nature and Society. Nature, (2005) 435, 814–818.
10. Zhao,K., Zhang, S. W. and Pan, Q.. Fuzzy Analysis for Overlapping Community Structure of Complex Network. In: IEEE International Conference on Chinese Control and Decision Conference(CCDC), Submitted for publication (2010).
11. Newman, M. E. J. and Girvan, M. Finding and Evaluating Community Structure in Networks. Phys. Rev. E, (2004) 69, 026113.
12. Golub, G. H. and Van Loan, C. F. Matrix Computations (3nd ed.). Johns Hopkins University Press. (1996)
13. Jolliffe, I. *Principal Component Analysis*. Springer (2002).
14. Liu, W., Tang, A., Ye, D., and Ji, Z. Nonnegative Singular Value Decomposition for Microarray Data Analysis of Spermatogenesis. Technology and Applications in Biomedicine. (2008) 225-228.
15. Kondor, R. I. and Lafferty, J. Diffusion Kernels on Graphs and Other Discrete Structures. 19th International Conference on Machine Learning (ICML), (2002)315−322.
16. Fouss, F. and Yen, L. An experimental Investigation of Graph Kernels on Two Collaborative Recommendation Tasks. In: IEEE International Conference on Data Mining (ICDM), (2006) 18-22.
17. Nelson, D. L., McEvoy, C. L., and Schreiber, T. A. The University of South Florida Word Association, Rhyme, and Word Fragment Norms. (1998) Retrieved from: http://www.usf.edu/FreeAssociation/.

Search Space Restriction of Neuro-evolution through Constrained Modularization of Neural Networks

Christian W. Rempis[1] and Frank Pasemann[1,2]

[1] Neurocybernetics Group, Institute of Cognitive Science
University of Osnabrück, 49069 Osnabrück, Germany
[2] Institute for Advanced Study, Wallotstr. 19, 14193 Berlin, Germany
{christian.rempis, frank.pasemann}@uni-osnabrueck.de
http://ikw.uni-osnabrueck.de/∼neurokybernetik/

Abstract. Evolving recurrent neural networks for behavior control of robots equipped with larger sets of sensors and actuators is difficult due to the large search spaces that come with the larger number of input and output neurons. We propose *constrained modularization* as a novel technique to reduce the search space for such evolutions. Appropriate neural networks are divided manually into logically and spatially related neuro-modules based on domain knowledge of the targeted problem. Then *constraint functions* are applied to these neuro-modules to force the compliance of user defined restrictions and relations. For neuro-modules this will facilitate complex symmetries and other spatial relations, local processing of related sensors and actuators, the reuse of functional neuro-modules, fine control of synaptic connections, and a non-destructive crossover operator. With an implementation of this so called ICONE method several behaviors for non-trivial robots have already been evolved successfully.

1 Introduction

The development of recurrent neural networks for behavior control of autonomous robots with evolutionary methods has a long and successful history [10], [4], [6]. Nevertheless, most experiments work with robots having only a small number of sensors and actuators, as in typical experiments described in [9], [8]. Although interesting non-trivial behaviors have to be expected to come up especially with complex robots having a larger number of sensors and actuators, only few experiments have been conducted in this domain. One main reason is that the search space for neuro-controllers gets inconveniently large if more and more sensor and motor neurons have to be used. This often makes it infeasible to evolve interesting solutions in reasonable time.

To cope with such large search spaces, strategies and heuristics have to be found that reduce the search space or that assist the experimenter to guide evolution towards effective network topologies. In this contribution, we propose that the manual segmentation of neural networks into smaller, *constrained* sub-networks, called *neuro-modules* [11][9], can significantly restrict the search space. This *constrained modularization* is based on domain knowledge about the behavior problem to be solved. The induced restrictions on the modules exclude large parts of the search space and focus the search on network topologies that have a higher chance to provide a desired solution. The kind

of solution hereby can be biased by the experimenter to a large extend during the modularization. Furthermore resulting network topologies often are better to understand than unconstrained ones, allow an easier identification of relevant network parts, and make the reuse of already evolved networks structures possible. With this approach the evolutionary algorithm is not used as a universal problem solver that creates complex networks from scratch. Instead evolution is used merely as a tool to help the experimenter to confirm his specific solution approaches, that are usually still too complex to be constructed by hand.

In the next chapter we define the terms *constrained modularization*, *neuron group* and *neuro-module* as they are used here. Then, in chapter 3, we describe how modularization of large neural networks can reduce the search space and why resulting solutions of modular neuro-evolution often are easier to understand. First indications of the usability of this approach, based on the implementation of this method, are discussed in chapter 4 and 5, followed by a conclusion in the final chapter.

2 Constrained Modularization of Neural Networks

2.1 Constrained Modularization

The decomposition of a recurrent neural network into smaller, hierarchically and spatially organized sub-networks is here called *modularization*. A network hereby is, based on domain knowledge and user experience, manually split into connected neuron groups by the experimenter (Fig. 1 shows an example). To each neuron group *functional constraints* can be added, that force the compliance of user defined limitations or structural restrictions. These constraint functions can implement any restriction and manipulate the neural network directly, so that violations of constrains, e.g. originating from mutation operators, can be counteracted immediately.

With this *constrained modularization* the user tries to restrict the network development in such a way, that only a certain, promising type of network structures is possible. Hereby the user constructs a kind of constraint mask for the neural network, that specifically limits the network topology and thus leads to a smaller search space for the evolutionary algorithm.

Neurons can be grouped in two different ways: (1) by simple neuron groups, or (2) by more restrictive neuro-modules. Neuron groups and neuro-modules both allow a detailed topological, hierarchical and functional partition of the network to exclude unwanted areas of the search space. Both types of grouping are defined in the next two sections.

2.2 Neuron Groups

The simplest way to group neurons is the creation of *neuron groups*. These groups may contain any number of neurons sharing topological, functional or other properties. Hereby neuron groups can arbitrarily overlap. Thus each neuron can be part of many neuron groups at the same time. Neuron groups can be target of *constraint functions*. These functions force the compliance of certain, user defined constraints, rules

and heuristics for their member neurons. This may include, for example, limiting the number of member neurons or synapses, forcing specific kinds of synaptic connection patterns, allowing synaptic plasticity for its members or resolving dependencies between neurons and synapses. The constraint functions therefore define the purpose of a group and contribute significantly to a search space restriction.

2.3 Neuro-modules

A stronger grouping of neurons is represented by so called *neuro-modules*. Neuro-modules are similar to neuron groups, but do not intersect partially; i.e., neurons can only be part of a single neuro-module at the same time. However, neuro-modules can be members of other neuro-modules and therefore can serve as *sub-modules*.

Neurons in a neuro-module are encapsulated by the module. Thus these neurons are only visible to the neurons of the same neuro-module. To connect the module with external neurons, it can provide a neural interface. This can be achieved by marking selected neurons of the module as *input* or *output* neurons (compare Fig. 1 where these neurons are marked with I and O). During evolution synaptic connections are inserted only between members of the same neuro-module and to interface neurons of sub-modules. Neuro-modules thus can be regarded as encapsulated, independent neural building-blocks with well defined interfaces. Their special purpose is to group strongly related neurons together (e.g. the sensors and motors of a joint or the neurons of a functional structure) and to control the way these neurons can connect to neurons outside of the module.

3 How Constrained Modularization Fosters Successful Evolutions

Modularizing and constraining a neural network according to domain knowledge of a behavioral problem can restrict the search space for neuro-evolution significantly. For comparison, an unconstrained minimal neuro-controller for walking of a humanoid robot with 42 motor and 37 sensor neurons already allows over 3300 synapses, while the same, but constrained modular network in Fig. 1 only allows 180 independent synapses. In this figure it can also be seen, that the modularized network is much more structured than an unconstrained one. It shows, that the constrained network already biases the possible network structures, here to get a symmetric network for walking based on internal oscillators or on an acceleration sensor (at the top module). This also shows that the initial networks for a neuro-evolution have to be specifically modularized regarding the given problem to be solved. Therefore, even for the same problem, different kinds of modularization promote different approaches to solutions. Experimenters can use this to bias the networks towards desired solution approaches.

Constrained modularization reduces the search space by constraining the structure, function and evolution process of the networks, as is described in the next sections.

3.1 Structure Constraints

Neuro-modules, with their ability to hierarchically structure a network and to shield their members from disruptive connections from arbitrary sources, bias the network

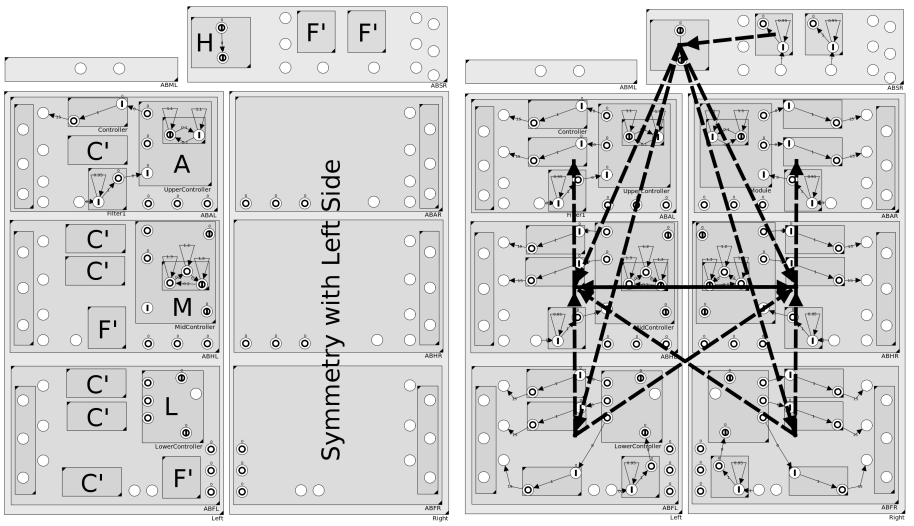


Fig. 1. Left: Constrained modularization of the control network of a humanoid robot. The 37 sensor and 42 motor neurons are separated into modules according to their locations on the robot (head, arms, middle body, legs). A symmetry constraint handles the neural structure of the right side. The upper left module was extended by an evolvable controller module and a filter module. Clones of these modules (*C' and F'*) have been added to each used motor neuron and acceleration sensor. Additional functional modules have been added (*H, A, M, L*) that can be exchanged during crossover by modules of the same type. These modules are expected to implement the actual behavior controller. During evolution neurons are only added to these functional modules. In (A) and (M) oscillator modules have been added that might be modified and incorporated into the control network. **Right:** This network is the result of executing the constraints for the left network. To additionally restrict the search space, synaptic pathways (*Black Dotted Lines*) have been added.

topologies towards local processing units, rather than towards networks with high connectivity. This excludes many – in principle also potentially successful – topologies. But as a heuristic, large, highly connected networks tend to be unable to evolve complex local processing sub-networks, because synapses from arbitrary sources influence most neuron clusters in a disturbing way. This problem increases as the number of neurons in the network gets larger, because the probability for a synapse to be unrelated, and therefore potentially disruptive, increases with every neuron. Therefore we expect highly connected networks to have a lower probability to provide interesting, non-trivial solutions [1]. Therefore neuro-modules can be used to promote plausible connections based on domain knowledge, such as grouping motors and sensors of the same joint together. Neuro-modules also allow the definition of *synaptic pathways*, i.e. to prevent or permit connections between modules explicitly.

A powerful constraint on the structure of neuron groups is the definition of *symmetries* and *repetitive structures*. Depending, of course, on the targeted behavior problem, many evolutions can be greatly restricted when the desired network is assumed to be symmetric. Examples are walking or squatting of a humanoid robot or the repetitive structure of a multi-legged walking machine. Symmetries remove large parts of the

search space, because the parameters of all symmetrized neurons and synapses are not part of the search space any more (e.g. the entire right side in Fig. 1).

An additional positive effect on the structure using modularization is the better readability of the resulting networks (see Fig. 1). Functional elements can be isolated more easily and signal paths can be better traced, because most synapses are locally connected and have less dependencies to other parts of the network.

3.2 Functional Constraints

Neuro-modules bias evolution to evolve local processing units, that are often related to local functions. Although, admittedly, it can not be guaranteed that the evolved structure of a module implements a single, well defined function, the tendency still is towards functions distributed over only a few, local modules. This still simplifies the isolation of such functions when an evolved network is analyzed.

Neuro-modules can also be used to represent predefined functional units, that may origin from previous evolutions or analytic reasoning. Once a functional processing unit is found by evolution, it can be reused in future evolutions as neural building block. Forcing evolution to reinvent already known processing units in each evolution from scratch only blows up the search space without any gain from the scientific perspective. With an additional *neuro-module insertion* mutation operator that can insert predefined neural building blocks from a library, larger, functionally more complex networks can evolve in shorter time.

Neural building-blocks can also be constrained with very *specific* constraint functions. Because building blocks can be constructed by hand – although often based on evolved structures – specialized constraint functions can be added. Such functions can be used to ensure, for instance, that the function or complex structure of a module is preserved independently of the mutations taking place. They can also be used to design complex modules, such as neural fields [3], memory units [15], oscillators [12], structures with adaptive synapses and the like.

Constraints can also be used to *clone* a mutable neuro-module and to reuse the same network structure in multiple places of the network. In this way the function of this module can still evolve, while it is used with all modifications in several places, profiting from enhancements immediately. A common usage of this is the definition of sensor filters or motor controllers (as in Fig. 1), where the same structure is required for any sensor or motor of the same type. If the sensor or motor works similar in all places, then the controller has not to be optimized multiple times.

3.3 Evolutionary Constraints

Neuro-modules are a suitable target for modification operators during evolution. Because neuro-modules are well structured, providing a specific interface to their surrounding network parts, they can be exchanged and replaced with only little impact on the rest of the network. This enables the usage of *modular crossover*. Crossover in most neural network implementations is highly destructive due to the potentially large structural differences between parents. Crossover between such unrelated networks most probably produces networks that are less fit than both of their parents, so that most of

these networks usually do not survive. Modular crossover is less affected by this problem, because crossover takes place only at well defined network parts, namely at the module level. Modules are only replaced by compatible modules, which means that their interfaces match and the module types are similar.

In addition to module exchange between parents, modules may also be exchanged by compatible modules from a library of predefined building blocks or by neuro-modules co-evolving with the behavior controllers in their own populations.

A particular benefit of constrained modularization for evolution is that the approach to solve a given behavior problem can be biased to a large extend in advance. This way the experimenter does not only specify the problem to be solved, but also influences to a high degree, how the problem is going to be solved. Also, the iteration of evolutions becomes much easier: The behavior problem may be solved first by applying sharp restrictions on the evolving networks. Then, iteratively, the network can be opened for new solution approaches to stepwise enhance the behavior.

4 Application

The modularization approach with the described features has been implemented in the ICONE (Interactively Constrained Neuro-Evolution) method. Currently the implementation supports structure evolution based on neuron, synapse and neuro-module insertions. Explicit specifications of neural pathways between neuro-modules are considered, as well as connection restrictions induced by the hierarchical interfaces of neuro-modules. Neuron groups and neuro-modules can be restricted with arbitrary, user defined constraint functions, such as symmetry, cloning and restrictions of neuron and synapse structures. A library of neuro-modules as basic building blocks is under continuous construction, including neuro-modules for different kinds of oscillations, memory, joint controllers, sensor filters, event detections, context switches and behavior interpolation. During evolution all aspects of the evolutionary algorithm can be modified on-line to guide evolution through the search space.

Manually modularizing large networks is not trivial. Therefore a graphical neural network editor was implemented that supports the visual manipulation of all mentioned aspects of the neural networks. Without such an editor, modularization is difficult to apply.

5 Examples

The modularization technique has been applied to develop neuro-controllers for several complex robots. These robots include for instance the six legged walking machine *Octavio* with 24 sensor and 18 motor neurons, and the *A-Series* humanoid robot with 37 sensor and 42 motor neurons. The developed behaviors include – among others – different kinds of walking, squatting and stabilized standing.

Some examples are shown in Fig. 2 and Fig. 3. Due to space limitations, details on the evolved behaviors will be presented in upcoming publications. But it can be said, that with pure structure evolution solutions for these kind of problems could not be found at all.

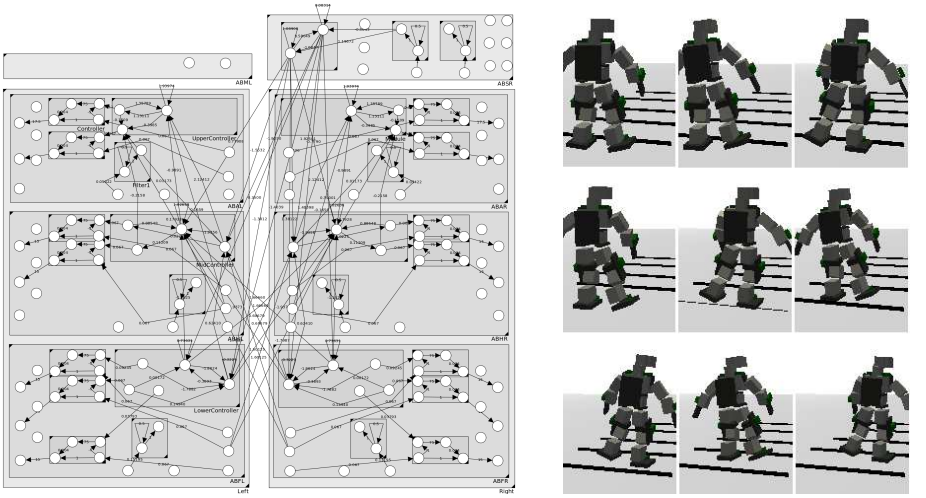


Fig. 2. Network and time-series of an evolved neuro-controller for walking with the *A-Series* humanoid, based on a constrained initial network similar to Fig. 1. The initial network was constrained to search for solutions based on the acceleration sensors of the shoulder (*Mid of Upper Right Module*). Substantially different networks can be produced starting with internal pattern generators as in Fig. 1. [Evolution: $\approx$ 400 generations with 150 individuals].

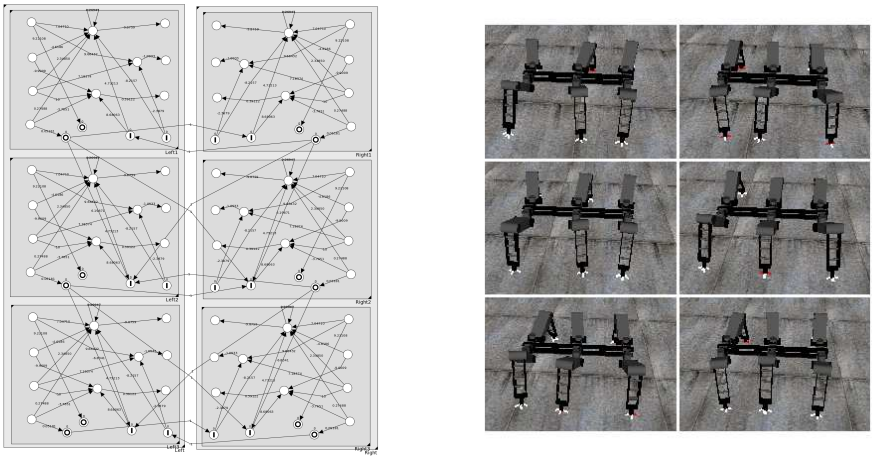


Fig. 3. Network and time-series of an evolved neuro-controller for walking with the 6-legged walking machine *Octavio*. Here only one leg controller (*Upper Left*) is evolved, all other legs clone this prototype controller. In addition the network has a right / left symmetry. The focus of this experiment is to find a universal leg controller for a multi-legged walking machine and appropriate interconnection patterns. With minor adjustments of the constraints the focus of the experiment can be changed, e.g. to find specialized leg controllers for front, mid and hind legs. [Evolution: $\approx$ 300 generations with 100 individuals].

6 Discussion

Modularizing a network by applying domain knowledge and user experience obviously restricts the search space for neuro-evolution algorithms. Attempts to restrict the search space have been conducted by many authors, because without restrictions the evolution of large network topologies of non-trivial robots becomes infeasible.

A common approach is the use of specific genome representations, that imply for instance a fully symmetric network [7], [13]. This approach can be compared to modularization with symmetry constraints. But because the symmetry is embedded directly in the genome representation or in the genotype-phenotype mapping, a new genome type has to be created for each new experimental scenario. Also complex, stacked symmetries are difficult to set up. During evolution such genomes are rigid and can not be extended if needed. Using, as proposed here, constraint functions to influence the relations of network parts, symmetry can be implemented as a simple extension of the network genome and can easily be removed or changed without changing the genome type. Furthermore constraints are not restricted to symmetry, but can enforce any kind of structural dependency, like complex spatial connections as used in neural fields [3].

Another approach to reduce the search space is to focus only on specific parts of the target robot. For instance, walking may be evolved with only the legs of a humanoid robot, replacing the entire upper body by a simple block of comparable mass [13]. This, indeed, reduces the search space, because all motors and sensors of the simplified body parts have been removed. Though, extending such a controller to the full robot becomes difficult, because the evolved controllers will ignore the influence of the other moving body parts. Also, for each new approach, a new simulated robot has to be designed, that focuses on the desired motor and sensor aspects, and therefore also iterative evolution in small steps becomes more difficult. With the proposed modularization technique, the complex target robot can be used right from the beginning. Unwanted sensors and motors can be excluded in the beginning by synaptic pathway restrictions and can be re-enabled at any time during the evolution. Therefore the evolution can start with a minimal subset of the robot's actuators and sensors, but can still include the non-essential robot parts to further optimize the controllers.

A third popular search space restriction approach is structure reuse. In most cases structure reuse is implemented within developmental evolution systems, like Cellular Encoding [5], where the genotype is mapped to a phenotype by applying a sequence of construction rules. These algorithms have been shown to reuse structures in multiple places while evolving the *structure blue-prints* only once. A disadvantage of such developmental approaches is, that the resulting modular structures are difficult to isolate for later usage. Furthermore, it is difficult to translate a complex starting network with this kind of modularity into its genotype representation, and to monitor and manipulate these modular structures during evolution. Neuro-modules as building blocks on the other hand do not require a complex mapping from genotype to phenotype and thus can be reused as entire structure with little effort. Other techniques [2], such as Modular NEAT [14], try to automatically define neuro-modules as building-blocks without a complex genotype-phenotype mapping. But also here, functional modules have to be reinvented in every evolution, because predefined modules can not be used. Furthermore, the reused sub-networks are arbitrarily aligned to the input and output neurons

without domain knowledge, so that – especially in large networks – proper use of the modules becomes unlikely again.

Constrained modularization allows the utilization of all mentioned search space restriction methods in a uniform, extensible framework. With appropriate libraries of functional neuro-modules new types of larger control networks can be developed, that might give deeper and even new insights into neural organization of behavior. Constrained modularization as a general principle does not really restrict experimenters in their approaches, because new approaches can be simply implemented by introducing new constraint functions.

7 Conclusions

In this contribution it was discussed how *constrained modularization* of large neural networks for robot control can significantly reduce the search space for neuro-evolution processes. Large neural networks can be spatially and logically partitioned by *neuron groups* and *neuro-modules*. Both types of grouping can be the target of *constraint functions*, that force the compliance of – partly very specific – constraints, such as network symmetries, dependencies, module cloning and connectivity structures between or within modules. The modularization is done manually to apply domain knowledge and to bias the search towards desired solution approaches. In this way the search space is restricted by the user to a well defined potential solution space, which increases the chance to find appropriate solutions. For modular neural networks new types of evolution operators are defined: *modular crossover* and *neuro-module insertions*. Modular crossover allows the exchange of sub-networks in a minimally destructive way. Insertions of functional neuro-modules as mutation allow the extension of a network with already working functional sub-networks, which eases the transfer of findings from previous evolutions and relieves evolution from reinventing already known structures. The described approach has been used to develop different behaviors for several robots with many sensors and actuators, including a multi-legged walking machine and humanoid robots. Detailed results are described in upcoming publications.

Acknowledgements

The authors thank Arndt von Twickel and Manfred Hild for many inspiring discussions. Thanks also to Verena Thomas and Ferry Bachmann for their contributions to the simulation environment. This work was partly funded by EU-Project Number ICT 214856 (ALEAR Artificial Language Evolution on Autonomous Robots. http://www.alear.eu).

References

1. Beer, R.D. (2010). Fitness Space Structure of a Neuromechanical System. To appear in Adaptive Behaviour.
2. Calabretta, R., Nolfi, S., Parisi, D., and Wagner, G. P. (2000). Duplication of Modules Facilitates the Evolution of Functional Specialization. Artificial Life, 6(1), pp. 69–84.
3. Coombes, S. (2005). Waves, Bumps, and Patterns in Neural Field Theories. Biological Cybernetics, 93(2), pp. 91–1008.
4. Floreano, D., Husbands, P., and Nolfi, S. (2008). Evolutionary Robotics. In Siciliano, B. and Khatib, O., editors, Springer Handbook of Robotics, pp. 1423–1451. Springer.
5. Gruau, F. (1995). Automatic Definition of Modular Neural Networks. Adaptive Behaviour, 3(2), pp. 151–183.
6. Harvey, I., Paolo, E., Wood, R., Quinn, M., and Tuci, E. (2005). Evolutionary Robotics: A New Scientific Tool for Studying Cognition. Artificial Life, 11(1-2), pp. 79–98.
7. Hein, D., Hild, M., and Berger, R. (2007). Evolution of Biped Walking Using Neural Oscillators and Physical Simulation. Proceedings of the RoboCup 2007: Robot Soccer World Cup XI.
8. Hülse, M., Wischmann, S., Manoonpong, P., von Twickel, A., and Pasemann, F. (2007). Dynamical Systems in the Sensorimotor Loop: On the Interrelation Between Internal and External Mechanisms of Evolved Robot Behavior. In Lungarella, M., Iida, F., Bongard, J. C., and Pfeifer, R., editors, 50 Years of Artificial Intelligence, volume 4850 of Lecture Notes in Computer Science, pp. 186–195. Springer.
9. Hülse, M., Wischmann, S., and Pasemann, F. (2004). Structure and function of evolved neuro-controllers for autonomous robots. Connection Science, 16(4), pp. 249–266.
10. Nolfi, S. and Floreano, D. (2004). *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*, ISBN 978-0262640565, Bradford Book.
11. Pasemann, F. (1995). Neuromodules: A Dynamical Systems Approach to Brain Modelling. In Herrmann, H. J., Wolf, D. E., and Poppel, E., editors, *Workshop on Supercomputing in Brain Research: from tomography to neural networks, KFA Julich, Germany*, World Scientific Publishing Co.
12. Pasemann, F., Hild, M., and Zahedi, K. (2003). SO(2)-Networks as Neural Oscillators. Computational Methods in Neural Modeling, 2686/2003, pp. 144–151.
13. Reil, T. and Husbands, P. (2002). Evolution of Central Pattern Generators for Bipedal Walking in a Real-Time Physics Environment. *IEEE Transactions on Evolutionary Computation*, Vol.6(2), pp. 159–168.
14. Reisinger, J., Stanley, K. O., and Miikkulainen, R. (2004). Evolving Reusable Neural Modules. In Deb, K., et al., editors, Genetic and Evolutionary Computation – GECCO-2004, Part II, volume 3103 of *Lecture Notes in Computer Science*, pp 69–81, Springer.
15. Rempis, C. W. (2007). Short-Term Memory Structures in Additive Recurrent Neural Networks. Master's thesis, University of Applied Sciences Bonn-Rhein-Sieg, Germany.

Automatic Modularization of Artificial Neural Networks

Eva Volna

University of Ostrava, 30ht dubna st. 22, 701 03 Ostrava, Czech Republic
eva.volna@osu.cz

Abstract. The majority of this paper relies on some forms of automatic decomposition tasks into modules. Both described methods execute automatic neural network modularization. Modules in neural networks emerge; we do not build them straightforward by penalizing interference between modules. The concept of emergence takes an important role in the study of the design of neural networks. In the paper, we study an emergence of modular connectionist architecture of neural networks, in which networks composing the architecture compete to learn the training patterns directly from the interaction of reproduction with the task environment. Network architectures emerge from an initial set of randomly connected networks. In this way can be eliminated connections so as to dedicate different portions of the system to learn different tasks. Mentioned methods were demonstrated for experimental task solving.

1 Reasons for a Modular Approach

The primary reason for adopting an ensemble approach to combining nets into a modular architecture is that of improving performance. There are a number of possible justifications for taking a modular approach to combining artificial neural nets. First, a modular approach might be used to solve a problem which could not have been solved through the use of a unitary net. A modular system of nets can exploit the specialist capabilities of the modules, and consequently achieve results, which would not be possible in a single net. Another reason for adopting a modular approach is that of reducing model complexity, and making the overall system easier to understand. This justification is often common to engineering design in general. Other possible reasons include the incorporation of prior knowledge, which usually takes the form of suggesting an appropriate decomposition of the global task. A modular approach can also reduce training times and make subsequent modification and extension easier. Finally, a modular approach is likely to be adopted when there is concern to achieve some degree of neurobiological or psychological plausibility, since it is reasonable to suppose that most aspects of information processing involve modularity.

A modular neural network can be characterized by a series of independent neural networks moderated by some intermediary. Each independent neural network serves as a module and operates on separate inputs to accomplish some subtask of the task the network hopes to perform [1]. The intermediary takes the outputs of each module and processes them to produce the output of the network as a whole. The interme-

diary only accepts the modules' outputs. As well, the modules do not interact with each other.

When a modular approach is adopted, for what ever reason, there are different ways in which a problem might be decomposed. In particular, task decomposition can be either explicit or automatic. Explicit decomposition is likely to depend on an understanding of the task and the capabilities of the modular components. It provides a way of incorporating prior knowledge and understanding of the task in question. For instance, a particular decomposition might be implied by the structure of the task, if for example, the data came from different sources or took different forms [3]. Similarly, modular decomposition might be guided by theories or evidence about the likely modular structures in the human brain, or the human information processing system. By contrast, automatic decomposition, where decomposition is accomplished through the blind application of a data partitioning algorithm, is particularly useful when expert knowledge of the task is not available.

There has been a considerable amount of research on automatic decomposition methods, for example, the mixture-of-experts [4] and hierarchical mixtures-of-experts approaches [6]. Under such methods, the input data is partitioned into several subspaces, and simple systems are trained to fit the local data. Such data partitioning is often more effective than training on the whole input data space. In general, the concern in this work is to improve performance, and as such it is closely related to the ensemble approach. Thus performance on a task could be improved by either taking a modular decompositional approach, or by creating an ensemble of parallel solutions to the problem, and combining them in some way. As yet, it is not clear where one approach is likely to be better than the other [7]. It is increasingly recognized that the effectiveness of ensemble approaches depends on the extent to which their failures are correlated and a decompositional approach promotes the reduction of such correlation. However, there are few direct comparisons of the relative effectiveness of a modular approach relying on automatic decomposition, and an ensemble-based approach. Neither are the two alternatives necessarily mutually exclusive, since it is possible to envisage an ensemble system, where each member was composed of a set of modules created through automatic decomposition. The majority of this paper relies on some forms of automatic decomposition tasks into modules. In this way can be eliminated connections so as to dedicate different portions of the system to learn different tasks.

2 Automatic Task Decomposition

An artificial neural network may show slow learning because it is being trained to simultaneously perform two or more tasks. For example, suppose that the mapping from the input units to each output unit constitute separate tasks and that the network is trained via backpropagation algorithm. During training, each output unit provides error information to the hidden units from which it receives a projection. It is possible that the error information from one output unit may indicate that a hidden unit's activation should be lager and, at the same time, the error information from another output unit may indicate that the same unit's activation should be smaller. This conflict in the error information is called spatial crosstalk. Although spatial crosstalk is clearly

seen in terms of the backpropagation algorithm, it is limited to networks trained using this algorithm. Therefore, spatial crosstalk may be considered as resulting from the connectivity of the network and not from the learning algorithm used to training the network. By maintaining short connections and eliminating long connections, spatial crosstalk can be reduced and tasks can be decomposed into subtasks. Although the three systems show in Fig. 1 [5] can be trained to perform the same mapping. System in Panel A has its hidden units fully interconnected with its output units and is most susceptible to spatial crosstalk. System in the Panel B has its hidden units on the top fully interconnected with its top output units and its hidden units on the bottom fully interconnected with its bottom output units. Thus, it consists of two separate networks (two 4-4-2 networks). If the mapping that this system is trained to perform can be decomposed so that the mapping from the input units to the top set of output units may be thought of as one task and the mapping from the input units to the bottom set of output units may be thought of as a second task, then this system has dedicated different networks to learn the different tasks. Because there is no spatial crosstalk between the two tasks, such a system may show rapid learning. The Panel C has hidden unit project to only a single output unit. It therefore consists of a separate network for each output unit (four 4-2-1 networks) and is immune to spatial crosstalk.

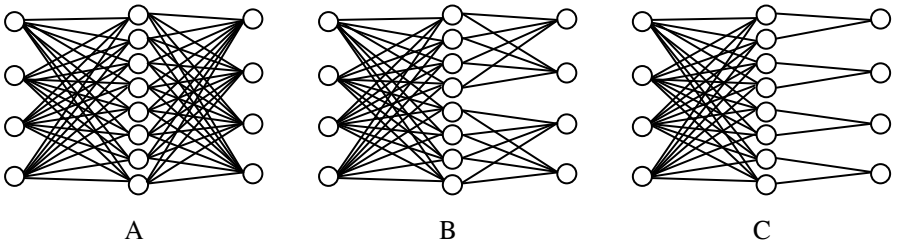


Fig. 1. A: One 4-8-4 network. B: Two 4-4-2 networks. C: Four 4-2-1 networks [5].

Artificial neural network with many adjustable weights may learn to training data quickly and accurately, but generalize poorly to novel data. One method of improving the generalization abilities of network with too many “degrees of freedom” is to decay or eliminate weights during training. A second method is to match the structure of the network with the structure task. For example, networks, whose units have local receptive fields, can learn to reliably, detect the local structure that is often present in pattern recognition tasks. A system that maintains short connections and eliminates long connections should generalize well because its degrees of freedom are reduced and because its units develop local receptive fields.

Artificial neural network often develop relatively not interpretable representations for at least two reasons. Networks whose units are densely connected tend to develop representations that are distributed over many units and, thus, are difficult to interpret. In addition, not interpretable representations often develop in networks that are trained to simultaneously perform multiple tasks. In contrast, networks, whose units tend to have local receptive fields, towards short connections may develop relatively local representations. Furthermore, such a system may be capable of eliminating connections so that different networks learn different tasks.

3 Evolutionary Module Acquisition

There is a simple model of evolutionary emergence of modular neural network topology introduced in the chapter [10]. We describe a method of optimization of the modular neural network architecture via evolutionary algorithms that uses a fix part of network architecture in the genome. Every individual is a multilayer neural network with one hidden layer of units. We have to fix its maximal architecture (e.g. number of input, hidden and output units) before the main calculation. Population P consists of $P = \{\alpha_1, \alpha_2, ..., \alpha_p\}$, where p is equal to a number of chromosomes in P. Every chromosome consists of binary digits that are generated randomly with a probability 0.5. Chromosome, with m hidden units a n output units is shown in Fig. 2, where $e_{ij} = 0$, if the connection between i - *th* hidden unit and j - *th* output unit of the individual doesn't exist, and $e_{ij} = 1$, if the connection exists ($i = 1,\ldots,m$; $j = 1,\ldots n$). Connections between input and hidden units are not included in chromosomes, because they are not necessary for modular network architecture creation. Each individual (e.g. the network architecture) is partially adapted by backpropagation, its fitness function is then calculated as follows (1):

$$Fitness_k = \frac{1}{E_k} \tag{1}$$

where $k = 1, \ldots, p$ (p = number of individuals in the population);
E_k is the error after backpropagation adaptation of the k-individual.

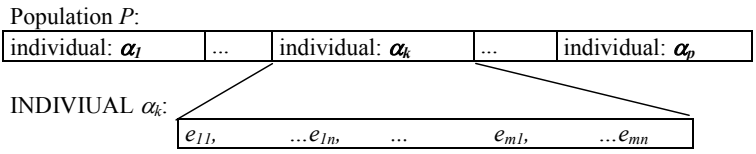


Fig. 2. A population of individuals.

Only two mutation operators are used, no crossover operators. The *first* mutation operator is defined in following way. In the every generation, one individual is randomly chosen and each bit is changed with probability 0.01 (e.g. if the connection exists – after mutation it does not exist and vice versa) in its chromosome. The *second* mutation operator is defined in following way, see Fig. 3. First, we define a pattern of t-consecutive zeroes that will be fixed during whole calculation. The pattern is determined by number of neurons in the output layer, which represent individual modules. Output neurons are organized into d modules, $t = min\ (t_i,\ i = 1\ ,..., d)$, where t is number of neurons in the pattern, and t_i is number of neurons in the i-th module. Defined pattern is represented as a continuous chain of t-zeros, which is not changed during applications of the second mutation operator. Fixation of t-zeros chain can be defended by biological motivation, where the protection against mutation is usually related to continuous section. Defined pattern in the chromosome allows temporary fixing the existing module against the application of the second mutation operator. Then we find the define pattern in each chromosome. If we find only

one continuous pattern, we fix it. If we find more than *n*-consecutive zeroes, we randomly choose *n*-consecutive zeroes from them and fix them. The fixed pattern represents a single atomic unit and the second mutation operator is not applied to it. Only to the rest of bits from chromosomes are changed with probability 0.01. Thus, each individual has a unique collection of fixed patterns. The second mutation operator is applied to every individual r - times, where r is a parameter and its value is define before calculation. Only the best individual or its best mutation is included into the next generation. Next, all individuals in the new generation release a portion of the pattern that was fixed that way they can once again be manipulated by reproduction operators. The process of evolutionary algorithms is ended when the population achieves the maximal generation or if there is no improvement in the objective function for a define sequence of consecutive generations.

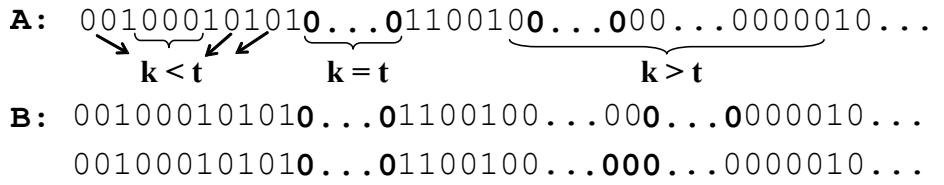


Fig. 3. The second mutation operator. The fixed pattern is *t*- consecutive zeroes, *k* is number of consecutive zeroes in the chromosome. A: An individual before mutation. B: Possible chromosomal representation of the individual after mutation.

4 Modularization Via Evolutionary Hill – climbing Algorithm

The second presented method is based on hill-climbing algorithm with learning [8]. Evolution of the probability vector is modeled by a genetic algorithm on the basis of the best evaluated individuals in this algorithm, which are selected on the basis of the speed and quality of learning of the given tasks [11]. Population P is presented in Fig. 2 and is defined in the same way as in the previous chapter. Individuals in the next generation are generated from the updating probability vector. Every individual (e.g. its neural network architecture) is partially adapted by backpropagation [2] and evaluated by the quality of its adaptation. The number of epochs is a very important criterion in the described method, because modular architectures start to learn faster than fully connected multilayer connectionist networks [9]. Our goal is to produce such a neural network architecture that is able to learn a given problem with the smallest error. A backpropagation error is a fitness function parameter. A fitness function value F_i of the i - th individual is calculated as follows (2):

$$F_i = \frac{\sum_{k=1}^{con} f_{ik}}{con} \tag{2}$$

where $i = 1, \ldots, p$ (p is number of individuals in a population);

$f_{ik} = \frac{1}{E_{ik}}$ is a fitness function value of the i-th individual in the k-th adaptation;

$k = 1, \ldots, con$ (con is a define constant, $con>1$);

E_{ik} is the backpropagation error of the i-th individual in the k-th adaptation.

Crossover and mutation operators are not used in the described method. This algorithm is based on the probability vector emergency. The probability vector is updated on the basis of well - evaluated individuals in the population. Entries $0 \le w_{ij} \le 1$ of the probability vector $\boldsymbol{w} = (w_{11},\ldots, w_{1n},\ldots, w_{m1},\ldots, w_{mn}) \in [0,1]^{mn}$, ($m$ is number of hidden units; n is number of output units) determine probabilities of appearance of '*1*' entries in given positions.

Entries of the Probability Vector are Calculated in the Next Generation as follows:

– We calculate F_{avg}, e.g the average fitness value of the population in the given generation (3):

$$F_{avg} = \frac{\sum_{i=1}^{p} F_i}{p}, \tag{3}$$

where p is a number of individual in the population;
F_i is a fitness function value of the i-th individual, see a formula (2).

– We choose a set of q individuals with $F_i \ge F_{avg}$, e.g. $\alpha_1, \alpha_2, \ldots, \alpha_q$ ($1 \le q \le p$, where p is a number of individuals in the population.

– Entries of the probability vector of the population $w'_k \in [0,1]$ are calculated as follows (4):

$$w'_k = (1-\lambda) w_k + \lambda w''_k \tag{4}$$

where $k = 1, \ldots, mn$ (mn is a number of the probability vector $\boldsymbol{w}$ entries);
w_k is a value of the k-th entry of the probability vector in the last generation;
λ is a constant ($0 < \lambda < 1$);
w''_k is a value of the k-th bit of the probability vector $\boldsymbol{w}$ that is calculated as follows (5):

$$w''_k = \frac{\sum_{i=1}^{q} (e_k)_i}{q} \tag{5}$$

where $(e_k)_i$ is a value of the k-th bit of the chromosome of the individual α_i ($i = 1, \ldots, q$) and it is true $F_i \ge F_{avg}$ for these individuals.

The best individual in the population is included to the next population automatically. Values of the chromosomes of the rest of individuals α_i ($i = 2, \ldots, p$) are calculated for the next generation as follows: if $w_k = 0(1)$, then $(e_k)_i = 0(1)$; if $0 < w_k < 1$ the corresponding $(e_k)_i$ is determined randomly by (6):

$$(e_k)_i = \begin{cases} 1 & \text{if } random < w_k \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where $k = 1, \ldots, p$ (p = number of individuals in the population).

The process of the evolutionary algorithm is ended if the saturation parameter $\tau(\boldsymbol{w})^*$ is greater than a predefined value.

5 Experiments

In the experimental task, a system (neural network) recognizes a binary pattern and its rotation. Neural network with one hidden layer of units with topology 9-13-8 adapted by backpropagation represents a system here. The creation of such modular system that would solve partial tasks correctly was our target. Basic set of training patterns are organized into a matrix (grid) 3x3, which is represented by binary vector. The direction of rotation is defined towards the basic pattern by four possibilities: (a) 0°-state without rotation, (b) turn 90°, (c) turn 180°, and (d) turn 270°. The training set includes four patterns that are defined in four different states, see Fig 4. Thus, we get 16 different combinations of shapes and their rotations. Eight output units are divided into two subsets of four units. Units in the "shape" subset are responsible for indicating the identity of the input. Each input is associated with one of the four "shape" units, and one of the four rotations. The system is considered to correctly recognize and locate an input.

Parameters of the Experimental part.

- Population (both methods):
  Number of individuals: 100.
  Neural network architecture: 9 – 13 – 8.
  Training algorithm: Backpropagation
  (learning rate: 1; momentum: 0; training times: 150 epochs in the partial training).
- Parameters of method from chapter 3:
  Probability of mutations: 0.01.
  Fix pattern in the second mutation: "0000".
  r: 5.
  Ending conditions: Maximal number of generations: 500.

* $\tau(\boldsymbol{w})$ = a number of entries (w_i) of the probability vector $\boldsymbol{w}$ that are less then w_{eff} or (1- w_{eff}), where w_{eff} is a small positive number.

– Parameters of method from chapter 4:
  con: 100; see formula (2).
  λ: 0.2; see formula (4).
  Ending conditions: The saturation parameter, $\tau(\boldsymbol{w})$: 0.99**m***n*
  (*m*=13, number of hidden units; *n*=8, number of output units); w_{eff} = 0.01.

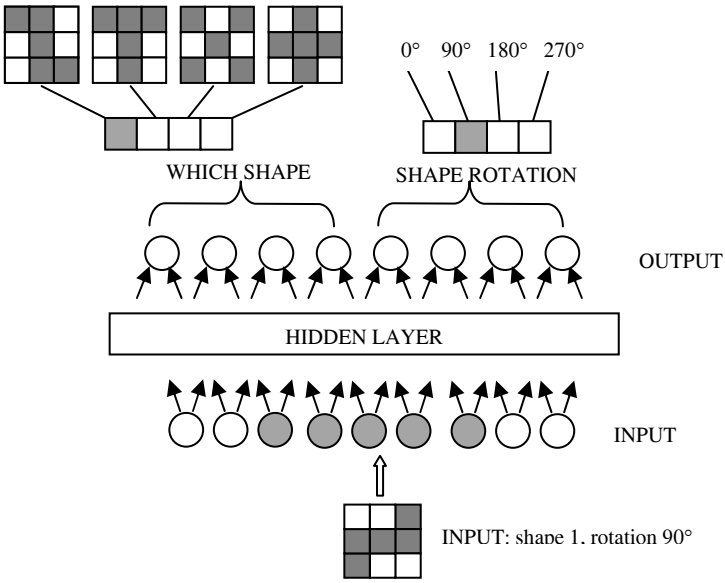


Fig. 4. A defined pattern in a training set.

Table 1 shows a table of results. The table shows an evolution of the best individual in the population. It is evidently seen, the connections among modules are eliminated faster than connection inside modules. These results support also the fact that systems were created dynamically during a learning process. Method from chapter 3 gives the following results: six hidden units of the best individual realise the "shape" task and its four units realise the "rotation" task in the last generation. Method from chapter 4 gives the following results: seven hidden units of the best individual realise the "shape" task and its four units realise the "rotation" task in the last generation. Calculation was terminated, when ending conditions were fulfilled, e.g. for method from chapter 3 was calculation terminated in the 498-th generation and for method from chapter 4 was calculation terminated in the 353-rd generation. Other numerical simulations give very similar results.

Table 1. Table of results.

	Method from chapter 3:			Method from chapter 4:		
generation	number of hidden units: „*shape*“ task:	number of hidden units: „*rotate*“ task:	number of interferences:	number of hidden units: „*shape*“ task:	number of hidden units: „*rotate*“ task:	number of interferences:
1	1	1	11	0	0	13
100	2	3	8	1	3	9
200	3	3	7	4	3	6
300	4	3	6	6	3	4
400	5	4	4	*GENERATION*: 353		
	GENERATION: 498			7	4	2
	6	4	3			

We made the following experiment. Neural network with modular architecture (the best individual) and network with the same arrangement of neurons, but by all connections between layers have been adapted via backpropagation to solve the above defined task. For each model was done 10 adaptations, the weight vector was at the beginning of each simulation generated randomly. In Fig. 5 the average error function values is shown: (a) modular neural network and (b) fully connected neural network during the whole calculation. Adaptation of each neural network was terminated after 1500 iterations. The figure shows that the network with a modular architecture, which includes only a limited number of connections, allows to learn the considered problem as efficiently as a monolithic networks designed within an appropriate architecture.

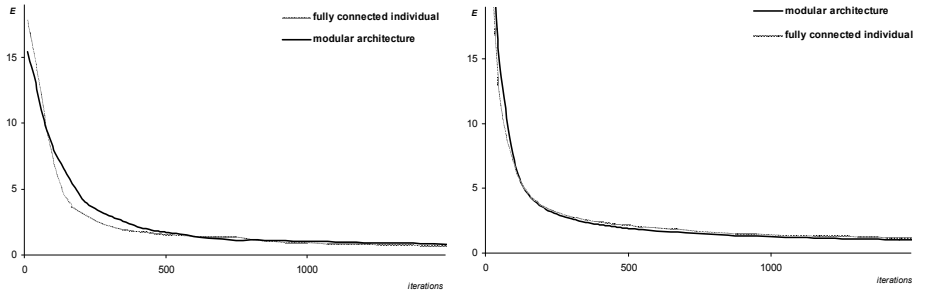


Fig. 5. The history of average error function value during whole calculation A: method from chapter 3; B: method from chapter 4.

6 Conclusions

Both described method are methods of automatic neural net modularization. The problem specific modularisations of the representation emerge through the iteration of the evolutionary algorithm directly with the problem.

When interpreting solutions, we have to be careful, because algorithms' parameters are not the object of the optimization process, but we obtain solutions just in dependence on these parameters. Both numerical simulations reflect the modular

structure significance as a tool of a negative influence interference rejection on neural network adaptation. As the hidden units in the not split network are perceived as some input information processing for output units, where a multiple pattern classification is realized on the basis of diametrically distinct criteria (e.g. neural network has to classify patterns according to their form, location, colors, ...), so in the beginning of an adaptation process the interference can be the reason that output units also get further information about general object classifications than the one which is desired from them. This negative interference influence on running the adaptive process is removed just at the modular neural network architecture, which is proved also by results of the performed experiment. The winning modular network architecture was the product of emergence using evolutional algorithms. The neural network serves here as a special way of solving the evolutional algorithm, because of its structure and properties it can be slightly transformed into an individual in evolutionary algorithm.

References

1. Di Fernando, A., Calebretta, R., and Parisi, D. (2001) Evolving modular architectures for neural networks. In French R., and Sougne, J. (eds.).Proceedings of the Sixth Neural Computation and Psychology Workshop: Evolution, Learning and Development. Springer Verlag, London.
2. Fausett, L. V. (1994) Fundamentals of neural networks. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.
3. Hampshire, J. and Waibel, A. The Meta-Pi network: Building distributed knowledge representation for robust pattern recognition. Technical Report CMU-CS-89-166. Pittsburgh, PA: Carnegie Mellon University.
4. Jacobs, R. A., Jordan, M. I., Nowlan, S.J., and Hinton, G. E. (1991) Adaptive mixtures of local experts. Neural Computation, 3, pp.79-97.
5. Jacobs, R. A., Jordan, M. I. (1992). Computational consequences of a bias toward short connections. Journal of Cognitive Neuroscience, 4, 323–336.
6. Jacobs, R. A. (1994) Hierarchical mixtures of experts and the EM algorithm. Neural Computation, 6, 181-214.
7. Jordan, M. I. and Jacobs, R. A. (1995) Modular and Hierarchical Learning Systems. In M. A. Arbib (Ed) The Handbook of Brain Theory and Neural Networks. pp 579-581.
8. Kvasnička, V; Pelikán, M.; Pospíchal, J. (1996) Hill climbing with learning (an abstraction of genetic algorithm). Neural network world 5, 773-796.
9. Rueckl, J. G. (1989) Why are “What” and “Where” processed by separate cortical visual systems? A computational investigation. Journal of Cognitive Neuroscience 2, 171-186.
10. Volna, E. (2002) Neural structure as a modular developmental system. In P. Sinčák, J. Vaščák, V. Kvasnička, J. Pospíchal (eds.): Intelligent technologies – theory and applications. IOS Press, Amsterdam, pp.55-60.
11. Volna, E. (2007) Designing Modular Artificial Neural Network through Evolution. In J. Marques de Sá, L. A. Alexandre, W. Duch, and D. P.Mandic (eds.) Artificial Neural Networks – ICANN’07, Lecture Notes in Computer Science, vol. 4668, Springer-Verlag series, pp 299-308.

GMPath - A Path Language for Navigation, Information Query and Modification of Data Graphs

Karsten Wendt, Matthias Ehrlich and René Schüffny

Technische Universität Dresden, Institute of Circuits and Systems
01062 Dresden, Germany
{wendt, ehrlich, schueffn}@iee.et.tu-dresden.de

Abstract. This paper presents a newly developed path language named GMPath intended to ease the navigation, information query and modification of general, directed model graphs for the FACETS Stage 2 Large Scale Reconfigurable Neural Hardware Simulator. Furthermore it introduces the reader to the relevant aspects of the FACETS system and its software framework accordingly.

1 Introduction

The project *Fast Analog Computing with Emergent Transient States – FACETS* [1] aims at the exploring of various computational aspects of biological neural networks. This encompasses the development of a novel neural hardware system as a joined effort of research groups of the *Ruprecht-Karls-Universität Heidelberg* and the *Technische Universität Dresden*. [2], [3]

The complexity of such a hardware system also requires an adequate software system, to configure, control and validate results. A graph based model description was developed, representing all aspects of the system. To interact with and query the models, the description is extended by a query interface.

This paper is split into three sections. The first section describes the current state of the FACETS systems, focusing modelling and mapping issues. Embedded in the FACETS software framework, the graph model itself will be introduced. The next section characterizes the motivation and requirements for the development of the path language GMPath and distinguishes it from existing query languages. Afterwards, basing on a meta description the grammar of the language is defined and the semantics of its elements are explained. Illustrating the use of GMPath the last section provides several examples, based on a given concrete graph model by explaining some queries in more detail.

2 Current State of the FACETS Systems from the Mapping Viewpoint

The systems involved in the FACETS mapping and configuration process can be separated into the hardware model on the one hand and the biological model on the other hand, which should be simulated on the hardware system after the configuration. To

encompass the modelling and mapping problem a so called *graph model* is used and integrated in the FACETS software framework. The models and the software framework are characterized in the following sections.

2.1 Biological Model

The biological systems, intended to be mapped to the FACETS hardware, can be considered as networks of neurons and synapses. A neuron is connected to a number of synapses and characterized by a set of parameters. A synapse connects a source and a target neuron and is as well assigned to parameters. The illustration of an example model is shown in figure 1.

2.2 Hardware Model

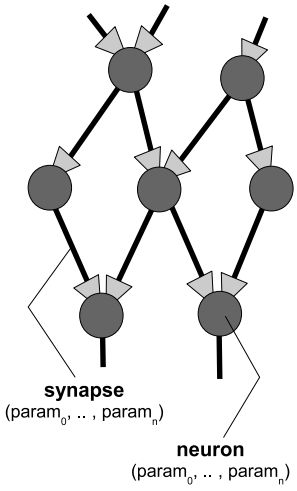


Fig. 1. Example of a biological system with 7 neurons and 11 synapses.

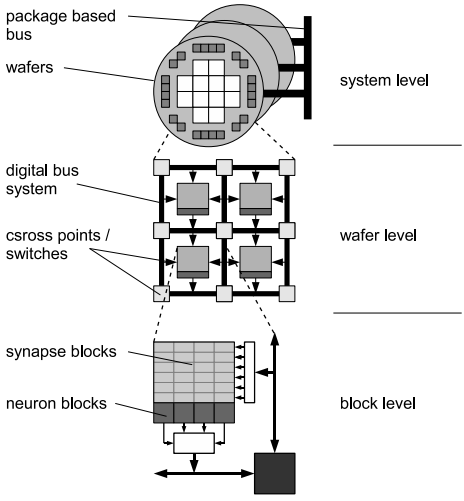


Fig. 2. Hierarchical view of the FACETS hardware system.

The current FACETS Stage 2 system [2] consists of analog modeled IF (integrate-and-fire) neurons and synapses, which implement a STDP (spike-time-dependency-plasticity) mechanism. The neuron's and synapses' behavior is defined by a set of configurable parameters.

A hierarchical view of the system is illustrated in figure 2. The neural signals generated by the hardware neurons (block level), are propagated via a complex transfer network to be fed as stimuli into the hardware synapses or to be recorded externally. It is also possible to generate external stimuli and apply those to the neural network.

The transfer network is designed to provide as much topological flexibility as possible, while complying with the technological constraints. Generated neural signals are

encoded blockwise to a digital bus- (wafer level) and an overlaying package based network (system level). These contains configurable cross points and switches, duplicating and routing the signals to their destined synapses. After decoding the signals stimulate their assigned hardware dendrites, whose combinations form the receiving neurons. [3] Due to the system complexity and network properties several mapping and configuration constraints result, forming a multi-criteria optimization problem, which will not be discussed here further. [8]

2.3 FACETS Software Framework

The FACETS software framework consists of

- a collection of programs to carry out experiments (e.g. comparative experiments on neural software simulators and the FACETS hardware [4]),
- a database of benchmarks provided by FACETS partners (e.g. [5], [6]),
- a system simulation of the FACETS hardware [7],
- *software for mapping and configuration* [9],
- and *visualization and analyzation tools.* [8]

As presented in previous work [10] we developed a model, integrating the FACETS model descriptions for the mapping, configuration, visualization and analyzation software. Based on a graph, it stands for an universal data representation for the software framework. The mapping, routing and configuration algorithms retrieve their information from this *Graph Model (GM)*, process, transform them and write back the results. Furthermore the decentralized structure makes the model suitable for parallelization. A summarized model characterization is given in the next section to clarify the structure whereon the newly developed query language *GMPath* (see 3) bases.

2.4 The Graph Model to Navigate

The graph model $G = (V, E)$, basing on a hyper graph [11] [12], consists of single nodes $v_i \in V$, that may hold a name or a value as a data item respectively. The nodes can be assigned to each other by three types of directed single edges $e_i \in E$ as shown figure 3:

- *hierarchical edges* - modelling a hierarchy of two nodes, i.e. to represent a container-component relationship
- *named edges* - modelling a labeled relationship between two nodes
- *hyper edges* - assigning a named edge to a node, i.e. to model a detailed description of a node-node relationship.

Because of its structure the model is fully navigable, which means every node or edge is reachable from every position in the graph. So it is possible to create flexible models with respect to the biological networks and hardware system complexity, although it consumes more memory and setup time than more compact descriptions.

Given this model the biological and the hardware system can be described as two combined graphs $G_B = (V_B, E_B)$ and $G_H = (V_H, E_H)$ respectively, also containing meta information, e.g. algorithm configurations and results. The model transformation is not discussed here further [9] [10].

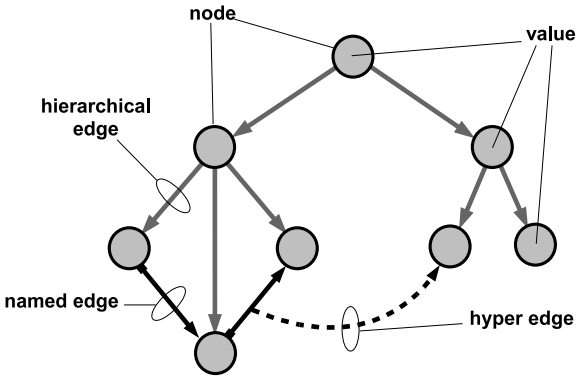


Fig. 3. Graph Model (example).

3 `GMPath` - A Language for Graph Model Navigation

The flexible and universal use of the GM by placing and routing algorithms, configuration, visualization and analyzation tools as well as the user defined modifications during run-time calls for an adequate and multi-purpose interface. The main requirements are:

- access from every node or edge as entry point
- step by step navigation along all elements (nodes and edges) follwing a path
- list based results, assignable to variables
- GM modification support

Inspired by the `XML` path language `XPath` [13] we developed a textual interface, named `GMPath`, to navigate in and retrieve information from the GM by entering at any node or edge, and address parts of it locally. Thus it can also classified as *Lokatorsprache*.

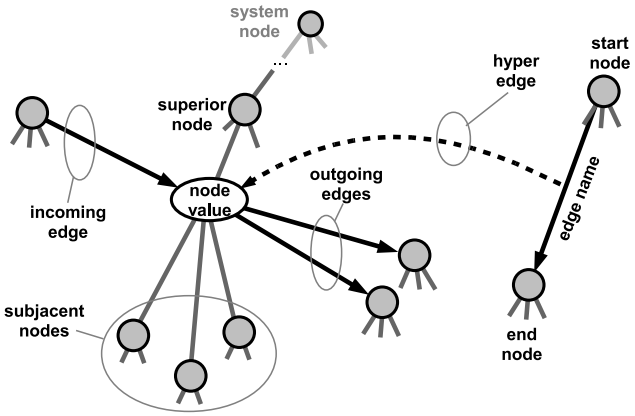


Fig. 4. Logical environment of GM nodes and edges.

Fig. 4 shows the logical environment of a GM node or edge respectively. Each adjacent element requires to be accessible within one ”separation step”, i.e. one basic navigational operation. Following the hierarchical structure the navigational focus should be able to shift from the current node (*node value*) up- or downward to the *superior node* or the *subjacent nodes*. Regarding the *named* and *hyper edges* it should be possible to move along these connections forward and backward to their *start* and *end nodes*.

Since all `GMPath` queries return result lists, containing the matching GM elements, they should be storable to variable names. By selecting these names, all nodes and edges can be used as an entry point for a new `GMPath` query. Furthermore `GMPath` requires to be able to address GM elements unambiguously to create new nodes and edges by this way. Thus, paths can built up during run-time by the software tools and processed with implemented API functions, parsing the paths and returning the result elements. This provides a dynamic interface to retrieve data from and manipulate the GM.

In the following sections the characteristics of `GMPath` and the query language `XPath` will be compared before the grammar and semantics are defined and explained consecutively.

3.1 Comparison to `XPATH`

As shown in the previous section the used data model is much leaner than a XML document, it does not include attributes and types of the contained data. However it is possible to interconnect nodes semantically via named edges to model distinct relationships between GM elements.

In opposition to `XPATH` `GMPath` does not distinguish between localization steps, axes tests or predicates, it only performs navigation steps along the GM structure with a matching result. Tests on subjacent nodes have to be executed as tests of reachability, comparisons of their assigned values are realized by localized `GMPath` queries.

In addition to the navigation along the hierarchy, known as axes in `XPATH`, localization or separation steps for named and hyper edges are necessary, to navigate strictly along the semantic relationships beside the hierarchical structure.

Furthermore in *Creation Mode* `GMPath` can be used to generate nodes and edges by addressing them unambiguously.

3.2 Grammar and Semantics

The grammar of `GMPath` is defined with the *GOLD Meta-Language* and written, tested and validated in the *Gold Parsing System* [14]. The meta-description uses the following elements to specify the *GMPath* grammar:

- *terminal symbols*, represented through Regular Expressions
- *rules*, using the Backus-Naur Form
- *character sets*, based on set notation

Terminal Symbols As shown in Tab. 1 `GMPath` contains five terminal symbols that cover the comment line (`%`) and two different general entry points ($SystemNode$ and

$HERE$) for the next query to start. Furthermore a wildcard symbol ($*$) encapsulates groups of GM elements without naming them and identifiers to store and represent node and edge lists.

Table 1. `GMPath` terminal symbols.

terminal symbol	description
$\%$	comment line
$*$	wildcard, placeholder for groups of nodes and edges
$SystemNode$	root node of the current GM, entry point of a query
$HERE$	current start element, entry point of a query
$(Edge-)Identifier$	(#) + alphanumerics + special characters for node and edge list names

Rules Rules define the syntax of the grammar. A `GMPath program` is a sequence of `operations` of arbitrary length, interpreted sequentially in order of their appearance.

```
<Program> ::= <operation> <Program>
```

`GMPath` differs between four different types of operations as shown in Tab. 2, which are explained below. Any operation in `GMPath` is delimited by a `\nl` symbol.

Table 2. `GMPath` operations.

operation	description
$commands$	switch between search only and manipulation mode
$assignment$	node or edge list assignment to variable names
$nodepathoperation$	path operation with a node list as result
$edgepathoperation$	path operation with a edge list as result

```
<commands> ::= 'EnableCreateMode' | 'EnableFindMode'
```

By default the `FindMode` is enabled. This means a query ending at non existing GM elements (e.g. a node name was not found) returns an empty list or a list with the invalid element respectively.

Enabling `CreateMode` causes the insertion of missing elements if addressed by a query unambiguously (e.g. a node name at a distinct position in the GM hierarchy).

```
<assignment> ::= Identifier '=' <node path operation> |
                 ConIdentifier '=' <edge path operation>
```

This operation assigns the result list of node- or edge path operations to identifiers to serve as substitutes. It is possible to embed identifiers in more complex operations to concatenate queries.

```
<node path operation> ::=
  <node path operation> {'/','//','\','\\'} <name> |
  <node path operation> '?(' <path operation> ')' |
  <edge path operation> {'>','<','^'} <name>  |
  <name>

<edge path operation>  ::=
  <node path operation> {'>','<','^'} <name> |
  <edge path operation> '?(' <path operation> ')' |
  <connection list name>

<path operation> ::= <node path operation> |
                     <edge path operation>
<name> ::= Identifier | '*' | 'SystemNode' | 'HERE'
<connection list name> ::= ConIdentifier | 'HERE'
```

`Node-` and `edge path operation`s are used to 'navigate' through the GM, using `separators` defining the logical direction (see also Fig. 4). On the one hand this can be done by moving along the hierarchy structure upward or downward node by node. On the other hand the GM elements are also accessible by navigating along the named- and hyper edges, passing alternately from node to edge forward and backward.

In general a `node-` or `edge path` operation is a concatenation of `node-` and `edge names` divided by `separators`, forming a path.

Table 3. GMPath separators.

separator	description
/	one step downward in hierarchy
//	all nodes below in hierarchy
\	one step upward in hierarchy
\\	all nodes above in hierarchy
>	outgoing edge(s), forward connected node(s)
<	incoming edge(s), backward connected node(s)
∧	to and from hyper connected node
?(`local query`)	intersection with local query

Separators The Separators are listed in Tab. 3. They are used to replace the current list of nodes or edges by a list of adjacent GM elements filtered by the given name, if not the wildcard symbol * is used alternatively. In other words the separators stand for basic 'moves' inside the GM as illustrated exemplary in Fig. 5. The intersection '?(`local query`)' processes an independent local query for every GM element as start position in the current list. Forming a filter function at the current path position, each GM element whose local query returns no result will be discarded.

Thus concatenating names of nodes and edges or wildcards with separators shapes complex GM queries by readdressing iteratively adjacent GM elements (see also Fig. 4). To clarify the usage of `GMPath` paths, some more advanced examples are appended in section 4.

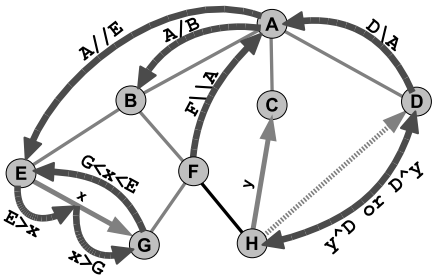


Fig. 5. Examples of basics operations (separators) to navigate through a simplified GM.

4 Examples of `GMPath` Usage

Based on the GM in Fig. 6 a few example queries may ease the understanding of `GMPath` and its application.

First a simple search query along the hierarchy and named edges is demonstrated, followed by a filter query based using local requests. The third example shows how to add new elements to the model.

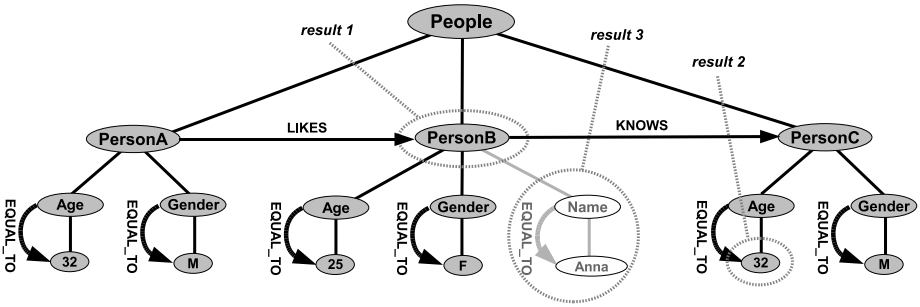


Fig. 6. Example GM for demonstrating `GMPath` queries.

4.1 Query 1 - Searching and Assignment

```
LikedPeople = SystemNode/*>LIKES>*
```

This query starts from the master node (*SystemNode*) and advances down hierarchically one 'separator step' (/), collecting all subjacent nodes from *PersonA* to *PersonC* due to the wildcard symbol (*). Defined by the next separator (>), the outgoing edges of all these three nodes are chosen. The following edge name *LIKES* selects only the edge from the *PersonA* node. The last separator (>) finalized by the wildcard symbol returns the target node of this edge regardless of the nodes name.

The resulting node list (*PersonB*) is stored to a variable named "LikedPeople" for later use.

4.2 Query 2 - Filtering

```
SystemNode/*?(HERE/Gender>EQUAL_TO>M)
  ?(HERE<KNOWS)/Age>EQUAL_TO>32
```

After collecting all person nodes two local queries (*?(..)*) are started. The first one selects only these with the gender of *'M'* (*PersonA*, *PersonC*), the second one checks for incoming *KNOWS* edges (*PersonC*). Afterwards the path steps down to the age of the resulting persons.

The query returns with the age of all male persons, who are known by someone (32).

4.3 Query 3 - Creating

```
EnableCreateMode
NameNode = SystemNode/PersonB/Name
NameValueNode = NameNode/Anna
NameNode > EQUAL_TO > NameValueNode
```

This example consists of more then one `GMPath` queries to create new GM elements, where the new line symbol is not shown.

First the creation mode is set by the *EnableCreateMode* command. In the next two lines new subjacent nodes (*Name* and *Anna*) are created below the *PersonB* node, addressed by their hierarchical locations. Assigned to two variable names (*NameNode* and *NameValueNode*) both new nodes are stored. The forth `GMPath` query, using the previous stored nodes, i.e. navigating from one list to the other by advancing forward along a not existing *EQUAL_TO* edge, creates the new connection and results in a structure as shown in Fig. 6.

5 Conclusions

This paper introduced the newly developed query language `GMPath`. `GMPath` bases on iterated basic localization steps through the logical environment of the current position, providing searching, filtering, storing and manipulation functions to the user. Queries start from every entry point of the model and can be build up dynamically by a program through available API functions.

In the FACETS software framework `GMPath` is used to provide the user with an universal interface to the internal graph models. Files containing `GMPath` commands are parsed before and after the programmatic GM creation, mapping and configuration process to allow modifications of parameters, of the model structure itself and the addition of user data. Furthermore `GMPath` forms an interface to graph models, which are displayed, debugged and analyzed by external visualizing and analyzation applications, allowing them to extract well-defined parts of the examined data model.

For the future we aim to expand the functionality of `GMPath`. We plan to implement set operations for the result lists, which are currently stored in variables (e.g. set unions, intersections and difference sets). Further the inclusion of `GMPath` queries

stored as strings in the data model itself should make the handling of complex queries more comfortable. Finally more options should be available to manipulate the GM (e.g. renaming or deleting elements).

Acknowledgements

The research project FACETS is financed by the European Union as Integrated Project (Nr. 15879) in the framework of the Information Society Technologies program.

References

1. Meier, K.: Fast Analog Computing with Emergent Transient States in Neural Architectures. Integrated project proposal, FP6-2004-IST-FET. Proactive, Part B. Kirchhoff Institut fr Physik, Ruprecht-Karls-Universität, Heidelberg (2004)
2. Schemmel, J., Grübl, A., Meier, K., Mueller, E.: Implementing Synaptic Plasticity in a VLSI Spiking Neural Network Model. Proceedings of the 2006 International Joint Conference on Neural Networks IJCNN 2006, pp. 1–6, IEEE Press (2006)
3. Ehrlich, M., Mayr, C., Eisenreich, H., Henker, S., Srowig, A., Grübl, A., Schemmel, J., Schüffny R.: Wafer-scale VLSI implementations of pulse coupled neural network. International Conference on Sensors, Circuits and Instrumentation Systems SSD'07, Hammamet-Tunisia (2007)
4. Brüderle, D., Grübl, A., Meier, K., Mueller, E., Schemmel, J.: A Software Framework for Tuning the Dynamics of Neuromorphic Silicon Towards Biology. Proceedings of the 2007 International Work-Conference on Artificial Neural Networks IWANN'07 Springer LNCS 4507, pp. 479–486, (2007)
5. Haeusler, S., Maass, W.: A Statistical Analysis of Information-Processing Properties of Lamina-Specific Cortical Microcircuit Models. Cerebral Cortex 17(1), pp. 149-162, (2007)
6. Vieville, T., Kornprobst, P.: Modeling Cortical Maps with Feed-Backs - International Joint Conference on Neural Networks (IJCNN), pp. 110-117 (2006)
7. Scholze, S., Ehrlich, M., Schüffny, R.: Modellierung eines wafer-scale Systems fr pulsgekoppelte neuronale Netze - Proceedings of Dresdener Arbeitstagung Schaltungs- und Systementwurf (DASS07), pp. 61–66 (2007)
8. Ehrlich, M., Wendt, K., Zühl, L., Brüderle, D., Vogginger, B., Müller, E.: A software framework for a wafer-scale neuromorphic hardware system, ANNIIP (2010), (submitted)
9. Wendt, K., Ehrlich, M., Mayr, C., Schüffny, R.: Abbildung komplexer, pulsierender, neuronaler Netzwerke auf spezielle Neuronale VLSI Hardware - Proceedings of Dresdener Arbeitstagung Schaltungs- und Systementwurf(DASS07) , pp. 127–132 (2007)
10. Wendt, K., Ehrlich, M., Schüffny, R.: A graph theoretical approach for a multistep mapping software for the FACETS project - Proceedings of the WSEAS CEA'08, pp. 189–194 (2008)
11. Diestel, R.: Graph Theory, Springer (2005)
12. Jordan, M.: Graphical Models. Computer Science Division and Department of Statistics. University of California, Berkeley, California 94720-3860, USA (2004)
13. Berglund, A., Boag, S., Chamberlin, D., Fernandez, M., Kay, M., Robie, J., Simeon J., editors. XML Path Language (XPath) 2.0. W3C Working Draft 02, W3C (2003)
14. http://www.devincook.com/goldparser/. *GOLD Parser System* (2007)

A Software Framework for Mapping Neural Networks to a Wafer-scale Neuromorphic Hardware System

Matthias Ehrlich[1], Karsten Wendt[1], Lukas Zühl[1], René Schüffny[1]
Daniel Brüderle[2], Eric Müller[2] and Bernhard Vogginger[2]

[1] Technische Universität Dresden, Lehrstuhl für Hochparallele VLSI-Systeme und Neuromikroelektronik, 01062 Dresden, Germany
{ehrlich, wendt, schueffny}@iee.et.tu-dresden.de
lukas.zuehl@mailbox.tu-dresden.de
[2] Ruprecht-Karls-Universität Heidelberg, Kirchhoff-Institut für Physik
69120 Heidelberg, Germany
{bruederle, mueller, bernhard.vogginger}@kip.uni-heidelberg.de

Abstract. In this contribution we will provide the reader with outcomes of the development of a novel software framework for an unique wafer-scale neuromorphic hardware system. The hardware system is described in an abstract manner, followed by its software framework which is in the focus of this paper. We then introduce the benchmarks applied for process evaluation and provide examples of the achieved results.

1 Introduction

Several current neuromorphic research projects, such as *Fast Analog Computing with Emergent Transient States – FACETS* [1] or the *Spiking Neural Network Simulator – SpiNNaker* [2], aim at the exploration of novel computational aspects of large scale, biologically inspired neural networks with over a million neurons, simulated in real-time or even with a speed-up in respect of the biological archetypes on full custom or modified general purpose hardware.

The undertaken hardware research of FACETS encompasses the development of a novel neuromorphic wafer-scale hardware system in an collaborative effort of the *Ruprecht-Karls-Universität Heidelberg – UHEI* and the *Technische Universität Dresden – TUD*. The current level of development, *Stage 2* incorporates the design of a wafer element and its dedicated software framework for the mapping of neural architectures onto the hardware substrate as well as the configuration and control of said system.

The wafer-scale hardware system is first described in section 1.1 followed by the details of the software framework in section 2. The benchmarks applied are presented in section 3 along with examples. An outlook concludes this contribution.

1.1 FACETS Stage 2 Architectural Overview

For the description of the FACETS Stage 2 hardware system as introduced by [1], [3] and in the following referred to as FS2 hardware we will focus on details of the architecture that influence the mapping of given neural networks onto the hardware. Figure 1 (a)

shows an abstract view of one wafer element of the FS2 hardware system. The foundation layer of the FS2 hardware is an array of reticles shown as light gray squares, housing *High Input Count Analog Neural Network – HICANN* or HC circuitry that was developed at UHEI [1] and implements neural functionality such as neurons, synapses and weight adaptation. On top resides a layer of communication circuits called *Digital Network Chip – DNC* developed at TUD [3]. The third and topmost layer represents a regular grid of FPGAs[3], colored dark gray. Disabled or inoperable components are colored white.

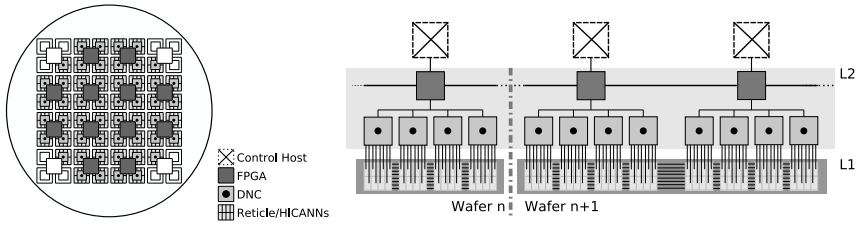


Fig. 1. Abstract view of a) one wafer from top and b) the communication hierarchy from side.

Figure 1 (b) depicts the communication networks, their hierarchy and connectivity. Two distinct communication networks can be distinguished. An asynchronous, address coded, named *Layer 1 – L1* utilized by HCs at wafer level for *intra-wafer* communication and a second one, named *Layer 2 – L2* utilized by DNCs and FPGAs for synchronous, packet based *inter-wafer* communication. Host computers are connected via *Ethernet* to the FPGAs to handle the mapping, configuration and control process described in the following.

1.2 The HICANN

A simplified view of the HC chip following [1], [4] is drawn in figure 2 as a symmetric array of neural and communication elements. The *dendritic membranes*, or *denmems* are the neural core components. Each denmem provides two synaptic input circuits emulating ion channels. Up to 2^6 denmems can be grouped, i.e. connected together to form a neuron with a higher synaptic input count or a more detailed model by increasing the number of conductive time constants. *Synapses*, situated in an adjacent *synapse array* are connected to the denmems. Whether a synapse is connected to the excitatory or inhibitory input of a denmem is decided row-wise in the *synapse driver*, or *syndriver*. A syndriver is fed from one of 2×2^7 vertical L1 bus lanes via *select-switches* or from a neighboring syndriver. It drives the synapses via *strobe lines*, as depicted as thin lines in figure 2 lens (1), and selects the receiving synapse via an address, the thick lines. A fixed part of the synapses address determines the strobe line to use and follows the address pattern shown in lens (2). Each synapse belongs to the denmem located below the synapse array in the same column. A group of denmems is connected to one of 2^6

[3] Field Programmable Gate Array

horizontal L1 bus lanes and L2 by a *priority-encoder* that multiplexes and prioritizes the bus access.

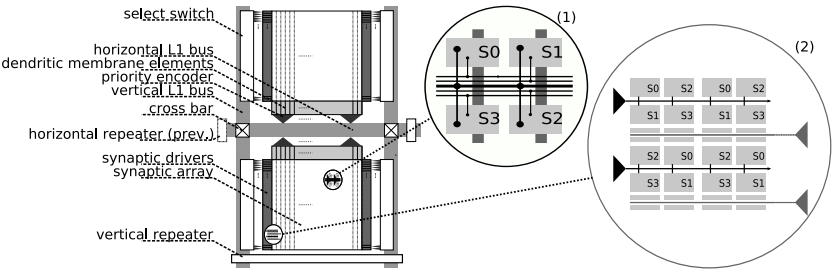


Fig. 2. A schematic view of one HICANN [1], [4].

Repeaters and *cross-bars* are then configured to interconnect the vertical and horizontal buses with *unidirectional* connections. The neural pulses generated by the denmems are transmitted asynchronously on L1 as bit sequence encoding the senders address or arbitrarily on L2 encoding the address and the pulse timing.

1.3 Parameter Space

Every denmem implements the dynamics of the *Adaptive Exponential Integrate-and-Fire – AdEx* model [5] including model's mechanisms such as spike frequency adaption and active spike generation. A total of 24 parameters determine the behavior of a denmem, some of which correspond directly to the AdEx model, others are of technical nature[4].

The synaptic weight of a synapse is determined by an individual digital weight value of 4-bit resolution and a fixed maximum conductance $g_{\max}$, which can be set for every synapse row by a programmable analog parameter. The synapse circuit generates a square current pulse, which is injected into one of the synaptic input circuits of the denmem, where it modulates a transient synaptic conductance. The amplitude of this square current pulse is $weight \times g_{\max}$ and its length is τ_{STDF}, where τ_{STDF} is modulated by the *short term depression or facilitation – STDF* [6] plasticity mechanism in the synapse driver.

We assume a hardware model setup for configuration of the FS2 hardware following [1], [4]. With an 8×8 HC reticle array of 8 HCs per reticle and 48 functioning reticles per wafer, thus a total of 512 HCs. Furthermore, 8 HCs per DNC result in 48 DNCs and 4 DNCs per FPGA give a total of 12 FPGAs. With $N_{MaxHC} \in \left\{2^3, 2^4, ..., 2^8\right\}$[5] a

[4] As configurable parameters allow to vary time constants of neural and synaptic dynamics it is possible to operate the FS2 hardware system with a speed-up from 10^3 to 10^5 compared to biological scale, depending on the system's load, as excessive speed-up may lead to pulse loss due to limited bandwidth.

[5] N_{MaxHC} is held constant for a network and determined by the detail level of the neuron model [1] or the synaptic input count of a neuron [4].

maximum neurons per HC the total number of available neurons is given by $N_{HW} = H \times N_{MaxHC}$, where H denotes the number of HCs available for mapping[6]. The number of synapses available on the hardware $S_{HW} = H \times S_{HC}$, with S_{HC} being number of synapses per HC, which for the used configuration is constant with: 2×256^2 and the number of dendritic elements per HC D which equals 2×256. With 2^6 denmems per priority encoder this results in 8 priority encoders and thus a 6-bit L1 address.

2 The FACETS Stage 2 Software Framework

The FS2 software framework provides the functionality to map a given network onto the hardware, configure it, control the simulation and examine the results of the mapping and simulation process.

2.1 PyNN & Hardware Abstraction Layer

For the FACETS hardware systems, a user interface is now available that provides a novel way to bridge the gap between the domains of pure software simulators and neuromorphic hardware devices [7], [8]. The Python-based neural network modeling language PyNN [9], see Figure 3 has been developed by FACETS members. It represents a simulator-independent set of functions, classes and standards for units and random number generation that can be used to describe complex models of networks of spiking neurons using a biological terminology - either in an interactive or in a scripting fashion. Models written with the PyNN API can be executed with various established software simulation tools such as NEURON [10], NEST [11], Brian [12] or PCSIM [13]. For all supported back-ends a specific Python module automatically translates the PyNN code into the native scripting language of the individual simulator and re-translates the resulting output into the domain of PyNN. Thus, PyNN allows to easily port experiments between all supported simulators and to directly and quantitatively compare the results. Among many other benefits, this unification approach can increase the reproducibility of experiments and decreases code redundancy.

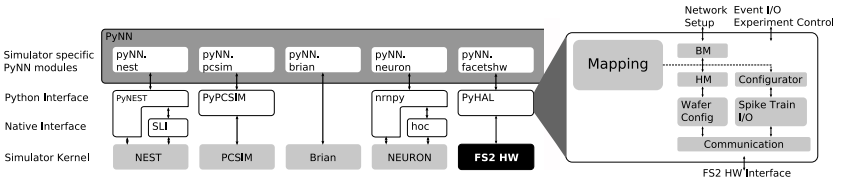


Fig. 3. PyNN framework following [9] and the FS2 HAL.

The integration of the FACETS hardware systems into the PyNN concept adopts these benefits. Additionally, the PyNN hardware module offers a transparent method via which the communities of computational neuroscience and neuromorphic engineering

[6] H is not necessarily equal to the total number of HCs available in the system.

can exchange experiments and results. With the novel approach, non-hardware-experts can be provided with a well documented interface that is very similar to interfaces of most established software simulators [14].

While PyNN itself represents a precise definition of the user interface, the *Hardware Abstraction Layer – HAL* module actually *implements* the automated translation of any given network setup into the data model described in the following, which performs the mapping of the experiment onto the available hardware resources and into the hardware parameter domain. The said translation process also conducts the transition between the Python domain of PyNN and the C++ objects of the mapping framework and all lower software layers.

2.2 Data Model

To cope with the hierarchical structure of the hardware system a data model resembling a hierarchical hyper graph was developed [15]. The graph model consists of *vertices* representing data objects and *edges* as relationships among them. Where a vertex holds atomic data, an edge can be a *hierarchical*, a *named* or a *hyper* edge. Hierarchical edges model a parent-child relationship, thus structuring the model. Named edges form a directed and named relation between two vertices from/to any location in the model and hyper edges assign a vertex to a named edge, characterizing it in more detail. Its flexibility allows to store every information during the configuration process, i.e. the models itself as well as the placement, routing and parameter transformation data.

2.3 Data Interface

To overcome the access of nodes and edges or subsets of the graphs elements by navigating the native data structure we provide a novel *path-based query-language*, named `GMPath`. Via GMPath, along with its corresponding API as described in the accompanying publication [16] data can be retrieved from or stored to the models by a program via static or dynamically created queries.

2.4 The Mapping Process

With regard to topology constraints between hardware blocks such as connectivity, connection counts, priorities and distances as well as source/target counts the mapping determines a network configuration and parameter set for the hardware. This is accomplished in the three steps of *placement*, *routing* and *parameter transformation*.

During placement, the mapping process assigns neural elements like neurons or synapses to distinct hardware elements. As placement comprises different optimization objectives, it can be characterized as a multi-criteria problem the solution quality of which influences the overall mapping results significantly. Possible objectives are, e.g. to minimize the neural input/output variability clusterwise, to minimize the neural connection count, also clusterwise, or to minimize routing distances while maintaining compliance with constraints such as parameter limitations or hardware element capacities. As the optimization problem is NP-complete a force-based optimization heuristic

with user-defined weightings, named NFC, was developed to achieve these objectives in acceptable computation time. This algorithm balances ”forces”, the implementation of said optimization objectives in an n-dimensional space until an equilibrium is reached. In a subsequent separation step it assigns its data objects to clusters with affine properties. We distinguish between the simple algorithms described in [17] and the NFC.

The routing subsequently determines a configuration for the synaptic connections on L1 and L2 and can be split into the two subsequent steps of intra- and inter-wafer routing. The intra-wafer routing algorithms [4] route connectivity exclusively on L1 and reserve L2 for inter-wafer routing which is inactive for a wafer-scale system.

Parameter transformation finally maps the model parameters of given neurons and synapses, such as weights, types or thresholds into hardware parameter space. As not every biological parameter, or its corresponding model parameter in the PyNN description, has its individual counterpart in hardware but is often emulated by a set of correlating parameters, an adequate biology-to-hardware parameter translation has to be found, e.g for the membrane circuits a transformation from 18 biological parameters of the PyNN AdEx neuron model description into a configuration of 24 adjustable electrical hardware parameters.

The desired speedup factor between 10^3 to 10^5 which is determined by the temporal dynamics of the membrane and synaptic circuitry is finally set by adjusting parameters as the size of the membrane capacitances, conductances responsible for charging it or the current controlling the synaptic conductance.

2.5 Analysis

A new standalone application named *Graph Visualization Tool – GraViTo* aids the user with the analysis and debugging of mapping data. GraViTo incorporates envisioNN and H3 graph viewer [18] modules that display graph models in textual and graphical form and gathers statistical data. One can selectively access single nodes inside the data structure and visualize its context, dependency and relations with other nodes in the system.

Views of GraViTo are shown in figure 4, such as the *tree view* to browse the hierarchical structure of the graph model, the GMPath *query view* and the *3D view*. The 3D view is specialized on rendering BM and HM and the mapping between them in three dimensional form to provide a contextual view over the models, their components and connectivity. It also provides a global overview over the hardware components and the networks. To support the analysis of the mapping results various statistics are gathered and displayed, e.g. as histograms for utilization of the crossbars, the HC blocks or the synaptic connection lengths.

3 Benchmarks

Benchmarks aid in evaluating the mapping process. First benchmarks concerning mapping efficiency with focus on intra-wafer routing and hardware utilization were carried out at UHEI [4] with random networks, macrocolumns and locally dense/globally

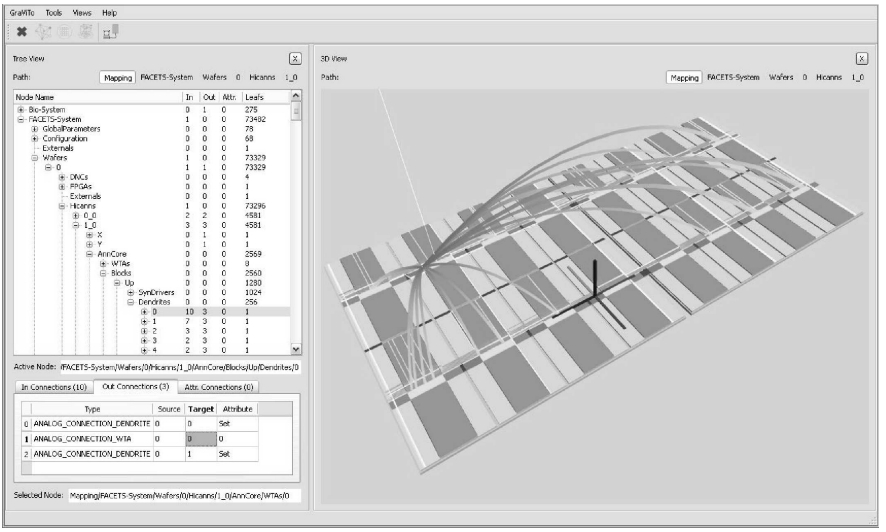


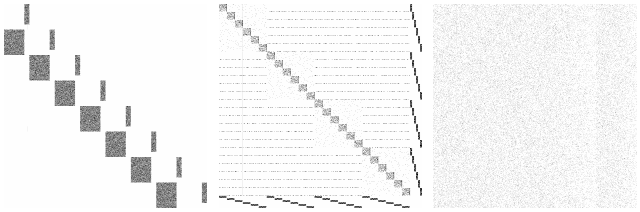
Fig. 4. Screenshot of GraViTo's viewers.

sparse connected networks in order to explore the system's design space. New benchmarks are listed in table 1. The new benchmarks are implemented in PyNN and were provided from FACETS project partners but also from the neuromorphic research community outside of FACETS.

Table 1. Selected Benchmarks.

Benchmark	Description
INCM ALUF	Synfire Chain based on [19], provided by *L'Institut de Neurosciences Cognitives de la Méediterranée – INCM, Marseille, France* in cooperation with *Albert-Ludwigs-Universität Freiburg – ALUF, Freiburg, Germany*
KTH	Layer 2/3 Attractor Memory following [20], provided by *Kungliga Tekniska Högskolan - KTH, Stockholm, Sweden*
UNIC	Model of Self-Sustained AI States following [21], provided by the *Integrative and Computational Neuroscience Unit – UNIC* of the *Centre national de la recherche scientifique – CNRS, Gif-sur-Yvette, France*

As an example we apply the mapping process to the scaled benchmarks in a 4×4 reticle configuration with an $N_{MaxHC} = 2^6$ to evaluate the mapping *quality*. As a measure of the overall mapping quality the parameters as defined in [4] apply. The *routing quality* $q_{Route} = S_{Map}/S_{BIO}$, with S_{Map} being the number of mapped synapses over S_{BIO}, which is the number of synapses in the BM. Thus, $(1 - q_{Route})$ is the *relative synapse loss*. The *hardware efficiency* is described by $e_{HW} = S_{Map}/S_{HW}$, where S_{HW} denotes the synapses available on the FS2 hardware for mapping. As a further parameter for network classification we define the connection density $\rho_{Syn} = S_{BIO}/N_{BIO}^2$.



(a) (b) (c)

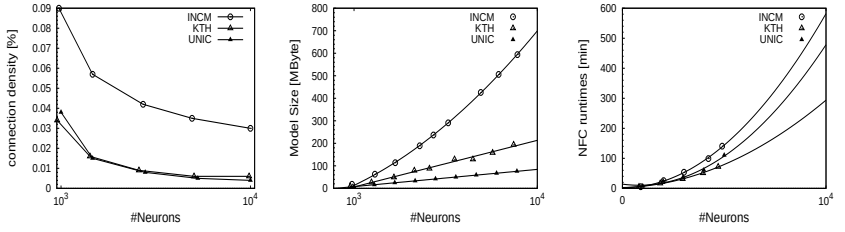
Fig. 5. Connection matrices of the (a) INCM, (b) KTH and (c) UNIC networks.

Connection matrices for networks of 10^3 neurons as shown in figure 5 illustrate the benchmarks synaptic connectivity types. Darker areas represent groups of neurons with an ρ_{Syn} above average.

As stated in [2] the worst scenario are randomly connected networks with a constant ρ_{Syn} due to their absent locality. In case of avg. S_{BIO} above the configured `HW` limit one may reduce the neurons per `HC`, provide more synapses and thus improve q_{Route} at the expense of less e_{HW}, but an expanded distribution of neurons and thus longer connections may consume even more routing resources in turn at a certain point again reducing q_{Route}.

The ρ_{Syn} of the benchmarks however decrease with approx. $1/x$, see 6 (a) leading to an almost constant or only slightly increasing average synaptic input count. Nevertheless the mapping results for networks with N_{BIO} above 10^5 show a clear decrease in q_{Route} by exceeding 15% compared to fully routed which may be caused by intra-wafer routing resources utilized to capacity, invigorated by an observation of the steepest decline in q_{Route} for UNIC, the network with the lowest avg. ρ_{Syn}.

Tests also showed that the `NFC` algorithm can minimize the routing losses compared to the simple algorithms up to 20% for networks with a higher locality, such as the INCM, the more efficient the larger the network.



(a) (b) (c)

Fig. 6. Networks avg. ρ_{Syn} (a), `BM` size (b) and `NFC` algorithm runtime (c).

As a second major requirement for the usability of the `FS2` hardware simulator platform a fast configuration and reprogramming is inevitable so we use the scaling test also to determine the software process' *scalability* in terms of time and space.

Figure 6 (b) shows that the `BM` graph grows almost linearly depending on the number of neurons and the synaptic density. So for the given benchmarks the model sizes for

networks with a neuron count of $N_{BIO} \leq 10^5$ and an approximate average $\rho_{Syn} \leq 10\%$ stay within a acceptable limt of $10GB$. The simpler algorithms runtime scales with $O(n)$ and remains within an upper bound of approximately 3 hours whereas the NFC algorithms, in spite of the cubical problem, grows below $O(n^2)$, as can bee seen in 6 (c) fulfilling the requirement of a resonable runtime for complex mapping problems.

Test where carried out under *Red Hat* $4.1.2$ running on an AMD Opteron™ 875 Dual Core CPU $@2.2GHz$ quad processor system with $32GByte$ of RAM.

4 Conclusions

Although the FS2 hardware system is on a higher level of abstraction similar to other reconfigurable hardware architectures it is unique in both its functionality and the systems dimension. So new algorithms and heuristics are necessary that take into account the peculiarities of such a system. We presented outcomes and benchmark examples of the complete FS2 software framework which seamlessly integrates the FS2 hardware system into PyNN.

As shown by the benchmarks, a mapping is found in a reasonable time, however, the networks structure of larger networks is modified by the software process and through hardware resource limitations. To examine the impact of these losses on the networks behavior comparative simulations with pre- and post- mapping netlists are carried out on simulators introduced in section 3. As a further consequence we consider the incorporation of L2 into intra-wafer communication as essential as it will alleviate the L1 losses. Iterative optimization of the mapping results will then trade-off between simulation speedup, hardware efficiency and routing quality by adjusting the software process parameters.

An in depth evaluation of the benchmark results will follow with the upcoming publication of the NFC algorithm.

Acknowledgements

The research is financed by the European Union in the framework of the Information Society Technologies program, project FACETS (Nr. 15879). Furthermore, we would like to thank *Jens Kremkow* of ALUF, *Pradeep Krishnamurthy* of KTH and *Andrew Davison* of CNRS for making the PyNN scripts available to us.

References

1. Schemmel, J., Fieres, J., Meier, K.: Wafer-scale integration of analog neural networks. In: Proceedings IJCNN2008, IEEE Press. (2008) 431–438.
2. Khan, M., Lester, D., Plana, L., Rast, A., X.Jin, Painkras, E., Furber, S.: SpiNNaker: Mapping Neural Networks onto a Massively-Parallel Chip Multiprocessor. In: Proceedings 2008 International Joint Conference on Neural Networks, IJCNN 2008. (2008) 2849 – 2856.
3. Ehrlich, M., Mayr, C., Eisenreich, H., Henker, S., Srowig, A., Gruebl, A., Schemmel, J., Schueffny, R.: Wafer-scale VLSI implementations of pulse coupled neural networks. In: International Conference on Sensors, Circuits and Instrumentation Systems SSD'07. (2007)

4. Fieres, J., Schemmel, J., Meier, K.: Realizing Biological Spiking Network Models in a Configurable Wafer-Scale Hardware System. In: IEEE International Joint Conference on Neural Networks IJCNN. (2008) 969 – 976.
5. Brette, R., Gerstner, W.: Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity. Journal of Neurophysiology 94 (2005) 3637–3642
6. Schemmel, J., Brüderle, D., Meier, K., Ostendorf, B.: Modeling Synaptic Plasticity within Networks of Highly Accelerated I&F neurons. In: Proceedings of the 2007 IEEE International Symposium on Circuits and Systems (ISCAS'07), IEEE Press (2007)
7. Brüderle, D., Müller, E., Davison, A., Muller, E., Schemmel, J., Meier, K.: Establishing a Novel Modeling Tool: A Python-based Interface for a Neuromorphic Hardware System. Front. Neuroinform. 3 (2009)
8. Davison, A., Muller, E., Brüderle, D., Kremkow, J.: A common language for neuronal networks in software and hardware. The Neuromorphic Engineer (2010)
9. Davison, A. P., Brüderle, D., Eppler, J., Kremkow, J., Muller, E., Pecevski, D., Perrinet, L., Yger, P.: PyNN: a common interface for neuronal network simulators. Front. Neuroinform. 2 (11) (2009) 1 – 10
10. Hines, M. L., Carnevale, N. T.: The NEURON Book. Cambridge University Press, Cambridge, U.K. (2006)
11. Gewaltig, M. O., Diesmann, M.: NEST (NEural Simulation Tool). Scholarpedia **2** (2007) 1430
12. Goodman, D., Brette, R.: Brian: a simulator for spiking neural networks in Python. Front. Neuroinform. 2 (2008)
13. Pecevski, D. A., Natschläger, T., Schuch, K. N.: PCSIM: A Parallel Simulation Environment for Neural Circuits Fully Integrated with Python. Front. Neuroinform. 3 (2009)
14. Brüderle, D., Bill, J., Kaplan, B., Kremkow, J., Meier, K., Müller, E., Schemmel, J.: Simulator-Like Exploration of Cortical Network Architectures with a Mixed-Signal VLSI System. In: Proceedings of the 2010 IEEE International Symposium on Circuits and Systems (ISCAS'10). (2010) *Accepted*
15. Wendt, K., Ehrlich, M., Schüffny, R.: Graph theoretical approach for a multistep mapping software for the FACETS project. In: 2nd WSEAS Int. Conference on Computer Engineering and Applications (CEA'08). (2008)
16. Wendt, K., Ehrlich, M., Schüffny, R.: GMPath - A Path Language for Navigation, Information query and modification of data graphs. In: 6th International Workshop on Artificial Neural Networks and Intelligent Information Processing (ANNIIP). (2010) *Accepted*
17. Wendt, K., Ehrlich, M., Mayr, C., Schffny, R.: Abbildung komplexer, pulsierender, neuronaler Netzwerke auf spezielle neuronale VLSI Hardware. Dresdner Arbeitstagung Schaltungs- und Systementwurf DASS 2007 (2007) pp. 127–132
18. Munzner, T.: H3: Laying out large directed graphs in 3d hyperbolic space. In: Proceedings of the 1997 IEEE Symposium on Information Visualization. (1997) 2–10
19. Kremkow, J., Perrinet, L., Aertsen, A., Masson, G.: Functional consequences of correlated excitatory and inhibitory conductances. (2009) *Submitted*
20. Lundqvist, M., Rehn, M., Djurfeldt, M., Lansner, A.: Attractor dynamics in a modular network of neocortex. Network:Computation in Neural Systems 17:3 (2006) 253–276
21. Destexhe, A.: Self-sustained asynchronous irregular states and Up/Down states in thalamic, cortical and thalamocortical networks of nonlinear integrate-and-fire neurons. Journal of Computational Neuroscience 3 (2009)

Design of a Multi-Agent System for Hierarchical Network Management in Wireless Sensor Network

Mubashsharul I. Shafique, Haiyi Zhang and Yifei Jiang

Jodrey School of Computer Science, Acadia University
Wolfville, NS, B4P 2R6, Canada
{093080s, haiyi.zhang, 095997j}@acadiau.ca

Abstract. In order to manage a sensor network efficiently, we can divide it logically into disjoint parts, called clusters. As sensor nodes are resource-constraint, it is desirable to chose suitable cluster heads and do role-switching of the cluster heads wherever appropriate. In this work, we design a system that selects new cluster head through collaboration of multiple software agents in each cluster. Our system allows to pick up cluster heads dynamically based on current network status. Agents in our design use *Fuzzy Logic*-based controller to find new cluster heads.

1 Introduction

Wireless Sensor Network is a deployment of sensor nodes that do data reporting to interested user via sink node. It offers a great facility to remotely monitor an unattended environment. Due to this feature, sensor networks are widely used in military surveillance, habitat monitoring, industrial plants, and in many other places. Once a sensor network is deployed, it is necessary to manage it efficiently in order to optimize network life-time.

A sensor network can be managed by hierarchical or flat organization. However, as hierarchical organization has several benefits over flat network structure, normally a sensor network is divided into clusters. Here in this work, we design a multi-agent system to rotate the role of cluster head nodes. Software agents in our design autonomously collaborate with each other in the distributed sensor network environment. The remaining part of this paper is organized as follows: section 2 presents some background knowledge and section 3 defines the problem that his work addresses. Next, section 5 explains the design of our system and inter-agent communication is presented in section 6. We conclude our paper in section 7.

2 Background Knowledge

2.1 Wireless Sensor Network

Wireless Sensor Network(WSN) consists of autonomous sensor devices (known as nodes or motes) that can sense an environment. For example, MicaZ sensor nodes from

Crossbow[1] can sense ambient light, barometric pressure, GPS, magnetic field, sound, photo-sensitive light, photo resistor, humidity and temperature.

Often times, these sensor nodes are battery-powered, and equipped with limited processing capacity and memory. However, WSNs are very effective and efficient at monitoring remote environment and communicate real-time data. Sensor nodes can detect events or phenomena, collect and process data, and transmit sensed information to the interested users.

2.2 Virtual Organization in Sensor Network

WSN differs from an IP network in regards to network backbone because, often times a WSN does not have any fixed infrastructure. Sensor nodes may run out of battery power or network topology may change due to some mobile nodes. So, in order to support data routing, WSN relies on virtual infrastructure which forms on-the-fly during system operation. One way to manage a sensor network, is to logically divide it into some disjoint clusters. A particular node in each cluster works as cluster head, and gathers data from other nodes of the cluster. The cluster head then forwards data to the next hop node towards the data sink, which is the final destination of any data. Figure 1 shows an example sensor network with three clusters. We show data reporting from a normal sensor node to the sink by arrow heads.

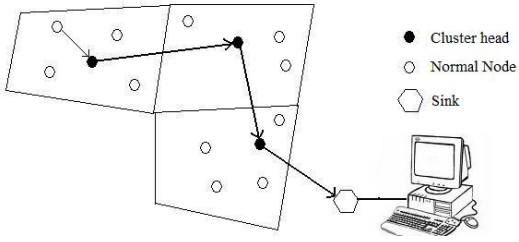


Fig. 1. Virtual organization in WSN.

3 Problem Definition

In a static network topology, cluster heads of a sensor network die quite early due to excessive relaying of data stream towards sink. So, in order to prolong network lifetime, a network should rotate the role of cluster heads. For example, if a current cluster head had drained a significant amount of energy, network should keep the provision to pick a new cluster head for that cluster. This kind of dynamic load balancing can delay the first node death and help decrease data packet loss in the network thereby.

Due to the large number of sensor nodes in a typical sensor network, it is not realistic to run any centralized algorithm to switch cluster heads. A network needs to have distributed architecture which can select new cluster heads with local information, and further, the role-switching should remain quasi-transparent to the remaining network.

[1] http://www.xbow.com/.

4 Related Works

For Wireless sensor Networks (WSNs), the major design goal is to minimize energy consumption and maximize the network lifetime. In the last few years, plenty of attempts of exploring advanced power conservation approaches have been used by researchers for wireless sensor networks. Cluster-based routing approach is one of the famous energy efficient routing approaches in WSNs. LEACH (Low Energy Adaptive Clustering Hierarchy) in [1], is the first hierarchical cluster-based routing protocol for WSNs. This algorithm uses random and periodic rotation of the Cluster Heads (CHs) for load balancing, which can evenly disperse the energy load among the sensor nodes in the network. More specifically, the cluster heads belonged to the corresponding clusters will only use for certain number of rounds. After a predefined round, new cluster heads will be randomly generated, which is based on a role that if the random number is less than a calculated threshold $T(n)$, the corresponding sensor node will be selected as the cluster-head for the current round. This randomized periodic role rotation ensures that all the nodes are equally likely to be cluster head nodes. However, simply random cluster head rotation will select unfavorable cluster heads, which will turn out high energy consumption in later rounds.

Due to the above reason, in [9], Energy-LEACH protocol improves the procedure of CH rotation. Similar to LEACH protocol, the process of CH rotation of E-LEACH is divided into rounds. In the first round, each sensor node has the same probability to be turned into CH, which means sensor nodes are randomly selected as CHs. In the next rounds, since the residual energy of each sensor node is different after every communication round, residual energy of node is considered as the main metric that decides whether the sensor nodes turn into CHs or not after the first round. The sensor nodes who have more remaining energy will become CHs rather than the ones with less remaining energy.

As for the aforementioned two LEACH algorithms, there is a shared drawback, that is, both of them consumes more CPU cycles, since each sensor node in WSNs has to calculate the threshold and generate the random numbers in each round. To overcome this shortcoming, a improved LEACH, called LEACH-C protocol, is proposed in [10]. LEACH-C is a centralized clustering algorithm and uses the same steady-state phase as in LEACH algorithm. This protocol also produces better performance on cluster heads rotation. During the set-up phase of LEACH-C, each sensor node sends information about its current location (this may determine by using GPS) and residual energy level to the Base station (BS). In order to ensure that the energy load is evenly distributed among all the sensor nodes in WSNs, The average node energy is computed by the BS, and then, determines which sensor nodes have energy below this average. Once the CHs and associated clusters are found, the BS broadcasts a message that obtains the cluster head ID for each node. If a cluster head ID matches its own ID, the node will be selected as a cluster head; otherwise the node determines its TDMA slot for data transmission and goes sleep until it's time to transmit data. The steady-state phase of LEACH-C is identical to that of the LEACH protocol.

Based on the above approach, in [11], cluster heads election using fuzzy logic is proposed, which can minimize energy consumption and provide a substantial increase in network lifetime compared with the probabilistically cluster heads selecting

approaches. According to this proposed approach, for a cluster, the node elected by the base station is the node having the maximum chance to become the cluster-head, which is based on three fuzzy descriptors: energy level in each sensor node, sensor node concentration and node centrality with respect to the entire cluster. The operation of this fuzzy logic cluster-head election scheme is divided into two rounds with each consisting of a setup and steady state phase similar to LEACH algorithm. During the setup phase, fuzzy knowledge processing is used for determining the CHs, and then the cluster is organized. In the steady state phase, the aggregated data is collected by CHs. After that, CHs perform signal processing functions to compress the data into a single signal. This composite signal is then sent to the base station. Fuzzy logic control model is core part of this proposed approach; it includes a fuzzifier, fuzzy rules, fuzzy inference engine, and a defuzzifier. To be specific, fuzzifier is used to take the crisp inputs from each of variables of energy, concentration and centrality and determine the degree to which these inputs belong to each of the appropriate fuzzy sets. Then, these fuzzified inputs are applied to the antecedents of the fuzzy rules. As for the defuzzification, the input for this process is the aggregate output fuzzy set chance and the output is a single crisp number. For fuzzy inference engine, Mamdani Method [13] is commonly used. In [12], similar cluster heads selections based on fuzzy logic algorithm are also presented.

To our knowledge, several attempts have also been used by some researchers to reduce energy consumption based on mobile agents [2][3][4][5]. Due to the constraints of bandwidth in wireless sensor network, the network's capacity may not satisfy the transmission of sensory data. In order to handle the problem of overwhelming data traffic, Qi, et al. [6] proposed Mobile Agent-based Distributed Sensor Network (MADSN) for multi-sensor data fusion. For this proposed approach, it not only achieves data fusion, but also reduces energy expenditure. However, the application of this approach can only be applied on cluster-based topologies. MADD approach in [7] is introduced to deal with this problem. Currently, most energy-efficient proposed approaches are focused on data-centric model, such as the directed diffusion. By selecting good path to drain quality data from source nodes, directed diffusion approach can achieve substantial energy gain. However, it still allows redundant sensory traffic to flow back to the Base station. The main advantage of MADD is to reduce the redundant sensory data. Through using mobile agent, data is aggregated at each source node and is brought back to sink. This allows substantial energy gain toward the network lifetime.

To explain the process of MADD approach, it starts when the mobile agent is dispatched from the BS with the interest and ends when it returns to the sink with the aggregated data. The processes involved in MADD are divided into three phases. First, the mobile agent is dispatched from BS to the first source node. Second, the mobile agent shifts from the first source node to the last source node, visiting selected source nodes in between. The drawback for this approach is that it doesn't always guarantee the best sequence of nodes to be visited.

To deal with the aforementioned limitations, Shakshuki et al. in [8] proposed a mobile agent for efficient routing approach (MAER) by using both Dijkstra's algorithm and Genetic Algorithms (GAs). As we know, the order of source nodes to be visited by the mobile agent greatly affects the energy consumption. Although MADD, the work presented in [7], allows the agent to autonomously select visit sequence of source nodes

for achieving data aggregation, it does not always provide an optimal sequence. To address this shortcoming, MAER in [8] introduces Genetic Algorithm (GA) to produce an optimal route.

5 Agent Architecture

An agent in our design consists of four modules: Problem Solver, Knowledge-base Updater, Scheduler, and Communicator. Further, each agent maintains own knowledge base in its memory, which gets updated as a result of inter-agent message communication. Here in this section, we describe different components of individual agent in detail.

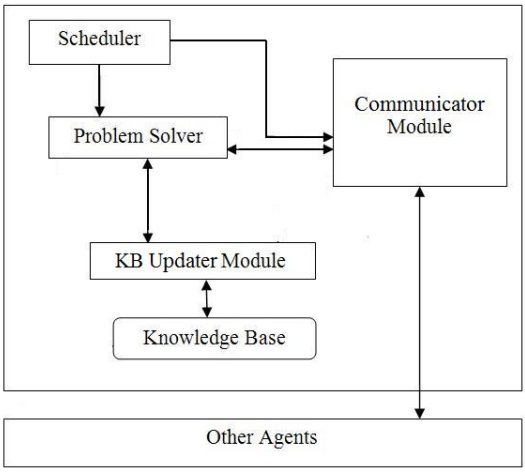


Fig. 2. Agent Architecture.

In our design, the role of an agent software switches between two modes- *Cluster head Mode* and *Normal Mode*. However, at any particular time, within each cluster, agent software of only one sensor node works in *Cluster head Mode*, others execute in *Normal Mode*.

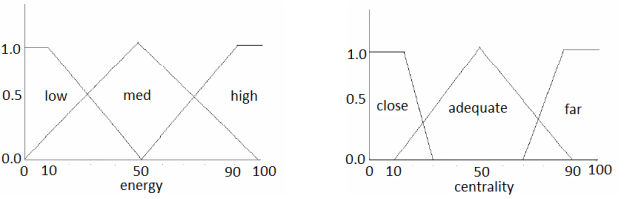
5.1 Scheduler Module (SM)

This module is in charge of scheduling a new round of cluster head selection. It is active in *Cluster head Mode* only. It interacts with *Problem Solver Module* and *Communicator Module*. After periodic interval, it checks the remaining energy of the cluster head node and initiates new cluster head selection if necessary.

5.2 Problem Solver Module (PSM)

This is the basic working module of an agent. Based on current working mode, PSM does two distinct tasks:

Weight Calculation in Normal Mode. We define the weight of a sensor node as a function of two parameters: *Centrality(C)* and *Remaining Energy(RE)*. As sensor nodes are resource-constraint, we use simple Fuzzy Logic based controller in agents to calculate node weight. During *Fuzzification* phase, we followed the mapping presented in [11] to convert crisp values to fuzzy values. Figure 3 portrays this *Fuzzification* process. We define our own rule-base for fuzzy controller in table 1. PSM uses this rule base to determine current weight value in fuzzy variable.



(a) Fuzzy set for fuzzy variable *energy* (b) Fuzzy set for fuzzy variable *centrality*

Fig. 3. Fuzzy Sets used in Fuzzy Controller.

Table 1. Fuzzy Rule Base.

Rule	Centrality	Energy	Result
1	Close	High	High
2	Adequate	High	High
3	Far	High	High
4	Close	Medium	High
5	Adequate	Medium	Medium
6	Far	Medium	Medium
7	Close	Low	Medium
8	Adequate	Low	Low
9	Far	Low	Low

New Cluster Head Selection in Cluster head Mode. PSM of cluster head agent operates on all Fuzzy weight values of normal sensor nodes. After comparing multiple Fuzzy weight values, if some nodes are equally suitable to be the new cluster head, PSM runs following scheme to break a tie:

In order to *Defuzzify*, we assign numeric values against different Fuzzy values *High/Close*=3, *Medium/Adequate*=2, and *Low/Far*=1 and bias the decision by putting 60% importance to *energy*. For example, if two nodes n_i and n_j are in a tie with Fuzzy values {Far, Medium} and {Close, Low} respectively, then PSM computes *Defuzzified* values as d_i = 1*0.4 + 2*0.6 = 1.6 and d_j = 3*0.4 + 1*0.6 = 1.8. Here d_i and d_j are crisp values for node n_i and n_j respectively. Even after this computation, if PSM can not break the tie, it compares node IDs and selects node with minimum ID as next cluster head.

5.3 Communicator Module (CM)

This module is in charge of sensor radio component and communication with other agents in the system. If agent software is running in *Cluster head Mode*, it does two things: (a) Upon receiving request from SM, it sends out *Discovery Message* to other agents; (b) If it receives *Weight Message*(s), it forwards weight value(s) to PSM.

On the other hand, if the agent is working in *Normal Mode*, CM does only one thing: it receives *Weight Message*(s) and forwards to the *Knowledge-base Updater Module*.

5.4 Knowledge-base Updater Module (KBUM)

This is the only module that interacts with agent *Knowledge-base*. KBUM uses a dedicated area of sensor node's memory to maintain the *Knowledge-base*. Any change in shared knowledge like- weight updates or rule updates are propagated to it by PSM and/or CM; and as a result, KBUM synchronizes agent's local memory with global state.

5.5 Knowledge-base (KB)

Knowledge-base in an agent holds a local copy of the shared knowledge. An agent's KB houses a snapshot of a fragment of global information, that it is interested in. KB includes rules for *Fuzzy* controller, latest weights of all neighbor nodes and current cluster head's node ID.

6 System Dynamics

6.1 Protocol Message Types

In a collaborative multi-agent system, message passing is an integral part which facilitates distributed processing. In this section, we present four different message types that agents in our design exchange during system operation.

Discovery Message. Once remaining energy of the current cluster head goes below a threshold value (which is 40% of initial energy in our implementation), cluster head agent broadcasts *Discovery Message*. A *Discovery Message* initiates new cluster head selection within a cluster.

Weight Message. *Weight Message* contains latest weight value of a node. This message is broadcasted by agent in a normal node as a response to an incoming *Discovery Message* from current cluster head.

Control Message. *Control Message* is used to broadcast any change in routing topology. After determining new cluster head, current cluster head agent broadcasts *Control Message* indicating new cluster head node ID.

Data Message. *Data Message* originates from normal node agents and is used to report current data readings to the cluster head.

6.2 System Work-flow

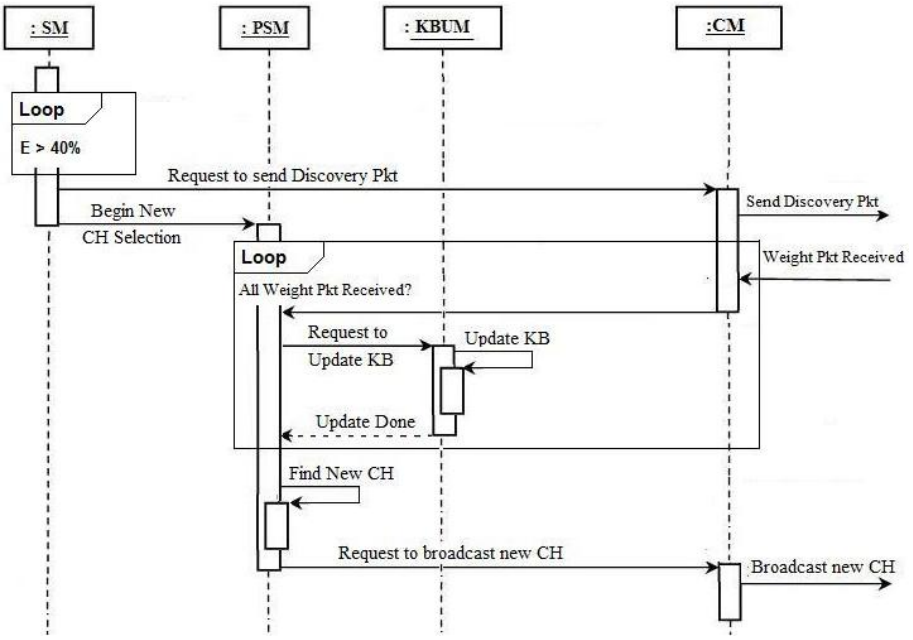


Fig. 4. Sequence diagram for *cluster head mode*.

In this section we explain the sequences of operation in our agent-based system. First, figure 4 shows action sequences for an agent operating in *Cluster head Mode*. After periodic interval, SM measures remaining energy of the cluster head node, and if it is bellow the threshold (*i.e.*, less than 40% of initial energy), SM starts a new cluster head selection round. It notifies PSM that a new cluster head selection process is in effect and also requests CM to send out *Discovery Message*.

CM then sends out *Discovery Message* in the wireless medium, and in response, it receives *Weight Message*s from other agents. CM forwards these node-weights to PSM for calculation. In turn, PSM requests KBUM to update *Knowledge base* with newly received weight values and once PSM receives weights from all nodes, it uses *Rule base* to select new cluster head as explained in section 5.2. After finding the new cluster head, PSM requests CM to convey other nodes about new cluster head, and as a result, CM sends out new *Control Message* containing newly chosen cluster head ID.

Figure 5 portrays action sequence of an agent working in *Normal Mode*. If its CM receives a *Discovery Message* from current cluster head, CM requests PSM to compute

own node weight. Upon computation, PSM returns own weight to CM. CM then encapsulates this weight value in a *Weight Message* and broadcasts that message in the neighborhood. Besides, if CM receives *Weight Message* from a different node, it requests KBUM to update agent's *Knowledge base*. In a similar way, KB update is also performed if an agent receives *Control Message* from current cluster head.

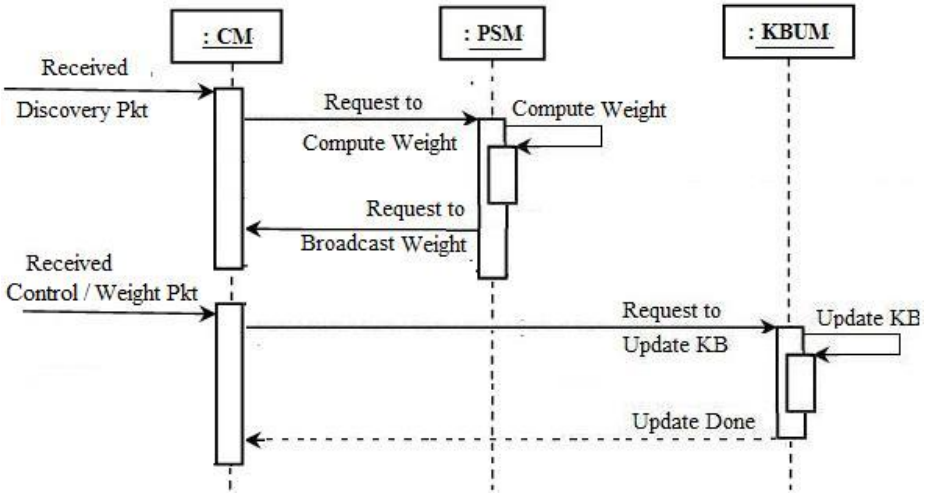


Fig. 5. Sequence diagram for *normal mode*.

7 Conclusions

In this work, we design a system to rotate the role of cluster heads in a Wireless Sensor Network. Multiple agents, each residing in individual sensor node, interact with each other and participate in new cluster head selection. In our design, agent-based architecture provides flexibility to network management. At the same time, as we use fuzzy controller, this design can be easily extended by adding new rules in the rule-base. Our future work focuses on implementing this design in TinyOS operating system for Wireless Sensor nodes.

References

1. W. Heinzelman, A. Chandrakasan and H. Balakrishnan: Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In: 33rd Hawaii International Conference on System Science (2000)
2. Y.-C. Tseng, S.-P. Kuo, H.-W. Lee, and C.-F. Huang: Location tracking in a wireless sensor network by mobile agents and its data fusion strategies. Computer Journal, vol. 47, no. 4, pp. 448-460, (2004)

3. H. Qi, S.S. Iyengar, and K. Chakrabarty: Multi-Resolution Data Integration Using Mobile Agents in Distributed Sensor Networks. IEEE Trans. Systems, Man, and Cybernetics Part C: Applications and Rev., vol. 31, no. 3, pp. 383-391, Aug. (2001)
4. Daniel Massaguer, Chien-Liang Fok, Nalini Venkatasubramanian, Gruia-Catalin Roman, Chenyang Lu: Exploring sensor networks using mobile agents. pp. 323-325, AAMAS (2006)
5. C.-L. Fok, G.-C. Roman, and C. Lu: Mobile agent middleware for sensor networks: An application case study. In: 4th Int. Conf. on Information Processing in Sensor Networks (IPSN'05), pages 382–387. IEEE, April (2005)
6. Hairong Qi, Yingyue Xu, Xiaoling Wang: Mobile-agent-based Collaborative Signal and Information Processing in Sensor Networks. In: Proceeding of the IEEE, Vol. 91, NO. 8, pp.1172-1183, Aug (2003)
7. Min Chen, Taekyoung Kwon, Yong Yuan, Yanghee Choi and Victor C. M. Leung1: Mobile Agent-Based Directed Diffusion in wireless Sensor Networks. EURASIP Journal on Advances in Signal Processing, Article ID 36871, (2007)
8. Shakshuki, E., Xing, X.Y. and Malik, H: Mobile Agent for Efficient Routing among Source Nodes in Wireless Sensor Networks. In: 3rd International Conference on Autonomic and Autonomous Systems (ICAS'07), June(2007)
9. Xiangning Fan and Yulin Song: Improvement on LEACH Protocol of Wireless Sensor Network. International Conference on Sensor Technologies and Applications (SENSORCOMM 2007), pp.260-264, (2007)
10. W. Heinzelman, A. Chandrakasan and H. Balakrishnan: An Application-Specific Protocol Architecture for Wireless Microsensor Networks. IEEE Transactions on Wireless Communications, Vol. 1, No. 4, October(2002)
11. Indranil Gupta, Denis Riordan and Srinivas Sampalli: Cluster-head Election using Fuzzy Logic for Wireless Sensor Networks. In: 3rd Annual Communication Networks and Services Research Conference, pp.255 - 260, (2005)
12. Xiaorong Zhu and Lianfeng Shen: Near optimal cluster-head selection for wireless sensor networks. Journal of Electronics (China), Science Press, co-published with Springer-Verlag GmbH, Vol. 4, pp.721-725, November(2007)
13. M. Negnevitsky: Artificial intelligence: A guide to intelligent systems. Addison-Wesley, Reading, MA (2001)

Evolutionary Optimization of Echo State Networks: Multiple Motor Pattern Learning

André Frank Krause[1,3], Volker Dürr[2,3], Bettina Bläsing[1,3] and Thomas Schack[1,3]

[1] Faculty of Sport Science, Dept. Neurocognition & Action
University of Bielefeld, D-33615 Bielefeld, Germany
{andre_frank.krause, bettina.blaesing,
thomas.schack}@uni-bielefeld.de
[2] Faculty of Biology, Dept. for Biological Cybernetics
University of Bielefeld, D-33615 Bielefeld, Germany
volker.duerr@uni-bielefeld.de
[3] Cognitive Interaction Technology, Center of Excellence
University of Bielefeld, D-33615 Bielefeld, Germany

Abstract. Echo State Networks are a special class of recurrent neural networks, that are well suited for attractor-based learning of motor patterns. Using structural multi-objective optimization, the trade-off between network size and accuracy can be identified. This allows to choose a feasible model capacity for a follow-up full-weight optimization. Both optimization steps can be combined into a nested, hierarchical optimization procedure. It is shown to produce small and efficient networks, that are capable of storing multiple motor patterns in a single net. Especially the smaller networks can interpolate between learned patterns using bifurcation inputs.

1 Introduction

Neural networks are biological plausible models for pattern generation and learning. A straight-forward way to learn motor patterns is to store them in the dynamics of recurrent neuronal networks. For example, Tani [1] argued that this distributed storage of multiple patterns in a single network gives good generalisation compared to local, modular neural network schemes [2]. In [3] it was shown that it is not only possible to combine already stored motor patterns into new ones, but also to establish an implicit functional hierarchy by using leaky integrator neurons with different time constants in a single network. This can then generate and learn sequences by use of stored motor patterns and combine them to form new, complex behaviours. Tani [3] uses backpropagation through time (BPTT, [4]), that is computationally complex and rather biologically implausible. Echo State Networks (ESNs, [5]) are a special kind of reccurent neuronal networks that are very easy and fast to train compared to classic, gradient based training methods. Gradient based learning methods suffer from bifurcations that are often encountered during dynamic behaviour of a network, rendering gradient information invalid [6]. Additionally, it was shown mathematically that it is very difficult to learn long term correlations because of vanishing or exploding gradients [7]. The general idea behind ESNs is to have a large, fixed, random reservoir of recurrently and

sparsely connected neurons. Only a linear readout layer that taps this reservoir needs to be trained. The reservoir transforms usually low-dimensional, but temporally correlated input signals into a rich feature vector of the reservoir's internal activation dynamics.

Typically, the structural parameters of ESNs, for example the reservoir size and connectivity, are choosen manually by experience and task demands. This may lead to suboptimal and unnecessary large reservoir structures for a given problem. Smaller ESNs may be more robust, show better generalisation, be faster to train and computationally more efficient. Here, multi-objective optimization is used to automatically find good network structures and explore the trade-off between network size and network error.

Section 2 describes the ESN equations and implementation. Section 3 introduces the optimization of the network structure and explains how small and effective networks can be identified. Good network structures are further optimized at the weight level in section 4. Section 4.1 shows how to combine structural and weight level optimization into a single, nested algorithm, facilitating a genetic archive of good solutions. In section 5, the dynamic behaviour of the optimized ESNs is shown for different bifurcation inputs.

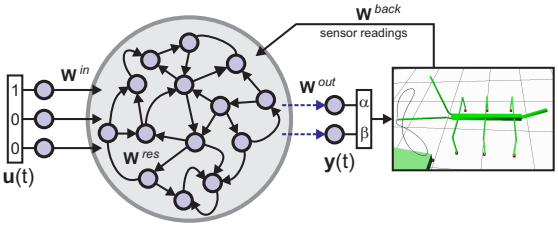


Fig. 1. General structure of an echo state network. Solid arrows indicate fixed, random connections, while dotted arrows are trainable readout connections. The output [????] sets the joint angles of a bi-articular manipulator, e.g., an bio-inspired active tactile sensor. Joint angles are fed back via the backprojection weight matrix $\mathbf{W}^{back}$.

2 Echo State Network

A basic, discrete-time ESN with a sigmoid activation functions was implemented in Matlab©2009b. The purpose of this ESN was to control the joints of a bi-articular manipulator that could serve as a bio-inspired, active tactile sensor. The overall goal was to use the input to the ESN to set the tactile sampling pattern as desired. The state update equations used are:

$$\begin{aligned} \mathbf{y}(n) &= \mathbf{W}^{out}\mathbf{x}(n) \\ \mathbf{x}(n+1) &= \tanh(\mathbf{W}^{res}x(n) + \mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}^{back}\mathbf{y}(n) + \nu(n)) \end{aligned} \tag{1}$$

where $\mathbf{u}$, $\mathbf{x}$ and $\mathbf{y}$ are the activations of the input, reservoir and output neurons, respectively. $\nu(n)$ adds a small amount of uniformly distributed noise to the activation values of the reservoir neurons. This tends to stabilize solutions, especially in models that use output feedback for cyclic attractor learning [8]. $\mathbf{W}^{in}$, $\mathbf{W}^{res}$, $\mathbf{W}^{out}$ and $\mathbf{W}^{back}$ are

the input, reservoir, output and backprojection weight matrices. All matrices are sparse, randomly initialised, and stay fixed, except for $\mathbf{W}^{out}$. The weights of this linear output layer are learned using offline batch training. During training, the teacher data is forced into the network via the back-projection weights (teacher forcing), and internal reservoir activations are collected (state harvesting). After collecting internal states for all training data, the output weights are directly calculated using ridge regression. Ridge regression uses the Wiener-Hopf solution $\mathbf{W}^{out} = \mathbf{R}^{-1}\mathbf{P}$ and adds a regularization term (Tikhonov regularization):

$$\mathbf{W}^{out} = (\mathbf{R} + \alpha^2\mathbf{I})^{-1}\mathbf{P} \tag{2}$$

where α is a small number, $\mathbf{I}$ is the identity matrix, $\mathbf{R} = \mathbf{S}'\mathbf{S}$ is the correlation matrix of the reservoir states and $\mathbf{P} = \mathbf{S}'\mathbf{D}$ is the cross-correlation matrix of the states and the desired outputs. Ridge regression leads to more stable solutions and smaller output weights, compared to ESN training using the Moore-Penrose pseudoinverse. A value of $\alpha = 0.08$ was used for all simulations in this paper.

3 Multi-objective Network Structure Optimization

Multi-objective optimization (MO) is a tool to explore trade-offs between conflicting objectives. In the case of ESN optimization, the size of the reservoir versus the network performance is the main trade-off. In MO, the concept of dominance replaces the concept of a single optimal solution in traditional optimization. A solution dominates another, if strictly one objective value is superior and all other objectives are at least equal to the corresponding objective values of another solution. Following this definition, multiple (possibly infinite) non-dominated solutions can exist, instead of a single optimal solution. The set of non-dominated or pareto-optimal solutions is called the pareto front of the multi-objective problem. The goal of MO is to find a good approximation of the true pareto front, but usually MO algorithms converge to a local pareto front due to complexity of the problem and computational constraints.

Usually, the structural parameters of an ESN are choosen manually by experience and task demands. Here, the full set of free network parameters was optimized using MO. The MO was performed with the function 'gamultiobj' from the *Matlab Genetic Algorithm and Direct Search (GADS) Toolbox*, that implements a variant of the 'Elitist Non-dominated Sorting Genetic Algorithm version II' (NSGA-II algorithm, [9]). The network structure was encoded into the genotype as a seven-dimensional vector of floating point numbers. The first six structural parameters were the sparsity and weight range of the input-, reservoir- and backprojection weights. The seventh parameter was the number of reservoir neurons. The search range of the algorithm was constrained to $[0, 1]$ for the sparsity values, to $[-5, 5]$ for the weight values and to $[1, 100]$ for the reservoir size ($[1, 500]$ for the 4-pattern problem). The optimization was started with a population size of 1000 and converged after around 120 generations. In each iteration of the MO, all genomes were decoded into network structures, the networks were trained and then simulated with random initial activations for 1000 frames per pattern. In order to neglect the initial transient behaviour, the first 50 iterations of network output were rejected. The network output and the training patterns are usually not in-phase. The

best match between training pattern and network output was searched by phase-shifting both output time courses by $\pm$ 50 frames relative to the training pattern and calculating the mean Manhattan distance across all pairs of data points. The training error was then defined as the smallest distance found in that range. The acceptable error threshold (fig.2) is expressed as the percentage of the amplitude of the training patterns, that is 1.0 units for all patterns. The pareto front for a circular pattern (Fig.2a) reveals that even very small networks are capable of learning and generating two sine waves with identical frequency and 90 phase shift. The smallest network found had only 3 reservoir neurons. Including the two output neurons, the overall network size was 5. In comparison, 7 neurons are required for this task when using gradient-based learning methods [10]. Network size increases with the complexity of the motor pattern, and especially when having to store multiple patterns in a single network. Storing 4 patterns in a single network required 166 reservoir neurons to reach an error below 5% (Fig.2d).

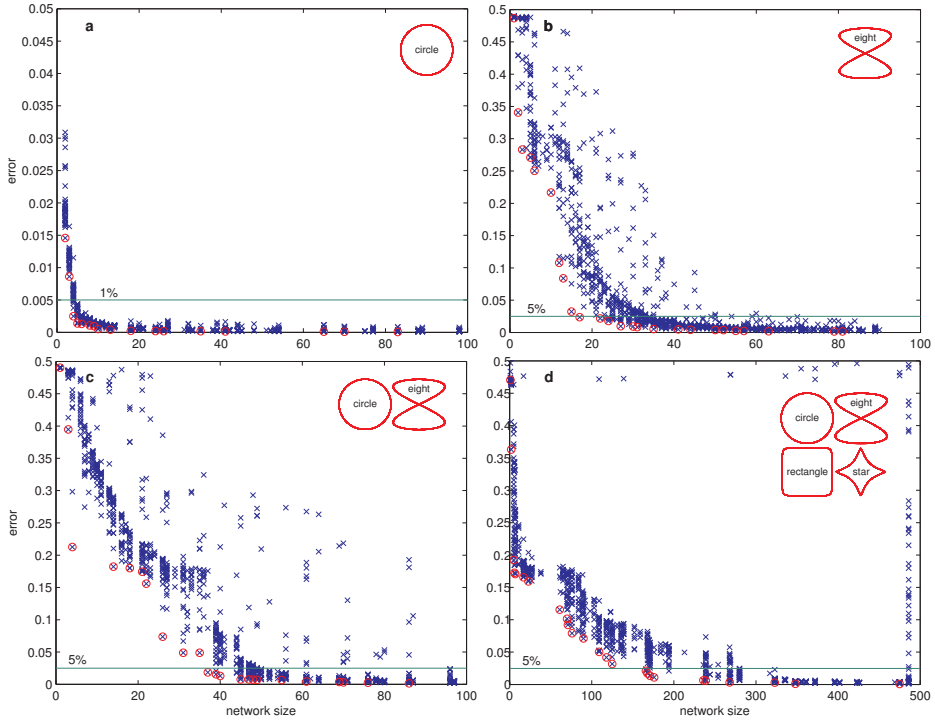


Fig. 2. Minimum reservoir size depends on task complexity. All panels show a set of pareto-optimal solutions (red circles) and the final population (blue crosses). (a) Learning a simple, circular pattern. All networks with 3 or more neurons show an error below 1%. (b) Pareto-front for the figure eight pattern. Learning this pattern requires a notably larger reservoir. Please note the different scaling of the error compared to the easier circle task. Networks with 17 or more neurons have an error below 5%. (c) Storing two motor patterns (circle and figure-eight) as cyclical attractors in a single networkrequires 37 or more reservoir neurons for errors below 5%. (d) Simultaneous learning of four patterns required 166 neurons.

4 Full Optimization of the Network Weights

From the pareto front of the two-pattern task, four candidate network structures were selected and optimized further, using a single-objective genetic algorithm. This time, all network weights except the output layer were fully optimized. The output layer was still trained by ridge regression. An initial random population of 200 parents was created from the network structure information of the selected candidate solutions with 4, 14, 26 and 37 reservoir neurons. Network weights were constrained to $[-5, 5]$ and decoded from the genome with a threshold function that preserves sparsity. The threshold function sets a weight to zero, if the genome value is between -1 and 1, see fig.3.

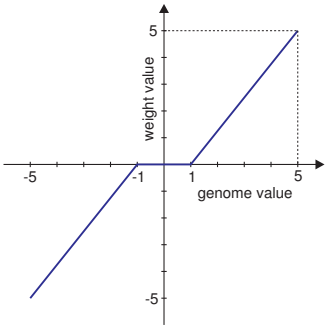


Fig. 3. Threshold function that decodes genome values into weight values, preserving sparse weight coding.

The Genetic Algorithm (GA) options were set to ranked roulette wheel selection, 20 elitist solutions, 80% crossover probability with scattered crossover and self adaptive mutation. Other options were left at their default values (see GADS toolbox, Matlab2009b). The GA-optimization was repeated 20 times for each network size. Fig. 4a shows the improvement in performance compared to the MO structure optimization run. A small network with only 14 reservoir neurons could reproduce the learned patterns with an error of 2.3%. Weight range and connectivity after optimization was analysed with an unpaired Wilcoxon rank sum test. Significant differences in connectivity and weight range were found (Fig. 4b) with a clear trend for smaller reservoir weights and less reservoir connectivity with increasing network sizes. Both input- and backprojection weights tend to increase with reservoir size (Fig. 4a). Although standard ESNs usually have full connectivity for input- and backprojection weights, evolutionary optimization seems to favor sparse connectivity for smaller networks, when given the choice (Fig. 4b).

4.1 Hierarchical Evolutionary Optimization

In the previous section, individual solutions of the MO structural evolution were selected and optimized further on the weight level, using a GA. Both steps can be combined by performing a full-weight GA optimization for each iteration of the MO. This

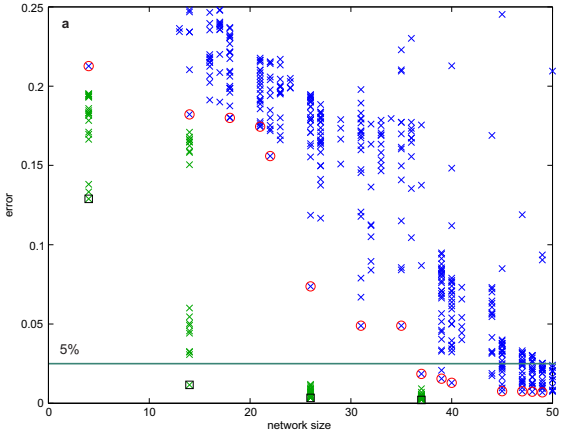


Fig. 4. Subsequent full-weight matrix optimization improves performance. Additional optimization of the four best networks of the two-pattern task with a reservoir size of 4, 14, 26 and 37 neurons. Starting from the best multi-objective solution, 20 GA runs were performed. a) Green crosses indicate the best fitness values of each run. Black squares indicate the overall best solutions that were found.

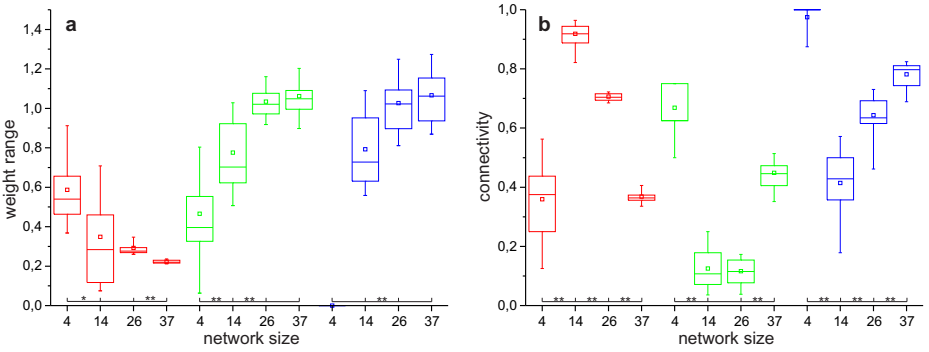


Fig. 5. Optimal weight range and connectivity depends on reservoir size. Network structure after full-weight optimization of the selected networks from fig.4. a) Weight range of all non-zero weights of the reservoir (red), the backprojection weights (green) and the input weights (blue). b) Connectivity (percentage of non-zero weights). Boxplots show 5%, 25%, 50%, 75% and 95% quantiles of N=20 datapoints. * p ¡ 0.05; ** p ¡ 0.01.

way, the pareto front improves by moving closer towards the origin of both optimization objectives. This nested, hierarchical optimization is computationally demanding. To speed up the convergence of the MO, good solutions of the full-weight GA are stored in an archive, keeping each iteration of the MO accessible. In subsequent iterations, the archived genome having the closest structure is injected into the new population of the full-weight GA. Good networks can emerge faster by facilitating cross-over with the archived solutions. This way, the full-weight optimization does not need to start from scratch in each iteration. See Fig.6 for hierarchical optimization of the two-pattern task. The MO had a population size of 200, running - at each iteration - a full-weight opti-

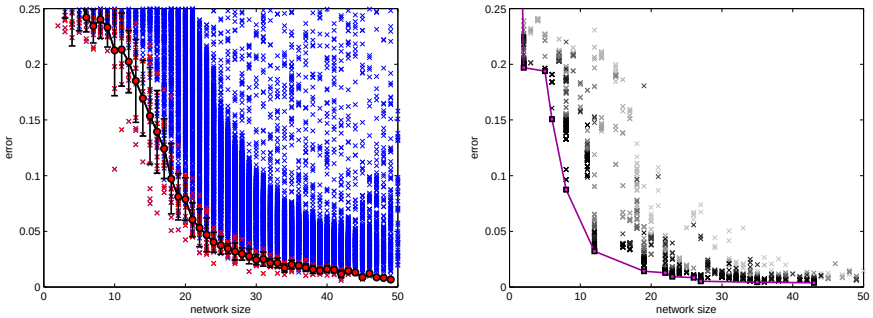


Fig. 6. Left graph: Average pareto front from N=30 repetitions of the strucural MO. Blue crosses show the final populations, red crosses show the pareto fronts, and the red circles show the mean and standard deviation of the pareto-optimal solutions for each network size. Right graph: Hierarchically nesting a full-weight GA optimization into the MO optimization gives a more accurate approximation of the true pareto front, as compared to structural MO alone. The plot shows a single run of the nested MO-GA optimization over 25 generations. Crosses show the population at each generation in grey levels ranging from light grey (first generation) to black (last generation). A single run outperforms the best solutions found in 30 runs of the structural MO, see Fig.7.

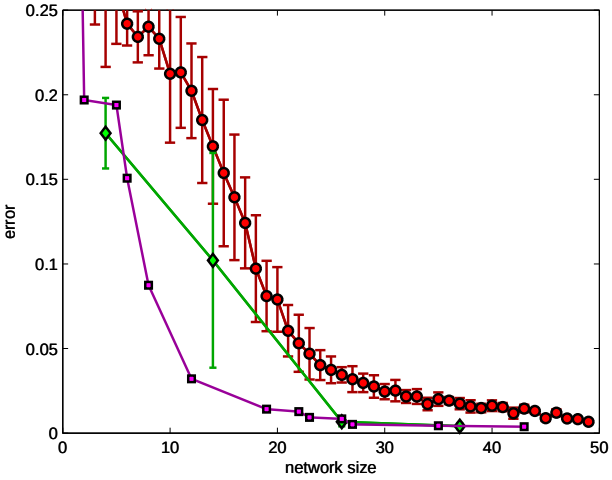


Fig. 7. Comparison of the different optimization runs. The structural MO is shown in red (circles), full-weight optimization of selected solutions from the structural MO in green (diamonds), and the hierarchical optimization in magenta (squares). A single run of the nested, hierarchical optimization shows almost the same performance as the full-weight optimization from section 4.

mization with a population size of 20 individuals for 50 generations. Fig. 7 compares the pareto fronts of the different optimization strategies. A single run of the nested optimization algorithm achieves almost the same result as the combination of structural and subsequent full-weight optimization.

5 Dynamic Network Behaviour

Most of the smaller networks show an unexpected behaviour. They are able to interpolate between the learned patterns, generating novel, not explicitly trained outputs. Fig. 8 shows the dynamical responses from the fittest networks of section 4.1. The first input value was changed gradually in 15 steps from 1.0 to 0.0, while the second input was changed from 0.0 to 1.0. A gradual morphing from the circular to the figure-eight pattern can be observed. It is surprising, that already a small ESN with six reservoir neurons can store two different patterns. Larger networks tend to converge to fixed points for input values other than the trained ones. This interpolation effect might be applied to complex and smooth behaviour generation for neural network controlled robots.

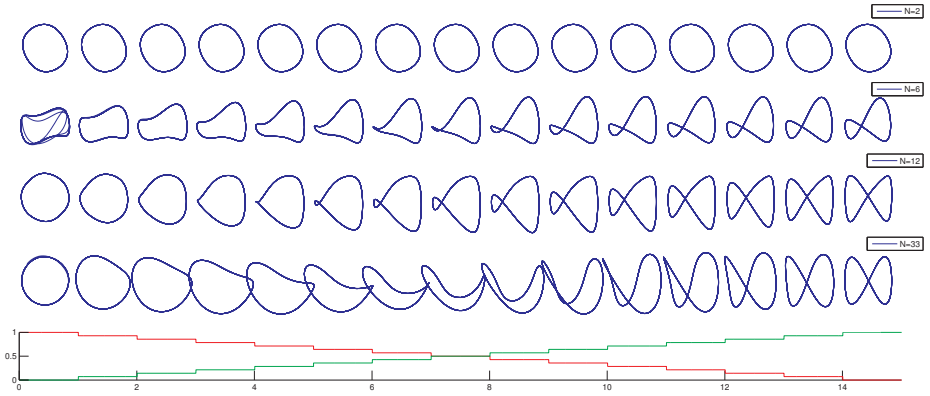


Fig. 8. Dynamic behaviour of selected networks with different reservoir sizes (blue trajectories). Shifting the dynamics of the networks by gradually changing the first input value (red) from 1.0 to 0.0 and the second input (green) from 0.0 to 1.0 in 15 steps. Changing the input to the network causes a slow morphing between the two learned patterns, allowing to generate new patterns that were not explicitly trained. Especially the small networks keep stable with no chaotic regions. Larger networks tend to converge to fixed points for input values other than zero or one.

6 Conclusions

Using MO, good candidate network structures can be selected as starting points for a followup whole-network optimization and fine-tuning using genetic algorithms. Both steps can be combined into a nested, hierarchical multi-objective optimization. The resulting pareto front helps to identify small and sufficiently efficient networks that are able to store multiple motor patterns in a single network. This distributed storage of motor behaviours as attractor states in a single net is in contrast to earlier, local module based approaches. *"If sequences contain similarities and overlap, however, a conflict arises in such earlier models between generalization and segmentation, induced by this separated modular structure."* [3]. By choosing a feasible model capacity, overfitting and the risk of unwanted - possibly chaotic - attractor states is reduced. Also, with the right choice of the network size, an interesting pattern interpolation effect can

be evoked. Instead of using a classic genetic algorithm for fine-tuning of the network weights, new, very fast and powerful black box optimisation algorithms [11] [12] could further increase network performance and allow to find even smaller networks for better generalisation. ESNs can be used for direct control tasks (see [13]) and scale well with a high number of training patterns and motor outputs [14]. A more complex simulation, for example of a humanoid robot, will show if direct, attractor-based storage of parameterized motor patterns is flexible enough for complex behaviour generation.

References

1. Tani, J., Itob, M., Sugitaa, Y.: Self-organization of distributedly represented multiple behavior schemata in a mirror system: reviews of robot experiments using rnnpb. Neural Networks 17 (2004) 1273 – 1289
2. Haruno, M., Wolpert, D. M., Kawato, M.: Mosaic model for sensorimotor learning and control. Neural Computation 13(10) (2001) 2201–2220
3. Yamashita, Y., Tani, J.: Emergence of functional hierarchy in a multiple timescale neural network model: A humanoid robot experiment. PLoS Computational Biology 4 (11) (2008)
4. Werbos, P.: Backpropagation through time: what it does and how to do it. In: Proceedings of the IEEE. Volume 78(10). (1990) 1550–1560
5. Jäger, H., Haas, H.: Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. Science 304 (2004) 78 – 80
6. Jaeger, H.: Tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the "'echo state network"' approach. Technical Report GMD Report 159, German National Research Center for Information Technology (2002)
7. Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J.: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer, J. F. K., ed.: A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press (2001)
8. Jaeger, H., Lukosevicius, M., Popovici, D., Siewert, U.: Optimization and applications of echo state networks with leaky integrator neurons. Neural Networks 20(3) (2007) 335–352
9. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation 6, No. 2 (2002) 182–197
10. Pearlmutter, B. A.: Learning state space trajectories in recurrent neural networks. Neural Computation 1 (1989) 263–269
11. Kramer, O.: Fast blackbox optimization: Iterated local search and the strategy of powell. In: The 2009 International Conference on Genetic and Evolutionary Methods (GEM'09). (2009) in press.
12. Vrugt, J. A., Robinson, B. A., Hyman, J. M.: Self-adaptive multimethod search for global optimization in real-parameter spaces. Evolutionary Computation, IEEE Transactions on 13(2) (2008) 243–259
13. Krause, A. F., Bläsing, B., Dürr, V., Schack, T.: Direct Control of an Active Tactile Sensor Using Echo State Networks. In: Human Centered Robot Systems. Cognition, Interaction, Technology. Volume 6 of Cognitive Systems Monographs. Berlin Heidelberg: Springer-Verlag (2009) 11–21
14. Jäger, H.: Generating exponentially many periodic attractors with linearly growing echo state networks. technical report 3, IUB (2006)

Application of Self-organizing Maps in Functional Magnetic Resonance Imaging

Anderson Campelo[1], Valcir Farias[1], Marcus Rocha[1], Heliton Tavares[1] and Antonio Pereira[2]

[1] Programa de Pós-Graduação em Matemática e Estatística
Universidade Federal do Pará, Belém, Brazil
campelo.ufpa@gmail.com, {valcir, mrocha, helinton}@ufpa.br
[2] Univesidade Federal do Rio Grande do Norte, Natal, Brazil
squareshorts@gmail.com

Abstract. In the present work, we used Kohonen's self-organizing map algorithm (SOM) to analyze functional magnetic resonance imaging (fMRI) data. As a first step to increase computational efficiency in data handling by the SOM algorithm, we performed an entropy analysis on the input dataset. The resulting map allowed us to define the pattern of active voxels correlated with auditory stimulation in the data matrix. The validity of the algorithm was tested using both real and simulated data.

1 Introduction

Functional magnetic resonance imaging (fMRI) is a non-invasive tool widely used for studying the human brain in action. The fMRI has been applied to cognitive studies and also in a clinical setting to monitor tumour growth, pre-surgical mapping, and to diagnose epilepsy, Alzheimer's disease, etc [1]. The fMRI measurements are based on blood-oxygen-level-dependent correlations (BOLD) [2],[3], with hemoglobin being used as endogenous contrast agent, due to the magnetic properties of oxy-hemoglobin (diamagnetic) and deoxy-hemoglobin (paramagnetic) [4].

The BOLD signal was obtained using two experimental paradigms. The first used a blocked design, with the subject being exposed to alternating periods of stimulation and rest. The event-related paradigm, on the other hand, required that the subject performs a simple task, intercalated by long resting intervals.

A fMRI dataset consists of images in 3D space $(x \times y \times z)$, with each image point, named a voxel, changing a long time (t). Most fMRI analysis try to identify how signal related to voxels in a region of interest (ROI) vary in time and to find out whether these variations are somehow correlated with the stimulus. This analysis, however, is a computational challenge due to the low signal to noise ratio in the BOLD response and the usually large amount of data that needs to be processed. Many analytical methods have been developed to deal with this complexity, some of them were created earlier to analyze positron emission topography-generated signals (PET).

Most methods available in the literature use statistical techniques to identify active regions, including *Student*'s t test [5], crossed correlation [6], and the general linear

model (GLM) [7]. These methods are based on the standard hemodynamic function, which models the BOLD response in the brain. Other popular methods are independent component analysis (ICA) [8], [9] and principal component analysis (PCA) [10].

Clustering techniques have also been used successfully, including K-means, fuzzy cluster and hierarchical clustering. Clustering techniques are based on the similarity observed in voxel's time series. The present study uses the Kohonen's self-organizing maps algorithm (SOM) to analyse fMRI data. The SOM [11] is a type of clustering technique which transforms a signal input pattern of an arbitrary dimension into a discrete map and implements transformations in a topologically organized way.

2 Material and Methods

2.1 Simulated Data

We simulated the fMRI experiment (64×64) depicted in Figure 1 with 120 slices by convoluting a block-like stimulus function with the canonical hemodynamic response function generated as a sum of two distribution functions [12]:

$$h(t) = \left(\frac{t}{d}\right)^{a} \exp\left(\frac{-(t-d_1)}{b}\right) - c\left(\frac{t}{d'}\right)^{a'} \exp\left(\frac{-(t-d')}{b'}\right), \tag{1}$$

with $a = 6$, $a' = 12$, $b = b' = 0.9$ and $c = 0.35$, having been determined experimentally by Glover in 1999 with auditory stimulation [13].

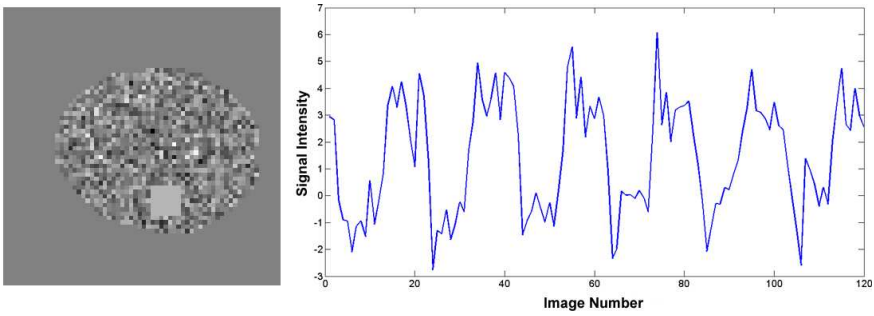


Fig. 1. Diagram showing the spatial distribution of active voxels and their intensity along time in simulated data.

The active area corresponds to 49 voxels, while 1349 voxels corresponded to the remaining grey matter. The other 2698 voxels corresponded to the background and are not time modulated. We added uniform Gaussian noise to reach a SNR of 2dB, which was calculated with the following expression:

$$\text{SNR} = 10 \log\left(\frac{\sigma_S^2}{\sigma_R^2}\right), \tag{2}$$

where $\sigma_S^2 2$ and σ_R^2 are signal and noise variances, respectively.

2.2 Real Data

The fMRI experiment used a 1.5 T Siemens scanner (Magnetom Vision, Erlangen, Germany), with the following parameters for EPI (echo-planar imaging) sequences: TE = 60 ms, TR = 4.6 s, FA = 90, FOV = 220 mm, and slice thickness of 6.25 mm. 64 cerebral volumes with 16 slices each were acquired with a matrix dimension of 128×128.

During the experimental procedure the subject received auditory stimulation in a blocked design, with 5 stimulation blocks (27.5s each) intercalated with 6 resting blocks (27.5 s each). During the task, the subject listened passively to a complex story with a standard narrative structure. After, the test the subject had to inform to the experimenter its comprehension of the story content.

Acquired images were preprocessed with the software SPM8 (Statistical Parametric Mapping) in order to increase the signal-to-noise ratio (SNR) and to eliminate incident noise associated with the hardware, involuntary movements of the head, cardiac and respiratory rhythms, etc.

2.3 Self-organizing Maps

fMRI data was analyzed with Kohonen's SOMs using an implementation available in the literature (see [14], [15], [16], [17], [18], [19]). Kohonen's SOM is an artificial neural network where neurons are disposed as a uni- or bi-dimensional grid layout. In a bi-dimensional layout, geometriy is free and can be rectangular, hexagonal, triangular etc. In a SOM, each neuron in a grid is represented by a probability distribution function of the input data.

The SOM algorithm responsible for map formation begins initializing the grid neurons weights with random values, which can be obtained from the input data. In the present work we used a bidimensional grid of dimension 10×10 $(i = 100)$ [19]. Each neuron in the grid is connected to every element of the input dataset, i.e., the dimension of weights $\mathbf{m}_i$ is the same as the input dataset:

$$\mathbf{m}_i = [m_{i1}, m_{i2}, \ldots, m_{in}]^T \in \Re, \tag{3}$$

where n indicates the total amount of points available in the time series generated by the fMRI experiment.

After each iteration t of the ANN, we selected randomly a vector from the input dataset, given by:

$$\mathbf{x}_i = [x_1, x_2, \ldots, x_n]^T \in \Re, \tag{4}$$

which indicates the time series of a given voxel from the fMRI dataset.

Then, $\mathbf{x}$ is compared to weights $\mathbf{m}_i$ in the grid using the minimun euclidean distance as criteriom for choosing a winner in the ANN [15],[19]. Since the correlation distance metric, however, seems to be a better method to discern similarities than conventional Euclidean distance [16], the winner neuron is selected by:

$$\mathbf{m}_c = \arg\max_i \{\text{corr}(\mathbf{x}(t), \mathbf{m}_i(t))\}, \tag{5}$$

with $i = 1 \ldots, M$ where M is the total number of neurons in the grid, $\mathbf{m}_c(t)$ represents the time series of the winner c and $\text{corr}(\mathbf{x}(t), \mathbf{m}_i(t))$ is the correlation coefficient between $\mathbf{x}(t)$ and $\mathbf{m}_i(t)$.

The updating of the weight vector $\mathbf{m}(t+1)$ in time $t+1$, with $t = 0, 1, 2, \ldots$ is defined by:

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + h_{ci}(t)[\mathbf{x}(t) - \mathbf{m}_i(t)], \tag{6}$$

which is applied to every neuron on the grid that is within the topological neighborhood-kernel h_{ci} from the winner. Thus, Equation (6) has the goal of approximating the weight vector $\mathbf{m}_i$ of neuron i towards the input vector, following the degree of interation h_{ci}. This approach transforms the grid, after training, in a topologically organized characteristic map, in the sense that adjacent neurons tend to have similar weights.

A function frequently used to represent the topological neighborhood-kernel h_{ci} is the Gaussian function, which is defined by:

$$h_{ci}(t) = \alpha(t) \exp\left\{\frac{-\|\mathbf{r}_c - \mathbf{r}_i\|}{2\sigma^2(t)}\right\}, \tag{7}$$

where $\alpha(t)$ is the learning rate, which has to gradually decrease along time to avoid that new data gathered after a long training session could compromise the knowledge already sedimented in the ANN; $\mathbf{r}_c$ and $\mathbf{r}_i$ determine the discrete position of neurons c and i in the grid; and $\sigma(t)$ defines the full-width at half-maximum (FWHM) of the Gaussian kernel. Parameters $\sigma(t)$ and $\alpha(t)$ gradually decrease by t/τ (τ is a time constant) after each iteration t, following an exponential decay.

2.4 Evaluating the SOM Quality

There are several mechanisms that can evaluate the quality of the generated map obtained after the learning process. In the present work we used the quantization error:

$$E_q = \frac{1}{N} \sum \|\mathbf{x} - \mathbf{m}_c\|^2. \tag{8}$$

The quantization error is defined as the mean error corresponding to the difference between each characteristic vector $\mathbf{x}$ and the winner neuron $\mathbf{m}_c$, where N is the total number of patterns.

2.5 Analysis of Entropy

Some authors recommend the ad-hoc reduction of voxel samples to optimize the algorithm implementation [20]. Thus, in order to improve analitical efficiency, only signals originating from the brain were actually processed. Besides, we performed an entropy analysis to each voxel of the characteristics set and eliminated all voxels with an entropy level below an empirically determined threshold. The Shannon's entropy, as well as other techniques based on Information Theory, has proved to be satisfactory in fMRI experiments [21].

The Shannon entropy of a random variable $\mathbf{X}$ with probability vector $(p_1, \ldots, p_n)$ is defined by:

$$H(\mathbf{X}) = -\sum_{i=1}^{n} p_i \cdot \log_2 p_i, \tag{9}$$

with $H(\mathbf{X})$ being the entropy of variable $\mathbf{X}$. The Shannon entropy [22] measures the uncertainty present in any dataset and allows the comparison of its properties with other datasets of similar dimensions, by representing the amount of information contained in each as a probabilistic event.

To calculate the entropy, the time series of each voxel is divided into two levels of intensity, then is calculated the probabilities of levels of intensity from the amount of time points at each level. Finally, the entropy of each time series is calculated according to Equation (9). The entropy of signals corresponding to non-active voxels tends to have low value because of an irregular configuration of the signal. On the other hand, the signal of a probable active voxel tends to present a high value of entropy, which is associated with a wide distribution of probability.

3 Results

In this work, the configuration parameters of the SOM were initialized according to previous studies [16],[19], both in real and simulated data. In Equation (7), the learning rate was initialized as $\alpha(0) = 0.1$ and the parameter effective width as $\sigma(t) = 7$. The number of SOM iterations was regulated dinamically according to error stabilization (Eq. 8).

3.1 Simulated Data

First, we calculate Shannon's entropy for each voxel of the simulated data. The values of entropy for the 1398 voxels contained in the interior of the artificial brain had varied of 0.4949 to 1, where the 49 voxels with activation signal had presented one high value of entropy (Figure 2a). After that, about 6% of 1398 voxels were eliminated from the input data of SOM, these voxels had presented a value of entropy $H < 0.85$ (Figure 2b).

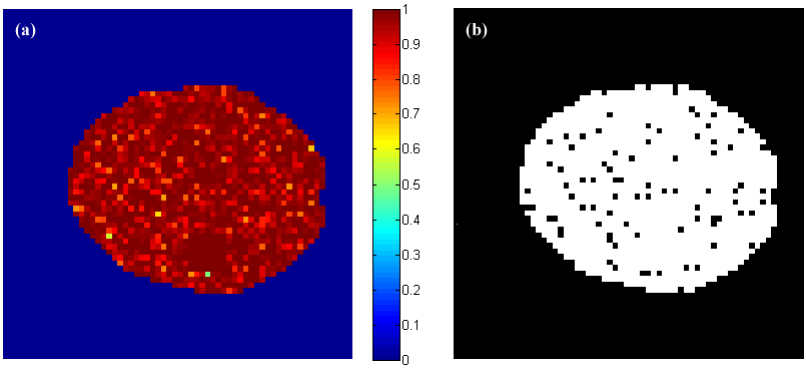


Fig. 2. (a) Entropic map of the artificial data; (b) Entropic analysis of synthetic data. The dark dots in the image were eliminated, the equivalent of 6% of the input data.

From the final conformation of the neuronal grid achieved after 100 algorithm iterations (Figure 3), we can observe that voxels with similar temporal patterns are clustered together in the SOM.

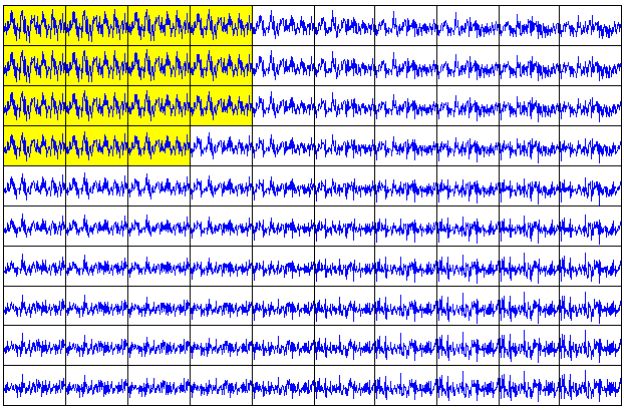


Fig. 3. 10×10 grid of neurons after implementation of the SOM algorithm, the cluster of neurons in yellow match the patterns of activity.

For better visualization of these clusters, there are several clustering methods that can be used, such as K-means [23], fuzzy logic [24] and correlation as a measure of similarity in a hierarchical clustering [16]. We use this last reference utilizing a simple correlation as a measure of similarity between neurons in the grid.

Figure 4 shows the active regions defined using the average signal from the neurons demarcated in the Figure 3. A correlation coefficient (CC) was determined between this average and each voxel in a fMRI dataset, showing only those with CC > 0.7056.

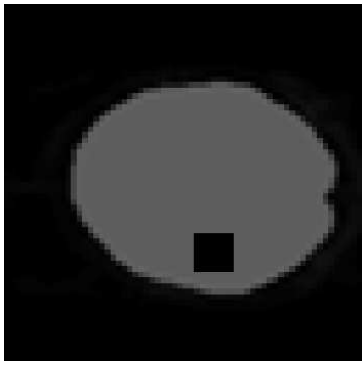


Fig. 4. The dark regions in the brain correspond to active regions as defined by the SOM after 100 iterations.

3.2 Real Data

The same analytical procedure used for simulated data was used to deal with real data. However, we adopted the quantization error (Eq. 8) to estimate the amount of steps of the algorithm and also act as quality controller of learning. Figure 5 shows the evolution of error to each 10 iterations, using normalized data. But even if the error has begun to stabilize at around 100 repetitions, the training is continued until 250 iterations in order to perform a fine tuning of the map features and thus produce a statistically accurate quantization of the input space. Analyzing the same figure, it is possible verify that the magnitude of error for the case where it was applied to analyze the entropic prior to SOM (Fig. 5a) presents lower, also have begun to stabilize somewhat earlier than the case where not used the Shannon's entropy (Fig. 5b).

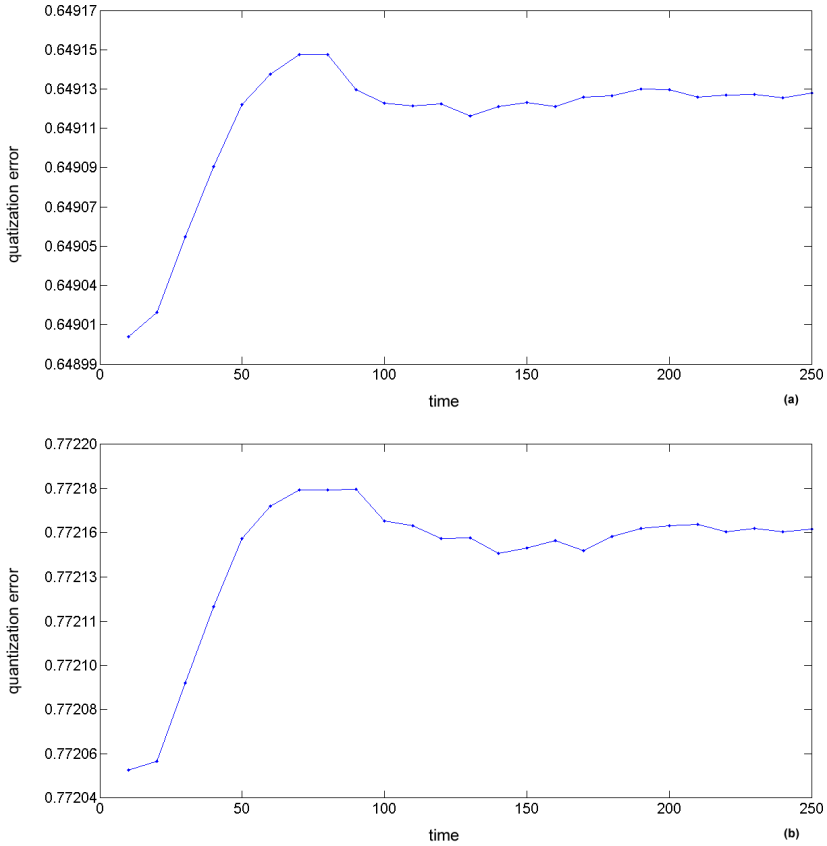


Fig. 5. Graph of the quantization error for a total of 250 iterations. (a) quantization error with the application of entropy; (b) quantization error without entropy.

Figure 6 reveals the active voxels, according to our method, in the eighth slice of fMRI data CC > 0.6. In it you can see two main regions as a result of the auditory task located in the temporal lobe.

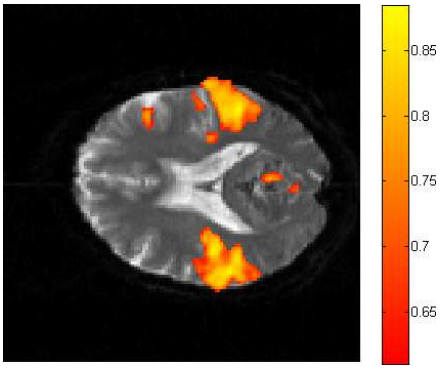


Fig. 6. Active regions correlated with the auditory stimulation defined after 250 iterations with the SOM.

4 Conclusions

The Kohonen's self-organizing map was applied in data of functional magnetic resonance in synthetic and real models, this last one representing an auditory experiment with the paradigm in block. With the purpose of increasing the efficiency of the analysis method was proposed to Shannon's entropy, which eliminated a range of $5 - 10\%$ of the set of input data. The configuration of the data after the entropy analysis allowed more likely to find groups of neurons active in the SOM grid with a smaller number of iterations. Moreover, in the temporal evolution of the quantization error of the SOM, it can be verified that entropy analysis decreased the amplitude of this error and admitted his slightly faster stabilization. The results of SOM, both for simulated data, as for real data, reaffirmed that it can be used as a tool for interpretation of fMRI data. And it has the advantage that the shaped of the hemodynamic response is not considered, that is, a HRF modeled mathematically is not used.

References

1. Fontoura, D. R., Ans, M., Costa, J. C., Portuguez, M. W.: Identifying language cerebral functions: a study of functional magnetic resonance imaging in patients with refractory temporal lobe epilepsy. Journal of Epilepsy and Clinical Neurophysiology, 14(1) (2008) 7-10.
2. Ogawa, S., Lee, T. M., Kay, A. R., and Tank, D. W.: Brain magnetic resonance imaging with contrast dependent on blood oxygenation. Proceedings of the National Academy of Sciences of USA, 87 (1990) 9868-9872.
3. Thulborn, K, R., Waterton, J. C., Matthews, P. M., and Radda, G. K.: Oxygenation dependence of the tranverse relaxation time of water protons in whole blood at high field. Biochimica et Biophysica Acta, 714 (1982) 265-270.
4. Pauling, L. and Coryell, C.: The magnetic properties and structure of hemoglobin, oxyhemoglobin, and carbon monoxyhemoglobin. Proceedings of the National Academy of Sciences of USA, 22 (1936) 210-216.
5. Huettel S. A., Song A. W., and McCarthy G.: Functional Magnetic Resonance Imaging. Sinauer Associates, 1st edition (2004).

6. Rabe-Hesketh, S., Bullmore, E., and Brammer, M.: The analysis of functional magnetic resonance images. Stat. Methods Med. Res. 6 (1997) 215-237.
7. Friston, K. J., Holmes, A. P., Worsley, K. J., Poline, J. P., Frith, C. D., and Frackowiak, R. S. J.: Statistical parametric maps in functional imaging: A general linear approach. Hum. Brain Mapping 2 (1995) 189-210.
8. McKeown, M.J., Makeig, S., Brown, G. G., Jung, T. P., Kindermann, S. S., Bell, A. J., and Sejnowski, T. J.: Analysis of fMRI data by blind separation into independent spatial components. Human Brain Mapping, 6 (1998) 160-168.
9. Biswal, B. B., and Ulmer J. L.: Blind source separation of multiple signal sources of fMRI data sets using independent component analysis. Journal of Computer Assisted Tomography, 23 (1999) 265-271.k
10. Friston, K. J., Frith, C. D., Liddle, P. F., and Frackowiak, R. S. J.: Functional connectivity: the principal component analysis of large (PET) datasets. Journal of Cerebral Blood Flow and Metabolism, 13 (1993) 5-14.
11. Kohonen T.: Self-organized formation of topologically correct feature maps. Biol. Cybernet. Vol. 43 (1982) 59-69.
12. Friston, K. J., Fletcher, P., Josephs, O. Holmes, A., Rugg, M. D., and Turner, R.: Event-related fMRI: Characterizing differential responses. NeuroImage, 7 (1998) 30-40.
13. Glover, G. H.: Deconvolution of impulse response in event-related BOLD fMRI. NeuroImage, 9 (1999) 416-429.
14. Chuang K. H., Chiu M. J., Lin C. C., and Chen J. H.: Model-Free functional MRI analysis using Kohonen clustering neural network and fuzzy C-means. IEEE Transactions on Medical Imaging, Vol. 18 (1999) 1117-1128.
15. Fischer, H. and Hennig J.: Neural network-nased analysis of MR time series. Magnetic Resonance in Medicine, Vol. 41 (1999) 124-131.
16. Liao, W., Chen, H., Yang, Q., Lei, X.: Analysis of fMRI Data Using Improved Self-Organizing Mapping and Spatio-Temporal Metric Hierarchical Clustering. IEEE Transactions on Medical Imaging, Vol. 27 (2008) 1472-1483.
17. Ngan, S. C. and Hu X.: Analysis of functional magnetic resonance imaging data using self-organizing mapping with spatial connectivity. Magnetic Resonance in Medicine, vol. 41 (1999) 939-946.
18. Ngan, S. C., Yacoub E. S., Auffernann W. F, and Hu X.: Node merging in Kohonen's self-organizing mapping of fMRI data. Artificial Intelligence in Medicine, vol. 25 (2001) 19-33.
19. Peltier, S. J., Polk T. A., Noll D.C.: Detecting low-frequency functional connectivity in fMRI using a self-organizing map (SOM) algorithm. Hum. Brain Mapp., vol. 20 (2003) 220-226
20. Gibbons, R. D., Lazar, N. A., Bhaumik, D. K., Sclove, S. L., Chen, H. Y., Thulborn, K. R., Sweeney, J. A., Hur, K., and Patterson, D.: Estimation and classification of fMRI hemodynamic response patterns. NeuroImage, 22 (2004) 804-814.
21. de Araujo, D. B., Tedeschi, W., Santos, A. C., Elias, J. Jr., Neves, U. P., Baffa, O.: Shannon entropy applied to the analysis of event-related fMRI time series. Neuroimage, 20(1) (2003) 311-317.
22. Shannon, C. E.: A mathematical theory of communication. Bell system technical journal 27 (1948) 379-423, 623-656.
23. Goutte, C., Toft, P., Rostrup, E., Nielsen F., A., and Hansen, L. K.: On clustering fMRI time series. NeuroImage, Vol. 9 (1999) 298-310.
24. Wismuller A., Meyer-Base, A., Lange O., Auer D., Reiser, M. F., and Sumners, DeWitt.: Model-free functional MRI analysis based on unsupervised clustering. Journal of Biomedical Informatics 37, (2004) 10-18.

Neural Networks with AR Model Coefficients Applied to the EMG Signal Classification

Marek Kurzynski and Andrzej Wolczowski

Wroclaw University of Technology, Dept. of Systems and Computer Networks
Wyb. Wyspianskiego 27, 50-370 Wroclaw, Poland
{path, marek.kurzynski}@pwr.wroc.pl

Abstract. The paper presents a concept of hand movements recognition on the basis of EMG signal analysis. Signal features are represented by coefficient of autoregressive (AR) model, and as classifier the MLP and Adaline networks are applied. The performance of the proposed method was experimentally compared against four different classifiers using real datasets. The systems developed achieved the highest overall classification accuracies demonstrating the potential of neural network classifiers based on AR coefficients for recognition of EMG signals.

1 Introduction

The activity of human organism is reflected in characteristic biosignals, which can be measured and next can be applied to the control of the work of technical devices. Electrical potentials accompanying skeleton muscles (called EMG signals) are an example of such biosignals. They can be detected and registered through the skin and used to the control of bio-prosthesis.

Although in the last decade many attempts have been made to determine the hand movements on the base of EMG signal analysis ([5, 10, 11, 13]), the reliable recognition of kind of grasp is still a hard problem. The difficulty increases along with the prosthesis dexterity (prosthesis movement repertoire), therefore it is still a need for research in developing EMG signal recognition.

The paper presents a concept of recognition of hand movements (type of grasp) on the base of EMG signal analysis. Signal features are represented by autoregressive (AR) model coefficients, and as classifier the MLP and Adaline network are applied.

The performances of proposed classification systems were compared against four (statistical (Bayes, kernel), fuzzy and k-nearest neighbours) classifiers using real datasets. For the purpose of experimental investigations a special measurement stand was elaborated which allow us synchronous recording the image of the moving hand and multichannel registration of EMG signals.

The paper is divided into three sections and organized as follows. In section 2 we provide an insight into the analysis of EMG signals which is the basis for the recognition of grasps. In Section 3 computer experiments on real data are described and their results are discussed.

2 EMG Signal Analysis

The recognition of hand movement on the basis of the myopotentials comprises three stages [13]: (1) the acquisition of the EMG signal; (2) extraction of the features differentiating the movements; (3) classification of the signal.

Each stage has an influence on the quality of the whole process, i.e. reliability of the grasping movement recognition.

2.1 EMG Signal Acquisition – the Measurement Stand

The block diagram of the designed measurement stand for EMG signal acquisition and identification of the relation between the hand movement and simultaneously created myopotentials, is presented in Fig. 1. The stand includes: (1) a video camera for recording the image of the moving hand; (2) specially designed 8-channel EMG signals measuring circuit (Bagnoli Desktop EMG System, DelSys); (3) the PC computer recording the results of the acquisition, equipped with high fidelity measurement board, containing 8 independent A/D converters (24 bits per channel) and USB port for USB video camera; (4) an application for synchronous recording of the video and EMG data streams and their analysis.

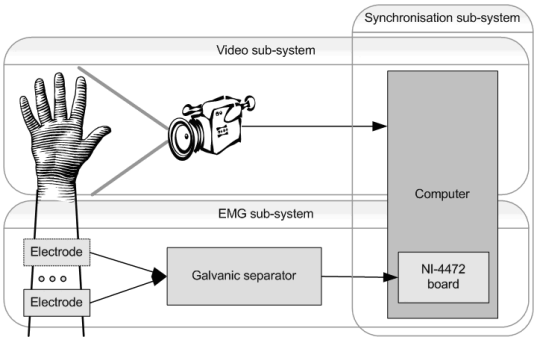


Fig. 1. The measurement system for identifying the relation between the hand movement and EMG signals.

2.2 Features Extraction

The extraction of features consists in determining such parameters that best differentiate the received signals for the sake of movement recognition. The extraction of features can be accomplished using various techniques including signal amplitude, EMG frequency characteristic and power spectrum analyzed by fast Fourier transform (FFT) method [6], the integral of the absolute value (IAV) and zero crossing signal [6, 7], time and frequency histograms [11], among others. In this paper it is proposed an efficient method to determine the input features based on autoregressive (AR) model.

The AR model belong to a group of linear prediction methods that attempt to predict an value y_n of a time series of data $\{y_n\}$ based on the previous values $(y_{n-1}, y_{n-2}, \ldots)$.

Deriving the linear prediction model involves determining the coeffiecients $(a_1, a_2, \ldots, a_p)$ in the equation:

$$\hat{y}_n = \sum_{k=1}^{p} a_k y_{n-k}, \tag{1}$$

where $\hat{y}_n$ is the estimated value of signal in a time n, a_k are the AR coefficients and p is the order of AR model.

Several estimators of AR coefficients are well known in the field of signal processing. We chose the Burg algorithm because of its many remarkable advantages (it does not apply window data, minimizes forward and backward prediction errors, gives high resolution for short data records, always produces a stable model) [9]. The Burg algorithm estimates the AR coefficients by fitting an autoregressive linear prediction filter model of a given order to the signal. Consequently, the Burg algorithm determines for each channel the set of p AR coefficients, which create the feature vector describing the EMG signal (r is the number of channels):

$$x = [a_{11}, a_{12}, \ldots, a_{1p}, a_{21}, a_{22}, \ldots, a_{2p}, \ldots, a_{r1}, a_{r1}, \ldots, a_{rp}]. \tag{2}$$

2.3 Classification

Two types of artificial feedforward neural networks were used in this study for classification of EMG signal: multilayer perceptron (MLP) and Adaline network.

1. **The MLP Classifier (MLP).** The network consists of the input, hidden, and output neuron layers. The input layer plays the role of a data buffer so that the data are normalized to belong to the [0, 1] range. There have been various numbers of input and hidden layer neurons, depending on the actual quantities of data. The number of output layer neurons is equal to the number of classes (types of grasps). The final classification is made according to the maximum rule. Both the hidden and output layer neurons have the sigmoid transition function. Neurons of the successive layers are connected on the each-to-each basis. In the experiments, the corresponding layers were trained by means of the error back propagation method with momentum term.
2. **The Adaline Classifier (ADA).** The single layer neural network that contains neurons with (positive) linear transfer functions. As previously, the number of neurons is equal to the number of classes and the final classification is made according to the maximum rule. In the experiments the Adaline network was trained by Widrow and Hoff learning procedure, also known as the delta rule.

3 Experiments

The proposed methods of EMG signal classification based on ANN techniques were experimentally tested and their performances were compared against the four following pattern recognition techniques: (1) Naive Bayes method (NB) [4]; (2) Parzen classifier with the Gaussian kernel and the optimal smoothing parameter (PAR) [4]; (3) 5-nearest neighbours classifier (5-NN) [4] and (4) classifier based on fuzzy relations (FR) [13].

3.1 Experimental Setup

The experiments were carried out on healthy persons. The electrodes, connected to the respective measuring channels, were put over the following forearm muscles: (1) the extensor muscle of the fingers, (2) the radial extensor of the wrist, short, (3) the superficial flexor muscle of the fingers, (4) the ulnar flexor muscle of the wrist, (5) the extensor muscle of the thumb, short, and (6) the flexor muscle of the thumb, long (see Fig. 2).

The experiments were conducted in MATLAB using PRTools and NN Toolbox.

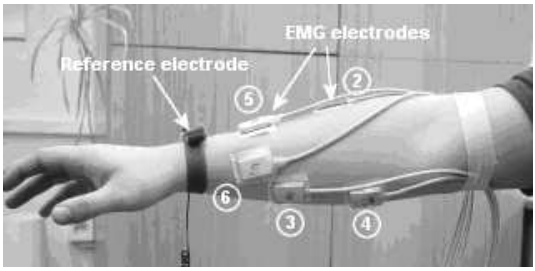


Fig. 2. The layout of the electrodes on the forearm.

In experiments five different types of grasps (classes) presented in Fig. 3 were chosen for recognition from the set defined by Schlesinger ([8]): 1) palmar, 2) tip, 3-4) cylindrical and cylindrical tight, 5) spherical. Our choice is deliberate one and results from the fact that the control functions of simple bioprosthesis are hand closing/opening and wrist pronantion/supination, however for the dexterous hand these functions differ depending on grasped object [2].

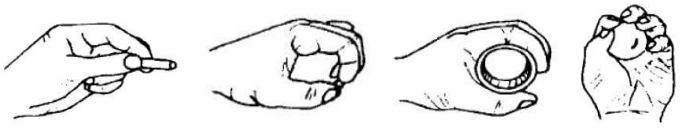


Fig. 3. Types of grasps recognized in experiment.

Each measurement lasted 2.5 s and was preceded with a 10 s break. In that way for the single grasp movements the discrete signals were obtained each of a size of 2500 samples (1 kHz sampling frequency) $\times$ 6 channels, together with the video sequences related to them, that picture the movement types (classes). The 300 measurements (60 measurements for each grasp type (class)) were created, and next gathered EMG signals were subjected to the feature extraction procedure for different orders of AR model p equal to 2, 3, 5, 7 and 10. Consequently, we got 5 datasets, each containing 300 patterns described by 12, 18, 30, 42 and 60 features, respectively. The training and testing datasets were extracted from each dataset using two-fold cross-validation method.

The ADA classifier comprised 5 neurons which inputs number was equal to the number of features (different for each dataset). Similarly, the MLP classifier comprises 5 neurons in the output layer and the number of input neurons (hidden neurons) was equal to 12 (8), 18 (10), 30 (15), 42 (20) and 60 (30) for the successive datasets, respectively. The number of epochs in the learning procedure for the both ANN classifiers was equal to 200.

3.2 Results and Discussion

Classification accuracies (i.e. the percentage of correctly classified objects) for methods tested are listed in Table 1. The accuracies are average values obtained over 10 runs (5 replications of two-fold cross validation). Statistical differences between the performances of the ADA, MLP classification methods and the four classifiers were evaluated using Dietterich's 5x2cv test [3]. The level of $p < 0.05$ was considered statistically significant. In Table 1, statistically significant differences are given under the classification accuracies as indices of the method evaluated, e.g. for the dataset with $p = 5$ the MLP classifier produced statistically different classification accuracies from the NB, 5-NN and FR methods. The row "'Mean"' contains results averaged over all datasets.

Table 1. Classification accuracies of classifiers compared in the experiment (description in the text). The best score for each dataset is highlighted.

	Classifier / Mean (SD) accuracy [%]					
AR order	NB_1	PAR_2	$5\text{-}NN_3$	FR_4	ADA	MLP
$p = 2$	73.2(5.2)	82.8(2.8)	86.5(4.2)	72.4(6.3)	84.2(3.1) 1,4	**87.7**(2.2) 1,2,4
$p = 3$	79.3(4.6)	90.2(1.9)	**94.2**(1.6)	80.6(2.4)	91.0(1.3) 1,4	93.6(1.2) 1,2,4
$p = 5$	81.5(2.2)	**97.6**(0.3)	85.3(1.4)	83.5(3.6)	94.8(1.1) 1,4	97.4(0.6) 1,3,4
$p = 7$	80.4(2.5)	98.2(0.7)	95.8(0.9)	87.2(1.3)	98.3(0.2) 1,3,4	**100** (0.0) 1,3,4
$p = 10$	82.7(2.3)	98.1(0.5)	96.9(0.4)	91.5(1.2)	**100**(0.0) 1,2,3,4	**100**(0.0) 1,2,3,4
Mean	79.4(3.4)	93.4(1.2)	93.8(1.7)	83.0(2.9)	93.7(1.1) 1,4	**95.7**(0.8) 1,2,3,4

The MLP classifier achieved the highest overall classification accuracy averaged over all datasets – it outperformed the NB, PAR, 3-NN and FR classifiers by 16.3%, 2.3 %, 1.9% and 12.7% on average, respectively. The ADA neural network that was the third best-scoring classifier, outperformed the NB, PAR and FR systems by 14.3%, 0.3% and 10.7% on average, respectively. The both ANN-based classifiers produced statistically significant higher scores in 29 out of 40 cases (5 datasets $\times$ 4 classifiers $\times$ 2 systems developed). The ADA and MLP classifiers also achieved the highest classification accuracy (i.e. 100%) when the datasets with 42 and 60 features were used.

Furthermore, they produced the best stability (the SD values of 1.1% and 0.8% averaged over all datasets), followed by the PAR classifier (1.2%). Results obtained indicate, that proposed methods of grasping movement recognition based on the AR model as an EMG signal feature extraction procedure, produced accurate and reliable decisions, especially in the cases with greater number of features.

References

1. Boostani R., Moradi M., Evaluation of the forearm EMG signal features for the control of a prosthetic hand, Physiological Measurement 24 (2003), 309-319
2. De Luca C., J., Adam R., et al., Decomposition of surface EMG signals, Journal of Neuropsychology 96, (2006) 1646-1657
3. Dietterich T.G., Approximate statistical tests for comparing supervised classification learning algorithms, Neural Computing 10 (1998) 1895-1923
4. Duda R., Hart P., Stork D., Pattern Classification, John Wiley (2000)
5. Ferguson S., Dunlop G., Grasp recognition from myoelectric signals, Proc. 2002 Australasian Conference on Robotics and Automation (2002) 83-87
6. Kuribayachi K., Okimura K. and Taniquichi T., A discrimination system using neural network for EMG-controlled prostheses, Proc. Int. Conf. IEEE Robot and Human Communication (1992) 63-68
7. Khoshaba T., Badie K., and Hashemi R., EMG pattern classification based on back propagation neural network for prosthesis control, Proc. Ann. Int. Conf. IEEE Eng. Med. Biol. Soc. (1990) 1474-1475
8. Reaz M., Hussain M., Techniques of EMG signal analysis: detection, processing, classification and applications, Biological Procedures Online 8 (2006) 11-35
9. Schloegl A., A comparison of multivariate autoregressive estimators, Signal Processing 9 (2006) 2426-2429
10. Wang, G., Zhiguo, Y., Xiao, H., et al., Classification of surface EMG signals using harmonic wavelet packet transform, Physiol. Measurement 27 (2006) 1255-1267
11. Wojtczak P., Amaral T., Dias O., Hand movement recognition based on biosignal analysis, Engineering Applications of Artificial Intelligence 22 (2009) 608-615
12. Wolczowski A., Krysztoforski K., Control-measurement circuit of myoelectric prosthesis hand, Acta Bioengineering and Biomechanics 4 (Suppl.) (2002) 576-578
13. Wolczowski A., Kurzynski M., Human-machine interface in bioprosthesis control using EMG signal classification, Expert Systems 27, (2010) 53-70

SPIDAR Calibration based on Neural Networks versus Optical Tracking

Pierre Boudoin, Hichem Maaref, Samir Otmane and Malik Mallem

Laboratoire IBISC, Université d'Evry, Evry, France
{pierre.boudoin, hichem.maareef, samir.otmane
malik.mallem}@ibisc.fr

Abstract. This paper aims to present all the study done on the SPIDAR tracking and haptic device, in order to improve accuracy on the given position. Firstly we proposed a new semi-automatic initialization technique for this device using an optical tracking system. We also propose an innovative way to perfom calibration of 3D tracking device using virtual reality. Then, we used a two-layered feed-forward neural network to reduce the location errors. We obtained very good results with this calibration, since we reduced the mean error by more than 50%.

1 Introduction

Virtual reality is a domain which is highly dependent on tracking systems. Users interact in 3 dimensions, with virtual entities in digital environments. In order to provide the best user experience, it's very important that 3D interaction has to be without any interruption. This interaction relies on the transformation of a real movement into an action in the virtual world. This work is done by a tracking solution. This tracking system has to be reliable and the most available as possible. This point is crucial in order to preserve data continuity and, so, data processing continuity and finally, 3D interaction continuity. The main device used in our system is an optical tracking solution, it's a very accurate device. On the other hand, it suffers from a huge defect: tracking-loss. That's a particular true defect when only one marker is used. So, it's essential to be able to switch to another device in these situations in order to compensate this defect. In our virtual reality system, we've got a SPIDAR [1] and we chose it to stand in for the optical tracking system.

SPIDAR [1], for *SPace Interaction Device for Augmented Reality*, is an electromechanical device, which has 8 couples of motor/encoder distributed on each vertex of a cubic structure. A string is attached to each motor via a pulley. These 8 strings converges to an effector. By winding their respective strings, each motor produces a tension. The vectorial sum of these tensions produce the force feedback vector to be applied on the effector, allowing the user to feel on what he is stumbling or to feel the weight of an object. By observing the encoders values, the system can compute the 3D position of the effector. The SPIDAR tracking is always available, but it suffers from a weak accuracy and repeatability. So it's impossible, when we want to 3D interact with accuracy, to use raw position given by the SPIDAR without performing a calibration.

In our case, it's a huge problem, since we used a 3D interaction technique, called *Fly Over* [2], which needs a continuous position vector. This technique is based on

different interaction areas offering to the user a continuity in the interaction. Indeed, the least jump of position during the swing of a system, would be likely to pass the pointer of *Fly-Over* of a zone of interaction towards another. This phenomenon involves a behavior of the technique thus, not wished by the user and creating consequently a rupture of the continuity of the 3D interaction. Thus, it's important to propose measures in order to consider the position given by the SPIDAR so that it is closest to the position given by the optical tracking system, and so, minimizing effects on the 3D interaction.

This research work is presented as follow. First, we talk about similar works on virtual reality devices calibration and correction. Then, we introduce our contribution on a new SPIDAR calibration method using multimodal informations. After that, we speak about the correction of the SPIDAR position using neural networks. Finally, we discuss about a hybrid tracking system based on a SPIDAR and an optical tracking solution.

2 Related Work

Since virtual reality systems use more and more devices, especially tracking devices, it's important to perform a good calibration of them. But not all tracking devices need a huge correction, thus infrared based optical tracking devices are accurate enough and so don't need to be corrected. On the other hand, it exists some mechanical, electromechanical or electromagnetic tracking devices which need to be calibrated and/or corrected.

Most of research works has been realized on the electromagnetic tracking devices because they suffers from electromagnetic distortions when magnetical materials are placed into the tracking range. Moreover, the tracking accuracy falls off rapidly depending on the distance from the emitter and the power of the emitter [3]. These effects induce non-linear errors on the location. In order to correct them, it exists different ways.

The easiest method is the linear interpolation [4] but it doesn't correct non-linear systems, so it's very limited. Polynomial fitting [5, 6] allows to correct non-linear errors. But depending on the number of coefficients, it could be very difficult using this method in near realtime conditions because it will produce a heavy load for the system. Moreover if the number of coefficient is too important, oscillations can appear, increasing errors rather than decreasing them. Moreover, these techniques often fail to capture small details in the correction. They are better in determining the overall shape of a non-linear function. Kindratenko [7] and Saleh [8] worked on a neural network based calibration of electromagnetic tracking systems and they obtained good results, better than with other methods.

But all these techniques are based on interpolation and they need a valid set of data to be effective. This set of data highly is often given by a calibration grid. A calibration grid is a representation of a set of points. All these point have a known position and can be compared with the position given by the device that we want to calibrate. But when we're working in 3D space, it's very difficult to make use of it because it's difficult to place accurately a device on a 3D points. In order to realize that we can use another

mechanical device, such a robot arm or a haptic arm [9]. Or we can place accurately passive sensors respecting a geometrical shape [10].

Our research work reaches these studies because the SPIDAR suffers from same non-linear distortions and 3D calibration problematic. So, we search solutions in the same direction.

3 Identification of the SPIDAR

3.1 Context

In order to preserve the data continuity, it is essential to correct a well-known problem appearing with tracking systems: data loss. Data loss appears when the tracking is unable to update the position calculation, conducting to a jump in the data when the system is re-enable to update the position. This phenomena is often mislead by occultation, especially in optical tracking system. A data loss can be managed by three methods:

1. **Prediction:** We can predict the following data state by knowing the previous data state through mathematical method , such Kalman filter.
2. **Compensation:** A device tracking loss, don't forbid us to use another device. It's very important in this case that the data incoming from the different devices to be expressed in the same space representation (same referential). This is necessary in order to obtain a data continuity when the system switch from one device to another.
3. **Correction:** The last possibility is to correct data incoming from the most available device, in our case the SPIDAR. To perform the correction, we could use the a priori knowledge on the SPIDAR position through another device.

3.2 Design Problems

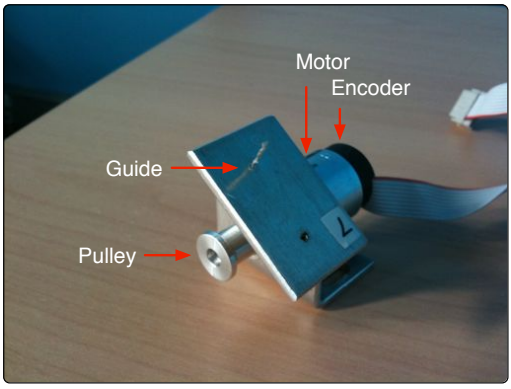


Fig. 1. Detailled view of a SPIDAR's motor and its winding guide.

SPIDAR is an electromechanical device and consequently it could suffer from design problems more or less awkward for computing the effector's position. These are problems we have identified:

1. **Encoders are Directly Mounted on the Motor's Axis.**
   This is an important problem because we must define the pulley's diameter in the configuration file of the SPIDAR's interface. However, this diameter is not constant, depending on the quantity of string winded. So, this information is skewed.
2. **Diameter of Pulleys is too Small.**
   The previous problem become more marked due to the small diameter of the pulley used. Thus, the diameter being too small, it variates noticeably as strings being winded go along. This phenomena would be less marked if the diameter used was more important.
3. **Winding Guides Badly Designed.**
   The present design of the winding guides, don't prevent a string from missing the pulley. This phenomena appears when the effector is being moved fast and consequently, that motors have to wind an important quantity of string. This is a real problem, because the encoder count one revolution but the string doesn't be winded.
4. **Size of Encoders.** Encoders' size is too small for counting the string quantity which must be winded. When an encoder overflows, the counter is reseted and the winded string quantity information is biased.
5. **Dimensions of the SPIDAR.** More dimensions are important and more every problem cited previously is marked. Some problems that are inconsiderable when dimensions are small, become not inconsiderable when dimensions are huge.

3.3 Experimental Protocol

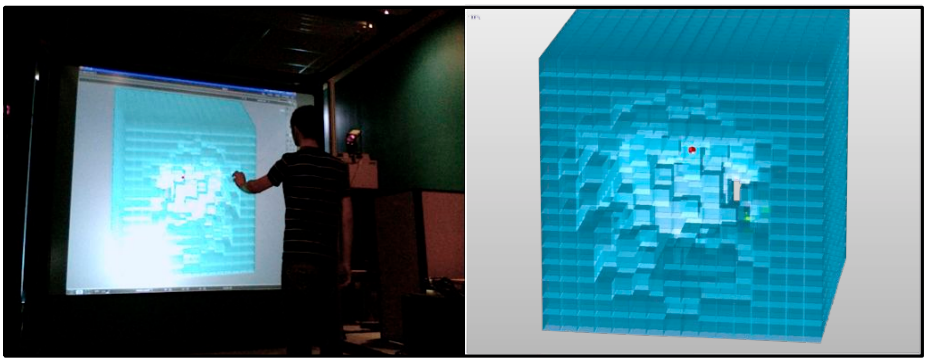


Fig. 2. On the left - A user using our virtual calibration grid in order to retrieve data for the neural network learning. On the right - Detailed representation of the virtual calibration grid.

We use what we called: a virtual calibration grid (see fig.2), which consists in the representation of a virtual scene, composed of many small cubes. Each cube corresponds

to a sub-space of the SPIDAR workspace. This set of small cubes covers the whole SPIDAR workspace. We recorded values, respecting this protocol in a workspace limited to 1 m^3 split into 4096 sub-spaces (16 x 16 x 16). The great advantage of this protocol is the homogeneity distribution of the data set.

The use of virtual reality for calibration allows more flexibility and less complexity because we don't have to move the SPIDAR effector with constraints or to place the effector with a great accuracy on a set of calibration point.

This calibration grid, is represented Fig.2. We can identify the SPIDAR's problem with it, following these steps:

1. The user move the real effector (which is in his hand) in order to place the virtual effector (which is a red sphere in the virtual scene) in each cube represented.
2. Each time the virtual effector is in collision with a cube, we record the position given by the SPIDAR and the position given by the optical tracking.
3. Once these positions is recorded the cube disappears insuring that there will be only one point for this sub-space.

3.4 Results

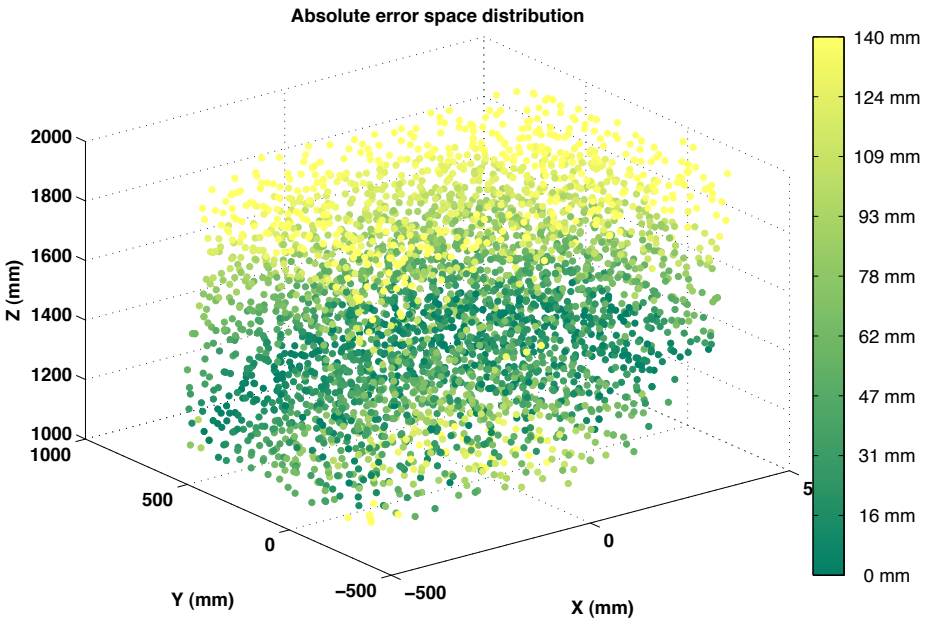


Fig. 3. Absolute error 3D distribution in the SPIDAR's workspace (Dark green is the best).

Figure 3 represents errors' space distribution. As we can see this is a onion skin distribution, meaning different spherical layers, the absolute error growing as the effector is going in outside layers. This identification of the SPIDAR leads us to these observations:

- Firstly, the mechanical study tells us that too many design problems
- Moreover, it's hard to quantify the final effect of these mechanical problems.
- Another problem, is the loss of knowledge towards the mathematical model used by the SPIDAR to compute the effector's position.

Finally, the SPIDAR suffers from a set of problems, which have more or less known causes and for which we don't know very well the influence on the whole system. In order to enhance the SPIDAR's accuracy, it could be interesting to orient ourself to a solution capable of estimating/correcting the effector's position without any knowledge on the mathematical model. We choose to test that way using neural networks for their capacities to learn a situation and to model any continuous mathematical function without any information on the model.

4 Multimodal Initialization of SPIDAR

4.1 Context

The SPIDAR is a device which needs an initialization at each startup. Initialization consists to define the origin of the referential in which 3D position will be expressed. We realize this task by placing the SPIDAR's effector in the center of its cubic structure with the greatest accuracy. A worse initialization brings about a decline in accuracy for the 3D position. Moreover, if the initialization is not identical at each startup, the new referential won't be the same and consequently prevent us from applying a data-processing for correcting the SPIDAR's position. So, it's very important to realize this initialization with a great attention. But, it's very difficult to hand-place the effector in the center of the SPIDAR structure with accuracy, due to the lack of markers to estimate this.

4.2 Proposition

Our contribution brings a semi-automatic SPIDAR initialization. This initialization use multi-modality in order to guide the user. These modalities are vision, audio and haptic. In order to carry out the more accurate initialization, we need to determine the initialization point in space with great accuracy. To realize that, our idea is to use the optical tracking system we have on our VR platform (and in most common VR platform). This tracking system offers a lower than 1 mm precision.

The geometrical disposition of IR cameras are known and the origin of the optical tracking system too. We also know the theoretical initialization point of the SPIDAR. Thus, we could determine the theoretical initialization point within the optical tracking referential. In order to know the position of the effector in the optical tracking space, we put an optical marker on it. Thus, if we compute the vector defined by the initialization point and the position of the effector, we could get the distance from the initialization point and the direction needed in order to converge to it.

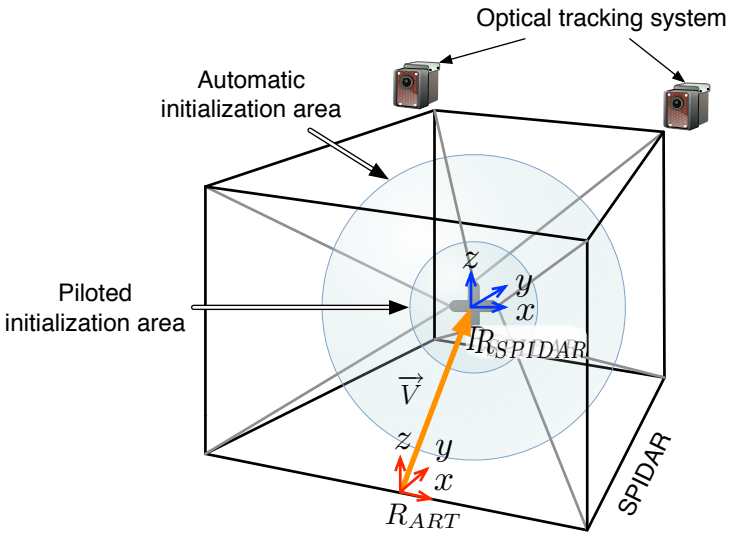


Fig. 4. Initialization areas and experimental framework.

4.3 Multimodal Parameters

In order to guide the user in the initialization step, we firstly need to convert these information into multimodal parameters. So we define a h function, returning an index, expressing the distance from the initialization point d. More $h(d)$ tends to 1, more the effector is near from the initialization point. In the opposite way, more $h(d)$ tends to 0 and more the effector moves away from the initialization point.

$$h(d) = \{\, h(d) = 0 \quad \text{if } d > d_{min} \tag{1}$$

With:

d, the distance in millimeter between the effector and the initialization point.
d_{min}, the minimal distance in millimeter from which $h(d)$ begins fluctuating, else $h(x)$ is equal to 0.

In the following part, we define multimodal parameters that help the user to perform a good initialization.

– **Sound Modality.** Beep frequency f depends on the distance d.

$$\begin{cases} f(d) = d \cdot 1000 & \text{if } d \leq 1000 \\ f(d) = 1 & \text{if } d > 1000 \end{cases} \tag{2}$$

– **Haptic Feedback Modality.** The direction vector is transformed into a force feedback vector to be applied on the effector by this equation.

$$\overrightarrow{F} = \overrightarrow{\Delta} \cdot s(d/d_f) \cdot F_{Max} \tag{3}$$

With:

$s(x)$, the sigmoid function regulating the force applied to the effector.

$$s(x) = e^{-10(\|x\|-0.5)^2} \quad \text{if } x \in [-0.5; 0.5] \tag{4}$$

Δ, the unit vector defining the direction to the initialization point. It's computed from the normalization of vector $\overrightarrow{V}$:

$$\overrightarrow{\Delta} = \frac{1}{\|\overrightarrow{V}\|} \cdot \overrightarrow{V} \tag{5}$$

F_{Max}, the maximum force to be applied on the effector.

4.4 Initialization Algorithm

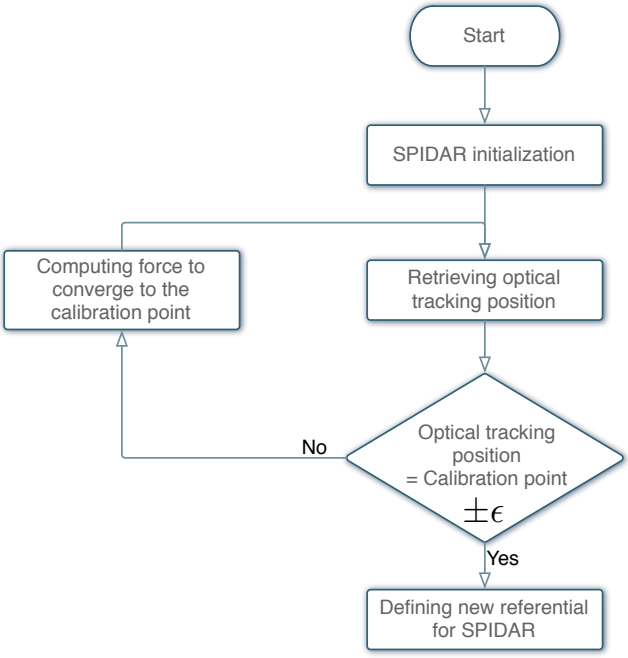


Fig. 5. SPIDAR initialization process algorithm.

Initialization process is done in two steps. A first initialization allows the user to place the effector in a area near from the initialization point with a 1 or 2 cm accuracy. We can't decrease below this distance using directly the SPIDAR's force feedback capabilities because it is not enough sensitive for moving the effector on small distances. In order to get a more accurate initialization, we need to add another step to the initialization process. Using the different modalities previously cited, the user's hand, holding the effector, will be guided to converge to the initialization point with a $1.2\ mm$ accuracy.

5 Calibrating the SPIDAR with Neural Network

5.1 Configuration & Learning

We used a two-layered neural network, the first layer having a sigmoid activation function and the second a linear one. It's a feed-forward back-propagation network using the *Levenberg-Marquardt* learning algorithm [11]. The mean quadratic error is used as performance function.

For the learning step, we use the SPIDAR's position vectors in input and the optical tracking's position vectors in output because this is what we want in theory. However, the whole vectors aren't used, only data where the two tracking systems are available has been used for the learning. It's important for the learning step to remove data which would decrease the neural network performances. Data income from the experimental protocol described previously. So we obtain 4096 measure points. This data set has been split into 3 sub-sets.

- 60% of data are used for the learning algorithm.
- 20% of data are used for the validation step, in order to prevent over-fitting phenomenon.
- 20% of data are used to perform a generalization, that is the observation of the neural network's response to the introduction of set of totally unknown data (data which haven't be used for learning) in input.

5.2 Optimal Number of Neurons

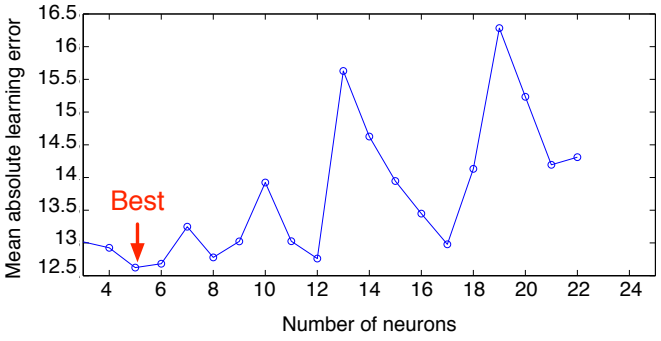


Fig. 7. Mean absolute error versus number of neurons in the hidden layer.

The optimal number of neurons in the hidden layers has been defined in an empirical way, by testing the result of learning with different number of neurons and by observing the mean absolute learning error. The more this error is high, the less the neural network is effective. The figure 7 shows the mean absolute error on the SPIDAR position according to the number of neurons in the hidden layer. By observing this result, we could determine that the best configuration among the 3 to 21 neurons configurations, is 5.

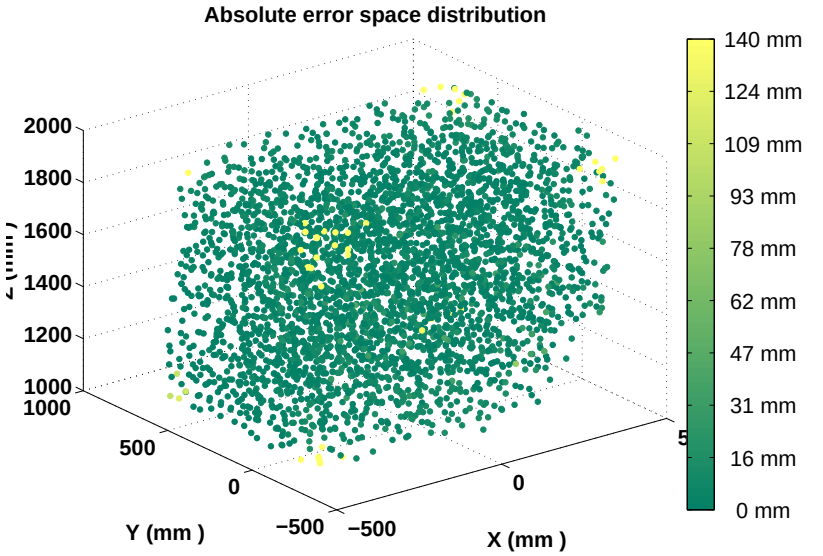


Fig. 8. Absolute error 3D distribution in the SPIDAR's workspace after calibration by the neural network (Dark green is the best).

Table 1. Characteristic values of absolute errors on SPIDAR location for neural network learning.

Absolute error	Raw	NN	PF1
mean (mm)	72.86	7.44	15.31
std (mm)	47.05	6.24	40.75
max (mm)	211.29	86.90	198.32

5.3 Neural Network Performances

In order to study the neural networks performances, we represent the same absolute error 3D spatial distribution as previously but after using our neural network. As we can see, figure 8, shows that the neural network is quite effective and greatly improve the SPIDAR accuracy in comparison with the figure 3. The neural network performs a good calibration in the whole workspace except in its corners.

We also traced errors bar graphs before and after using our neural network (see Fig.9) and we put representative data in arrays comparing them with raw location (Raw), neural network calibrated location (NN) and for information purpose only, linear interpolation calibrated location (PF1). Each bar graph is coupled with an array resuming characteristic values of the error distribution, where *mean* is the empirical mean error, *std* is the standard deviation of the data set and *max* is the maximum error.

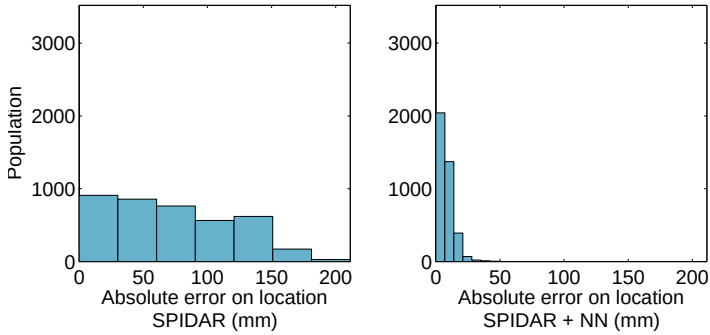


Fig. 9. Absolute errors distribution bar graph before and after calibration by the neural network.

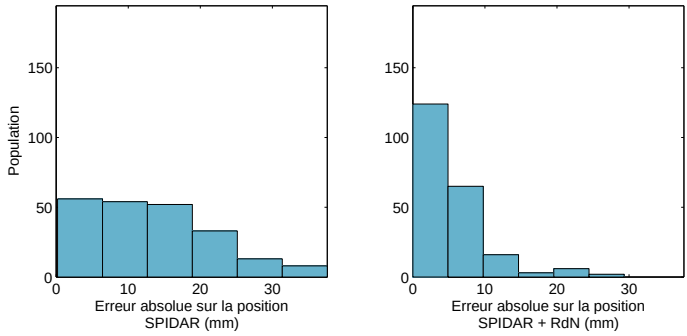


Fig. 10. Absolute errors distribution bar graph before and after calibration by the neural network on the 1^{st} data set.

5.4 Generalization

In order to evaluate neural network performances, we need to observe this response output with unknown data sets. This step is called *generalization*. Figures 10 show results obtained with two generalization data sets. These data sets have been recorded during two time-split measure campaigns and using our semi-automatic initialization procedure.

The neural network has been nicely set up and is robust at any data as soon as the data belongs to the SPIDAR's workspace. Such results aren't surprising since the learning protocol was extremely rigorous and covered all the SPIDAR's workspace guaranteeing the neural network to perform a good interpolation. Moreover tables 1 and 2 shows us that the standard deviation with neural networks is smaller than the raw or the other method ones, which indicates a smoother and a more accurate interpolation.

Table 1 and figure 10 shows us that the used neural network has a good response to the generalization. Thus, the absolute mean error on the position has been reduced by 2.5 times, going from 13.3 mm to 5.31 mm.

Table 2. Characteristic values of absolute errors on SPIDAR location in generalization with data set 1.

Data set 1	Raw	NN	PF1
mean (mm)	13.23	5.31	7.91
std (mm)	8.41	5.14	7.68
max (mm)	37.60	29.42	32.51

6 Conclusions

In this paper we propose a method to calibrate SPIDAR using a feedforward neural network coupled with a semi-automatic initialization. The semi-automatic initialization allows us to place the SPIDAR referential at the same 3D position at each startup with an accuracy of $1.2\ mm$. This way, we can use a method for calibrating the SPIDAR which, don't need to be updated at each startup. We choose a feedforward neural network in order to compensate non linear errors on location and their abilities to estimate a targeted output from a source without any knowledge on the mathematical model. We obtain good results and our whole calibration procedure is efficient. Testing our neural network in generalization shows us that our calibration is quite robust, even if we reset the SPIDAR. We plan to make the initialization procedure fully automatic.

References

1. Kim, S., Ishii, M., Koike, Y., Sato, M.: Development of tension based haptic interface and possibility of its application to virtual reality. Proceedings of the ACM symposium on Virtual reality software and technology (2000) 199–205
2. Boudoin, P., Otmane, S., Mallem, M.: Fly over, a 3d interaction technique for navigation in virtual environments independent from tracking devices. VRIC '08 (2008) 7–13
3. Nixon, M., McCallum, B., Fright, W., Price, N.: The effects of metals and interfering fields on electromagnetic trackers. Presence, 7, (1998) 204–218
4. Ghazisaedy, M., Adamczyk, D.: Ultrasonic calibration of a magnetic tracker in a virtual reality space. Virtual Reality Annual International Symposium (VRAIS'95) (1995)
5. Bryson, S.: Measurement and calibration of static distortion of position data from 3d trackers. Siggraph'92 (1992)
6. Ikits, M., Brederson, J., Hansen, C.D., Hollerbach, J.M.: An improved calibration framework for electromagnetic tracking devices. IEEE Virtual Reality Conference 2001 (VR 2001) (2001)
7. Kindratenko, V.: A survey of electromagnetic position tracker calibration techniques. Virtual Reality,5, (2000) 169–182
8. Saleh, T., Kindratenko, V., Sherman, W.: On using neural networks to calibrate electromagnetic tracking systems. Proceedings of the IEEE virtual reality annual international symposium (VRAIS '95) (2000)
9. Ikits, M., Hansen, C., Johnson, C.: A comprehensive calibration and registration procedure for the visual haptic workbench. Proceedings of the workshop on Virtual environments 2003, 39 (2003) 247–254
10. Leotta, D.: An efficient calibration method for freehand 3-d ultrasound imaging systems. Ultrasound in medicine & biology (2004)
11. Bishop, C.: Neural networks for pattern recognition. Oxford University Press (2005)

On Human Inspired Semantic SLAM's Feasibility

Dominik Maximilián Ramík, Christophe Sabourin and Kurosh Madani

Images, Signals, and Intelligent System Laboratory
(LISSI / EA 3956), PARIS-EST Creteil (UPEC)
Senart Institute of Technology, Avenue Pierre Point, Lieusaint, 77127, France
{dominik.ramik, sabourin, madani}@univ-paris12.fr

Abstract. Robotic SLAM is attempting to learn robots what human beings do nearly effortlessly: to navigate in an unknown environment and to map it in the same time. In spite of huge advance in this area, nowadays SLAM solutions are not yet ready to enter the real world. In this paper, we observe the state of the art in existing SLAM techniques and identify semantic SLAM as one of prospective directions in robotic mapping research. We position our initial research into this field and propose a human inspired concept of SLAM based on understanding of the scene via its semantic analysis. First simulation results, using a virtual humanoid robot are presented to illustrate our approach.

1 Introduction

In mobile robotics, the ability of self-localization with respect to the environment is crucial. In fact, knowing precisely where the robot is, and what kind of objects surround it in any given moment of the time enables it to navigate autonomously and to interact with an unknown environment in a conscious manner. An informal definition of the Simultaneous Localisation And Mapping (SLAM) describes it as a process, in which a mobile robot explores an unknown environment, creates a map of it and uses it simultaneously to infer its own position on the map. In the real world SLAM applications, data association has often to be done under large amount of uncertainty. Moreover, the real environment is usually very complex and dynamic and it is not easy for a robot to interpret it in a reliable way. It is this complexity, what makes SLAM a challenging task. A comprehensive list and principle explications of nowadays most common SLAM techniques can be found in [1], [2] and [3]. Although from its beginning until present days the research community achieved a significant advance on the field of SLAM [4], it is not yet a solved problem. Autonomous navigation in dynamic environment [5] or understanding the mapped environment by including semantics into maps [6] are the actual challenges of this research field..

In this paper, the state of the art in robotic mapping is investigated. We identify a relatively new field of research within the field of SLAM, which is attempting to perform simultaneous localization and mapping with the aid of semantic information extracted from sensor readings. One of the research interests of our laboratory (LISSI) is autonomous robotics notably in relation to humanoid robots. We are convinced that the research on semantic SLAM will bring a useful contribution on this

topic. We position our initial research into this field, drawing our inspiration from human way of navigation and place description. In fact, contrary to most of current SLAM techniques, which tend to infer precisely and globally the navigation environment, the human way of doing is based on very fuzzy description of the environment and it gives preference to local surroundings of the navigation backdrop. A simulation using a virtual humanoid robot (Nao robot) is presented to demonstrate some of the proposed ideas. The real Nao will be used in our further work.

The paper is organized in the following way: section 2 focuses on the state of the art in semantic SLAM. In the third section, we are discussing our approach to image segmentation and scene interpretation. Section 4 gives an overview of the robotic humanoid platform we use. The fifth section presents our initial results and the paper concludes with section 6.

2 Semantic SLAM

In this section, one of the latest research directions on the field of SLAM, the so-called semantic SLAM, is discussed. The concept itself may be perceived as a very important and pertinent one for future mobile robots, especially the humanoid ones, that will interact directly with humans and perform tasks in human-made environment. In fact, it is the human-robot interaction, which is probably one of important motives for employing semantics in robotic SLAM as humanoids are particularly expected to share the living space with humans and to communicate with them.

Semantics may be incorporated into the concept of robotic SLAM in many different ways to achieve different goals. One aspect of this may be the introduction of human spatial concepts into maps. In fact, humans usually do not use metrics to locate themselves but rather object-centric concepts and use them for purposes of navigation ("I am in the kitchen near the sink" and not "I am on coordinates [12, 59]") and fluently switch between reference points rather than positioning themselves in a global coordinate system. Moreover, presence of certain objects is often the most important clue for humans to recognize a place. An interesting work addressing the mentioned problems has been published in [7], in which the world is represented topologically with a hierarchy of objects and place recognition is performed based on probability of presence of typical objects in an indoor environment. A part of the work shows a study based on results of questioning about fifty people. The study was aimed to understand human concepts of self-localization and place recognition. It proposes that humans tend to understand places in terms of significant objects present in them and in terms of their function. A similar way (i.e. place classification by presence of objects) has been taken by [8] where low-level spatial information (grid maps) is linked to high-level semantics via anchoring. During experiments, the robot has interfaced with humans and performed tasks based on high-level commands (e.g. "go to the bedroom") involving robots "understanding" of the concept of bedroom and usage of low-level metric map for path planning. However, in this work, object recognition is black-boxed and the robot is not facing any real objects in the experiments but only boxes and cylinders of different colours representing different real-world objects.

An approach treating this "gap" between object recognition and semantic SLAM is presented in [9]. Here, a system based on a mobile robotic platform with an omni-directional camera is developed to map an outdoor area. The system generates a semantic map of structures surrounding the robot. Buildings and non-buildings labelled on the map. In [6], a more general system is presented, employing a wheeled robot equipped with a laser 3D scanner. Authors show ability of their robot to evolve in an indoor environment constructing a 3D semantic map with objects like walls, doors, floor and ceiling labeled. The process is based on Prolog clauses enveloping pre-designed common knowledge about such an environment (i.e. the doors are always a part of a wall and never a part of the floor). This enables the robot to reason about the environment. Further in the paper, an object detection method using the laser range data is shown with a classifier able to distinguish and to tag different objects surrounding the robot including humans and other robotic platforms. In [10] active object recognition is performed by a mobile robot equipped by a laser range finder and a camera with zoom. A semantic structure is extracted from the environment and integrated to robots map. It allows the robot to reach previously detected objects in an indoor environment. Another object recognition technique is shown in [11] including an attention system. Based on recognized objects a spatial-semantic map is built.

3 Image Segmentation and Scene Interpretation

Section 2 showed the pertinence of semantic SLAM in the frame of state of the art robotic mapping. It is exactly this field, on which we are focusing our research. Our motivation comes from the natural ability of human beings to navigate seamlessly in complex environments. Obviously, the way we are orienting ourselves in the space is very different from what contemporary robots do. To describe a place, we use often very fuzzy language expressions and approximation (as shown in [7]). This is in contrast with most of the current SLAM algorithms. In navigation or place description people also rarely use "global coordinates" but rather divide the scene into some kind of hierarchic clusters around distinctive objects, which then act as local origin of coordinates. Another interesting point is that people are able to infer distance of an object according to its apparent size and their experience of object's true size. From what has been mentioned so far it is clear that recognition of objects and understanding of their nature (semantic treatment) is an integral part of human navigation or "human SLAM" and not just an extra layer of it. We believe that application of semantics and human inspired scene description could bring a considerable benefit in development of robust SLAM applications for autonomous robotics.

To initiate a semantic scene interpretation, the image has to be segmented first. Although there exist many approaches to image segmentation (see [12]) for a reference), not all of them are suitable for purposes of mobile robotics, because it requires image treatment being done in real time. Our segmentation algorithm breaks the input image into parts containing similar colors present in different brightness levels. This should reflect that different shades do not usually mark different objects but only different light conditions on the same object.

We have chosen to use the YCbCr color model within our algorithm. This color model consists of three channels. The Y channel is dedicated to the luminance component of the image and stores the information about light and dark parts of the image. The other two channels Cb and Cr contain respectively the blue and the red chrominance component of the image. The YCbCr color model is more practical for purposes of our color segmentation algorithm, than classical RGB. It is because YCbCr separates the luminance of color and the color itself into different channels, while in RGB both color and its lightness are mixed together. The algorithm works in two stages, coarse to fine. In the first one, the Cr and Cb components of the image are acquired, their contrast is stretched and median filter is applied on both of them to remove noise. Then a single scan is made through rows and columns of the image and the first position that is not already occupied by a detected component is chosen as a seed point with coordinates x_{seed} and y_{seed}.

Eq. 1 captures how seed point is used to extract a segment of interest (S) from the image. The P stands for all the pixels in the image, whereas p is the actually examined pixel. Predicate C is true only if its two arguments (p, p_{seed}) are in four-connectivity and I stands for intensity of pixel. Seed pixel is denoted by p_{seed}. A pixel of the image belongs to the segment S under the following conditions: the difference of intensities of the current and the seed pixel is smaller than a threshold and there exists a four-connectivity between it and the seed pixel. Applying this on both chroma sub-images (Cr and Cb) we obtain segments denoted as S_{Cr} and S_{Cb}. A new segment S is then determined following Eq. 2 as the intersection of segments found on both chroma sub-images without pixels already belonging to an existing segment.

$$\forall p \in P;\ C(p, p_{seed}) \,\&\, |I(p) - I(p_{seed})| < \varepsilon \rightarrow p \in S \tag{1}$$

$$S = S_{Cr} \cap S_{Cb} - S_{all}. \tag{2}$$

At the end of the scan, a provisory map of detected segments is available leaving out components whose size is below certain threshold. At this stage, the image is often over segmented due to the method of selection of seed points. However, it serves as the first guess about the positions of regions. In the second step, all the provisory segments are sorted according to their area and beginning with the largest one the algorithm of segmentation is run again. This time the seed points are derived from centers of segments defined by Eq. 3 and Eq. 4.

The seed point k_{seed} is determined as such a pixel from the skeleton whose distance from its closest contour pixel is maximal. Here, K is the set of pixels of skeleton belonging to segment S and C is the set of contour pixels of S. D_i denotes the minimal distance between the given pixel k_i and the contour. In this step, similar segments from the previous step are effectively merged. At this point, found segments may contain distinctive areas of different brightness having similar chroma. The ultimate step of the algorithm is in constructing a histogram of luminance values of each segment. The histogram is then polished by application of sliding average. If multiple significant clusters are found in the histogram, the segment is broken-up accordingly to separate them. Having finished this step, found segments are stored for further use.

$$D_i = \arg\min_{k_i} \left| k_i - c_j \right| \qquad where$$
$$i \in \{0,...,|K|\} \ and \ k_i \in K = \{k_0,...,k_m\} \ and \ c_j \in C = \{c_0,...,c_n\} \tag{3}$$

$$k_{seed} = \max_{i \in \{0,...,|K|\}} D_i \tag{4}$$

$$Q = 4\pi n / o^2. \tag{5}$$

In the next processing step, the segments are labeled with linguistic terms describing their horizontal and vertical position and span with respect to the image frame. Both average color and its variance are computed for each segment along with the number of pixels forming the segment. The compactness (Q) of the segment is computed following Eq. 5, where n denotes the number of pixels of the segment and o the number of pixels forming the contour of the element. These features, which represent each segment, are then used in a set of linguistic rules, acting as a prior knowledge about the world. The aim is to determine the nature of each segment and its appurtenance to an object of the perceived environment. For example, a compact segment found in mid-height level surrounded by the wall is considered as a "window".

4 Humanoid Robot Platform

The robotic platform we use for simulation and experiments is described in this section. It is based on Nao, a humanoid robot manufactured by Aldebaran Robotics[1]. The robot is about 58cm high with height slightly exceeding 4kg. Its degrees of freedom are as follows: 2 DOF for the head, 5 DOF for each arm, 1 DOF for the pelvis, 5 DOF for each leg and 1 DOF for hands to control the grasp. Concerning the available sensors, it is equipped with two CMOS cameras with resolution up to 640x480px. One camera is on the front of the head and the other is covering the space around the feet of the robot (this one was added specially because of the usage of Nao in RoboCup robotic football matches). Two channel sonar and 2 IR sensors are in robot's chest. It also possesses a tactile sensor, bumpers and inertial sensors. To interact with humans, robot is equipped with voice synthesizer and a speech recognition unit.

The robot can operate in fully autonomous mode using its AMD Geode 500MHz processing unit to run programs and behaviors stored in its memory. Alternatively, it can be operated remotely from another computer via a WiFi (or Ethernet) connection. To perform simulations, a virtual version of Nao is available for the Webots simulation program developed by Cyberbotics[2]. The program allows us to create a virtual world and to simulate robots interaction with it including gathering of sensor data from cameras and IR/sonar sensors. Nao can be programmed in different manners.

[1] http://www.aldebaran-robotics.com
[2] http://www.cyberbotics.com/

The choice of languages includes C, C++, Python and URBI and the code can be run locally on robot's CPU or distantly via a network connection. After having explored different ways of programming

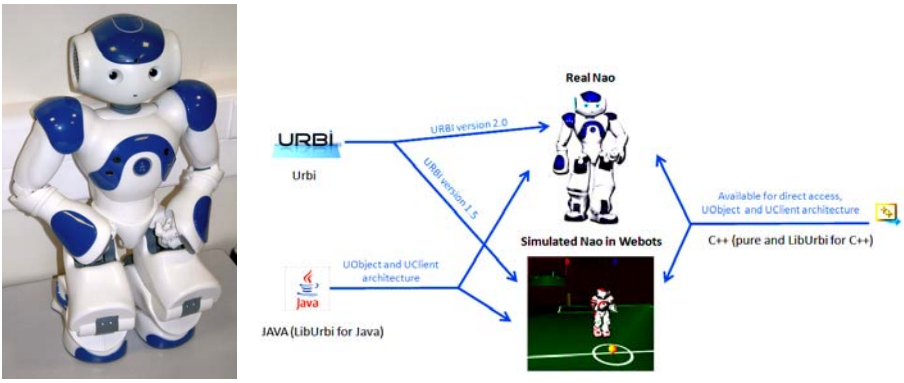


Fig. 1. A scheme describing our humanoid robotic platform, showing different possibilities of programming it. On the left a photo of the real Nao used in our laboratory.

Nao, we have chosen URBI developed by Gostai[3] for development purposes (see Fig. 1). There are several reasons for this choice. First, URBI is a specific language developed especially for robotics and by its nature allows simple and fast development of robotic behaviors. Moreover, it provides a simple way of managing parallel processes, which may be a complex task in other languages. Although programming in URBI involves writing in URBI script, which is then interpreted by an interpreter, URBI programs do not suffer from lose of performance. The code of its internal objects is written in C++ to keep high efficiency of the language. LibURBI connectors allow user to develop own objects using so called UObject architecture and to plug them into the language. These objects can be developed in C++ or Java code (a connector is available for Matlab as well). User-created objects can be run directly on the robot or transparently on a remote machine via CORBA technology. With these properties, URBI seems to us to be suitable for developing complex behaviors on robots as well as computationally intensive tasks as image processing.

For the demo simulation presented in the next sections, we used the simulated robot described above and we are going to use its real equivalent in our further research on the field of semantic SLAM. The task itself may be not perceived as being strictly specific for humanoid robots. However, the motivation to use a humanoid robots comes from the fact, that they are specially designed with the aim to interact with humans and to act in human-made environment. If they are already imitating humans in their physical form, why should not we enable them to do the same on the level of their software? The concepts we are exploiting here come from human approach to navigation and orientation in the space. Thus embedding such human inspired semantic SLAM capabilities onto a humanoid robotic platform seems pertinent to us.

[3] http://www.gostai.com/

5 Results

After having the image segmented, all segments are labeled and interpreted by a set of rules representing prior knowledge about objects. Following the mentioned rules segments can be even merged so that e.g. multiple fragments of floor partially occluded by objects laying on it are labeled as belonging to the same object of type "floor". Fig. 2 gives an example resulted from the left image (supposed as the original image acquired by robot's vision system). Fig. 3 depicts the intermediate steps of segmentation. This "semantic" information is subsequently used to approximate the actual distance of certain objects. Having an object of type "window", it is looked-up in a table containing usual sizes of different objects and once found the size information is used along with the pixel size of the object on the image and the field of view of the camera to compute the approximate distance of the window (see the right image in the Fig. 2). This is described by Eq. 6 (simplified for horizontal size only). The distance d to an object is the product of estimated real width w_{real} of the object and tangent of its width in pixels w_{px} on the image multiplied by fraction of the horizontal field of view φ and the width w_{image} of the image in pixels

$$d = w_{real} * \tan (w_{px} * \varphi / w_{image}) \quad (6)$$

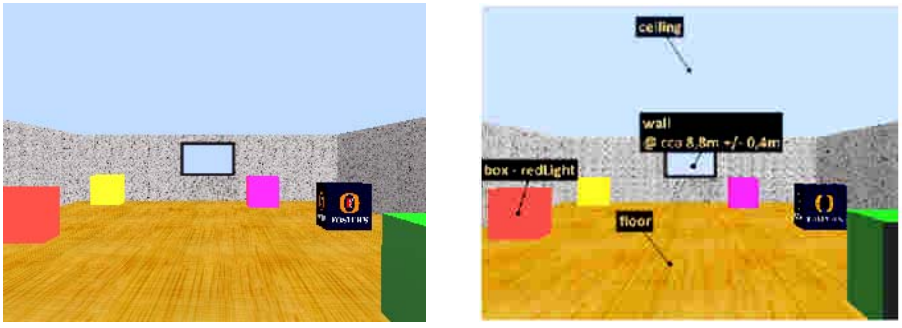


Fig. 2. A view of the robot's random walking sequence. The left image is the original one. The right image shows the result after the interpretation phase. Some of the detected objects are labeled. The opposing wall is labeled also with its approximate distance with respect to the robot.

The aim of this computation is absolutely not to infer the exact distance of an object, but rather to determine whether it is "far" or "near" in the context of the simulated world or if it is nearer to the robot in comparison to another object. This can help in the further process of creation of the map of the location.

This demo, however limited, gives a preliminary idea of principles of semantic mapping and more importantly, it gives a starting point for the research in the area of semantic SLAM. Resigning to precise metric position of every object in the mapped world and replacing it only by rough metric and human expressions like "near to" or "beside of" is believed to enable us to create faster and more robust algorithms for robotic SLAM. Using of "object landmarks" to navigate in an environment is certainly more meaningful that using e.g. simple points in case of classical SLAM. Knowing

the nature of an object gives an opportunity to distinguish between important and random objects. One can imagine a robot with an ability of choosing landmarks for purposes of its navigation by itself. With the knowledge about available objects, it could prefer to pick up the most important and stable objects that are unlikely to change their place or appearance in the lifetime of the map.

Fig. 3. The two segmentation steps relative to the result of Fig. 2.

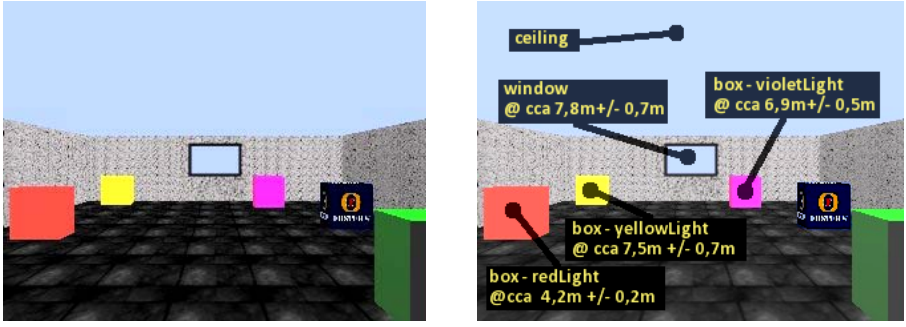


Fig. 4. The same room with different textures (left) and resulted interpreted image (right).

The seed point k_{seed} is determined as such a pixel from the skeleton whose distance from its closest contour pixel is maximal. Here, K is the set of pixels of skeleton belonging to segment S and C is the set of contour pixels of S. D_i denotes the minimal distance between the given pixel k_i and the contour. In this step, similar segments from the previous step are effectively merged. At this point, found segments may contain distinctive areas of different brightness having similar chroma. The ultimate step of the algorithm is in constructing a histogram of luminance values of each segment. The histogram is then polished by application of sliding average. If multiple significant clusters are found in the histogram, the segment is broken-up accordingly to separate them. Having finished this step, found segments are stored for further use.

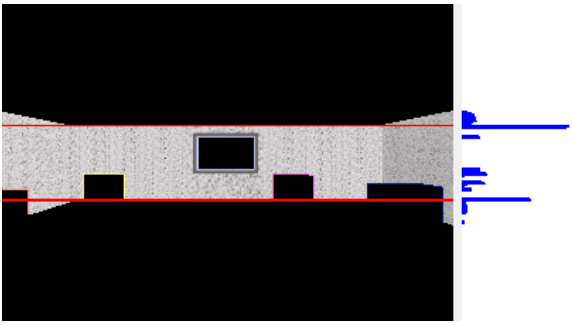


Fig. 5. Detection of the wall as a overall (macro) object in robot's environment.

It is important to notice the robustness of the proposed approach and the fact that the estimated distances, even if approximated, are relevant enough for extracting pertinent features relating environmental information. Fig. 4 gives results obtained from the right image showing the above-mentioned purpose. Fig. 5 gives an example of extended possibilities of the technique in detecting environmental information. In fact, it allows potentiality of a higher-level semantic labeling of objects constituting the robot's environment. Here, one can notice that in the given example (detection of the room's wall including the associated objects as window, etc…) allows the possibility to link previously labeled objects (for example the window) to the "wall" in term of "room's wall with the window".

6 Conclusions and Perspectives

Simultaneous localization and mapping is an important ability for an autonomous mobile robot. State of the art techniques have been discussed here giving an idea about the current state on the field of SLAM. In spite of a great advance in SLAM techniques in past years, most of the existing SLAM solutions can accommodate only a particular case or environment. A stable and generally usable SLAM solution is still missing. Given the state of the art of SLAM, one of the basic directions, which are expected to play a key role in future development of SLAM is so called "semantic SLAM": adding a semantic level into robotic mapping should help robots to go beyond simple "structural" information about the world that surrounds them. It should enable them to "understand" it.

In this paper, we identify the pertinence of semantic SLAM for the future development in mobile robotics and we present our initial research on this field. Our research is strongly inspired by the human way of navigation and place description. The semantic information about objects in the scene may improve mapping capabilities of robots. It should enable them to reason about their environment as well as to share their knowledge with humans and receive commands using human concepts and categories in a seamless way.

For description of a scene by semantic means, a good algorithm for image segmentation is an important starting point. Preferably, it should perform segmentation using both color and texture information. For real time use, fast and efficient algorithms are

required. A part of our future work will be dedicated to further development of such an algorithm. Another part of our future work will be focused on development of algorithms of semantic SLAM we outlined in this paper. They will be consequently implanted and verified in an indoor environment on the real Nao robot.

References

1. Durrant-Whyte, H., et al. Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms. Robotics and Automation Magazine, Vols. 13, No 2, pp. 99-110, (2006).
2. Durrant-Whyte, H., et al., Simultaneous Localisation and Mapping (SLAM): Part II State of the Art. Robotics and Automation Magazine, Vols. 13, No 3, pp. 108-117 (2006)
3. Muhammad, N., Fofi, D. and Ainouz, S. Current state of the art of vision based SLAM. Image Processing: Machine Vision Applications II, Proceedings of the SPIE, Vol. 7251, pp. 72510F-72510F, (2009)
4. Thrun, S. and Leonard, J. J. Simultaneous Localization and Mapping. [ed.] B. Siciliano and O. Khatib. Springer Handbook of Robotics. Berlin Heidelberg: Springer-Verlag, 37, (2008)
5. Hahnel, D., et al. Map Building with Mobile Robots in Dynamic Environments. Proceedings of the IEEE International Conference on Robotics and Automation. Taipei: IEEE, Vol. 2, pp. 1557-1563, (2003)
6. Nüchter, A., Hertzberg, J. Towards semantic maps for mobile robots. Robotics and Autonomous Systems. Amsterdam : North-Holland Publishing Co., Vol. 56, pp. 915-926, (2008)
7. Vasudevan, S., et al. Cognitive maps for mobile robots-an object based approach. Robotics and Autonomous Systems. Amsterdam: North-Holland Publishing Co., Vol. 55, pp. 359-371, (2007)
8. Galindo, C., et al. Multi-Hierarchical Semantic Maps for Mobile Robotics. Internati. Conf on Intelligent Robots and Systems (IROS 2005). Edmonton : IEEE, pp. 2278- 2283, (2005)
9. Persson, M., et al. Probabilistic Semantic Mapping with a Virtual Sensor for Building/Nature detection. International Symposium on Computational Intelligence in Robotics and Automation. Jacksonville : IEEE, pp. 236-242, (2007)
10. Ekvall, S., Jensfelt, P. and Kragic, D. Integrating Active Mobile Robot Object Recognition and SLAM in Natural Environments. International Conference on Intelligent Robots and Systems, 2006 IEEE/RSJ. Beijing: IEEE, pp. 5792-5797, (2006)
11. Meger, D., et al. Curious George: An attentive semantic robot. Robotics and Autonomous Systems. Amsterdam : North-Holland Publishing Co., Vol. 56, pp. 503-511, (2008)
12. Lucchese, L. and Mitra, S. K. Color image segmentation: A state-of-the-art survey. Proc. Indian Nat. Sci. Acad. (INSA-A). Vols. 67-A, pp. 207-221, (2001)

Author Index

Proceedings of ANNIIP 2010
6th International Workshop on
Artificial Neural Networks and Intelligent Information Processing
ISBN: 978-989-8425-03-4
http://www.icinco.org