Optimization of Robotic Mobile Agent Navigation

Sándor Tihamér BRASSAI[1,2], Barna IANTOVICS[2], Călin ENĂCHESCU[2]

[1] Petru Maior University,
str. Nicolae Iorga no. 1, Tirgu Mureş

[2] Sapientia University,
sos. Sighisoarei 1.C. Corunca/Tirgu Mures, Romania,
tiha@ms.sapientia.ro

Abstract: The traveling salesman problem (TSP) has many applications in economy, transport logic [1] etc. It also has a wide range of applicability in the mobile robot path planning optimization [2]. The paper presents research result of solving the path planning subproblem of the navigation of an intelligent autonomous mobile robotic agent. Collecting objects by a mobile robotic agent is the final problem that is intended to be solved. For the robotic mobile agent's path planning is used an unsupervised neural network that can find a closely optimal path between two points in the agent's working area. We have considered a modification of the criteria function of the winner neuron selection. Simulation results are discussed at the end of the paper. The next future development is the hardware implementation of the self-organizing map with real time functioning.

Keywords: robotic mobile agent; neural network; unsupervised learning; computational intelligence

1. Introduction

Robotic mobile agents have a wide range of applications in different areas [3], such as: access dangerous areas to humans, underwater explorations, monitoring the environment, painting and de-painting applications [26]. In our research, as mobile agent a robotic mobile agent is considered. The main properties of the mobile agent are: the intelligence in operation, autonomy, reactivity and mobility. Many scientists are working on finding new solutions for different subsections of robotic mobile agent and multi-agent [4] applications such as [3] navigation, localization, optimal path planning, path following [5] object detection, movement and modelling [6] of the mobile robots with multiple implementation solutions [7]. In this paper we will focus on path planning optimization of the mobile agent using neural networks.

We will give solutions for:

- a TSP and a modified TSP problem solving when the agent does not have to get back to the starting point.
- Finding a closely optimal path from the resolved TSP. For solving the TSP a Kohonen map was used with a proposed cost function in the winner neuron's selection. In the following sections, the TSP problem, the network structure and training, and results with the resolved TSP with Kohonen map and optimization of path planning between a starting node and a target node on the map will be presented.

The paper is organized as follows: Section 2 discuss the problem that we intend to solve and presents some existent solutions for TSP solving; in Section 3 our proposal for path finding is presented with preliminaries of self-organizing map architecture, a modified

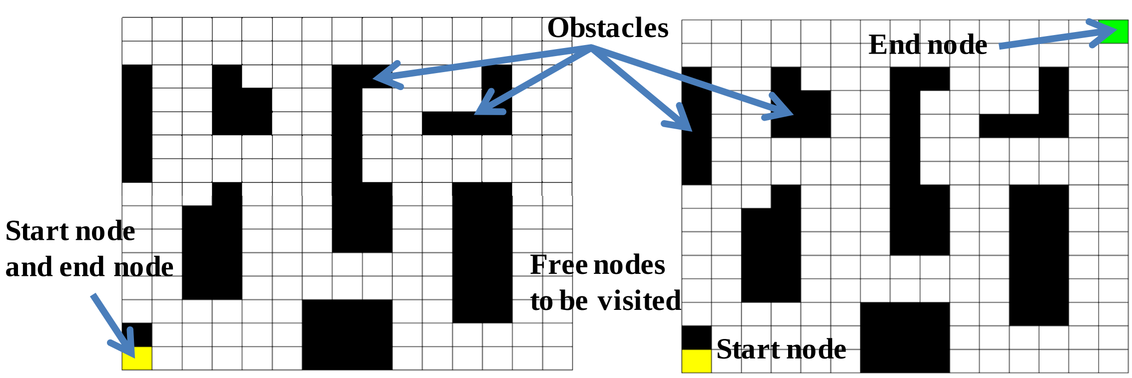


Figure 1. (a) represents the map, the agent finishes the work at the starting node; (b) the agent finishes the work at the end node.

solution for TSP solving, and the optimal path finding; Section 4 presents the conclusions of the research and the future research direction.

2. TSP Problem Formulation

In our research the following two tasks for a mobile agent were taken into consideration: In the first case the mobile agent has to cover (supervise) an area and to move back to the starting node (Figure 1. (a)). In the second case the mobile agent has to cover an area starting from one node and finishing the work in the end node (Figure 1.(b)). The second application can be used if we have a large area discomposed in subareas, and for each subarea the entering and finishing nodes are defined. The agent has to visit each node (marked in the figure with a white square) avoiding the obstacles and move back to the starting respectively to finish the task in the ending node. Each node must be visited only one time.

Solutions for TSP solving

Many algorithms for solving TSP were developed, like: combinatorial algorithms [2],[8],[9], branches and bounds [10], an efficient algorithm proposed by Clark and Wright [11], ant colony optimization algorithms [12], [13], particle swarm optimization algorithms [10], genetic algorithms [14].

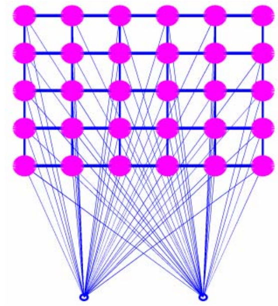
There are also several artificial intelligence based solutions for solving the TSP in mobile robots path planning such as genetic algorithms [14], [15], solutions based on artificial recurrent networks [16], [17], fuzzy clustering algorithms [18] etc.

3. Our Proposal for Path Finding Optimization

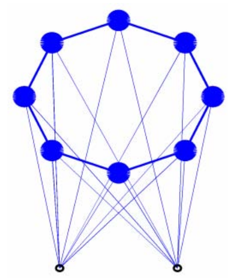
Preliminaries

A Kohonen map (self-organizing map - *SOM*) is an artificial neural network that uses an unsupervised training algorithm [19]. The output of a Kohonen map is processed as a linear combination of the network weights and the network inputs (equation (1)). The general structure of the network is presented in the Figure 2 (a), (b), and (c).

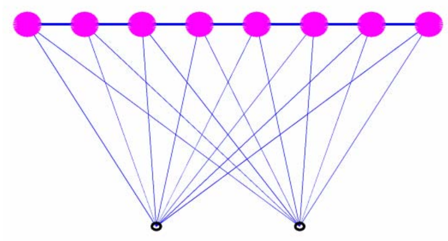
The network output is composed in general accordingly to the equation (1), but according to specific optimization applications, the network output can be processed using a cost function.



(a) Grid topology



(b) Ring topology



(c) Neurons placed along a line

Figure 2. The general structure of the network

$$y_i = \sum_{j=1}^{M} w_{ij} \cdot x_j \quad i = 1..N, \quad j = 1..M\,, \tag{1}$$

$$y_i = \|\overline{w}_i - x\| = \sqrt{\sum_{j=1}^{M} \left(w_{i,j} - x_j\right)^2}\,, \tag{2}$$

$$\begin{aligned} y_i &= \alpha\|\overline{w}_i - \overline{x}\| + (1-\alpha)\left(\|\overline{s} - \overline{x}\| + \|\overline{w} - \overline{x}\|\right) = \\ &= \alpha\sqrt{\sum_{j=1}^{M}\left(w_{i,j} - x_j\right)^2} + \\ &(1-\alpha)\left(\sqrt{\sum_{j=1}^{M}\left(s_j - x_j\right)^2} + \sqrt{\sum_{j=1}^{M}\left(t_j - x_j\right)^2}\right) \end{aligned} \tag{3}$$

$$w_{i,j}[k+1] = w_{i,j}[k] + \mu\Phi\left(r, r^*\right)\left(x_j - w_{i,j}[k]\right), \tag{4}$$

where y_i represents the network's *i*-th output, $w_{i,j}$-the network weight between the *i*-th processing element and the *j*-th input, *N* the number of the network's processing elements respectively M the number of inputs of the network. The self-organizing map uses a neighbourhood function to preserve the topological properties of the input space. The neurons of the self-organizing map are placed based on a topology. The topology can be linear, hexagonal, a two or three-dimensional grid type or also a random type topology (Figure 2). The selection of the most suitable topology corresponding to the input space is very important. To teach the organizing map, generally, an unsupervised learning algorithm is used. After processing the network's output, based on a criteria function, the winner processing element will be defined. The weights of the winner processing element and the ones of the processing elements in the neighbourhood of the winner are trained based on the Hebb (or anti Hebb) rule.

During the training process of the network, the neurons are organized according to the topology so that neurons with similar weights will be arranged closely to each other according to the topology. The Mexican hat is frequently used as a neighbourhood function, but several times the Gaussian function is considered.

In this paper multiple cost functions have been tested according to equations (2) and (3). In equation (2) the Euclidian distance is processed between the network input and the weights of the network. The network with the minimal value will be selected as the winner neuron. In the eq. (3) the cost function was extended with a penalization member. Parameter α determines the extent to which prevails one or the other part of the cost function. Two other types of cost functions have been tested. The equations are not presented here, but can be deduced very simply by changing the Euclidian norm with the Manhattan norm.

For resolving the TSP problem with the self-organizing network, the structure of the network is presented in Figure 2(b). The network neurons represent the nodes that the agent visits. A topology has to be defined so that it corresponds to the expectations of the TSP task. Each neuron can have two and only two neighbouring nodes. One is from which the agent arrives and the other is where the agent will be in the next step. If the agent needs to get back to the starting point, this means that the first and the last neuron are the same. It can easily be concluded that this is equivalent to a ring-type topology.

As mentioned, the neuron represents a node where the agent arrives, and the weights of the network represent the position of the node on to the navigation map of the agent. The network structure for the TSP in Figure 2(b) is presented. The TSP is resolved based on the classification of the inputs of the network. According to the normalized Hebb rule used for network training equation (4), weights are shifted towards the network current inputs. The neurons will be rearranged corresponding to the ring topology.

The neighbourhood function defines/influences weights for which neurons will be updated. The neighbourhood function value for the neuron close to the winner has a significant value close to 1 and the value of the neighbourhood function for neurons far from the winner will have an insignificant value close to zero, and will block the update of weights for these neurons. As neighbourhood function the Gaussian function was used, with the centre point of the function at the winner neuron index on the topology map. r and r^* represent the neuron positions on the topology map for the neuron for which the weight update is processed, respectively for the winner neuron (equation 4).

Modified TSP solving

A modified task of the TSP considered in this paper is when the agent does not have to move back to the starting point, the target point is previously defined on the map, where the agent has to finish the task (Figure 1(b)). The question is to solve this problem using the self-organizing map.

If the agent does not have to move back to the starting point, the ring type topology is not suitable. For solving the problem, a one dimensional topology is proposed to be used. The neurons are placed along a line (Figure 2 (c)).

Unlike as it was expected, the starting and destination node of the solution do not coincide with the initially specified starting and target nodes. From the path resulted with the self-organizing map the starting and destination points should be searched for. The result of the self-organizing map, the order of going through all off the nodes, is a vector with network weights. By finding the starting node and the target node index from the vector, the path from the starting node to the target node is solved, and from the starting node all of the nodes can be reached. In case of the ring type topology two paths exist to the target node. In case of the linear topology only a single path exists. These paths are not optimal, but with a simple searching algorithm a closely optimal solution can be found. In our research we have considered the cost function as the length of the path (the number of nodes).

The problem to be resolved is that the first neuron weights to converge to the starting node coordinates respectively the last neuron weights to converge to the destination node coordinates. The solution lies in the training algorithm. The proposed solution in this paper is to not update the first and last neuron weights and overwriting the first and last neuron weights with the starting and destination node coordinates. This simple amendment will result in that, that the starting and destination nodes of the solution will coincide with the initially specified start and target nodes.

Optimal path finding

All three forms of the network resolve the task to reach all the nodes on the map. The route from the starting node to the target node is resolved, but the solution is not optimal or close to the optimal. The proposed algorithm for finding a closely optimal route between the starting and the ending node is presented in the following:

```
Algorithm Route Finding
Input: W {weight vector resulted after network training}
Output: path {Closely Optimal Path}
N ← length(W)
W_new(1) ←W(1)
p←0
Step 1:  deleting duplicate nodes
for k in 2 to N
if dist(W(k)-W(k-1)) > ε, then
      W_new(k) ← W(k)
      p:=p+1
end if
end for
Step 2: finding start and end node indexes
Min_value=0;
For i in 1 to P
If min_value<dist(start_node_index-W_new(i)) Then
Min_value←dist(start_node_index-W_new(i))
                Start_nodex_index<-i
End if
End for
For i in 1 to p
If min_value<dist(end_node_index-W_new(i))
Min_value← dist(start_end_index-W_new(i))
end_node_index←i
End if
End for
Step 3: finding neighborhood nodes with lower cost to taget
Create table intermediar_table with records
current_node_index, neighborhood_node_index,
cost_from_start_to_current_node,
cost_from_neighborhoud_node_to_end_node
k:=1
for i in start_node_index to end_node_index
for j in  range neighborhood_node(i)
            if cost(j,end)<cost(i,end) then
            intermediar_table(k,1) ←i
            intermediar_table(k,2) ←j
            intermediar_table(k,3) ←
cost(start_node_index,i)
            intermediar_table(k,4) ←-
cost(j,end_node_index)
            k=k+1
end if
end for
end for
Step 4: ordering the intermediate_table
order intermediar_table by column(3) asccending;
Step 5: deleting overlapped sections
for i in 1 to length(intermediar_table)
for j in i+1 to length(intermediar_table)
if
overlap(range(intermediar_table(i,1),intermediar_table(i,
2)) range(intermediar_table(j,1),intermediar_table(j,2))
then
delete intermediate_table_row(j)
end if
end for
end for
Step 6: extracting the route
j:=1;
for i in 1 to length(intermediate_table)
for k in intermediate_table(i,1) to intermadiate_table(i,2)
path(j) ←W_new(k)
j:=j+1
end for;
end for;
return path
EndRouteFinding
```

In the first step the duplicate nodes were deleted. If the number of neurons is equal to the number of nodes on the map, the network does not find the solution, because a set of neurons will not be active during the network training process.

Finding the indices from the resulted weight vector with the starting and ending nodes is realized in step two. The vector values which correspond to node coordinates on the map are compared with start and end node positions.

In step three are found the nodes for which a neighbourhood node with a lower cost to the target node exists.

Calculating the cost from the start to the end node and from the neighbourhood nodes to the end node for each node found in the previous step and storing the results in a table with the following fields: current node index, neighbourhood node index, and the calculated costs.

Ordering the table in ascending order according to the cost column (column number three) is realized in the next step of the algorithm.

The rows from the table for which the detected section with a higher cost overlaps the section with a lower cost are deleted.

In the final step the closely optimal path from the start to the end point is extracted.

Figures 3, 4 and 5 are present results of the self-organizing map training for solving the TSP and modified TSP .

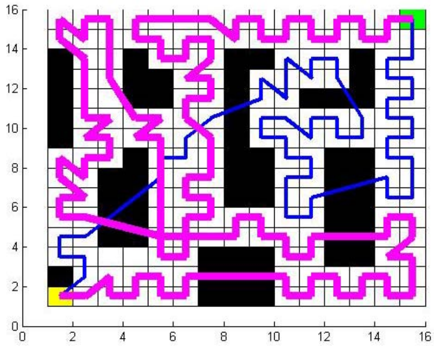


Figure 3. Network training results using a ring type topology

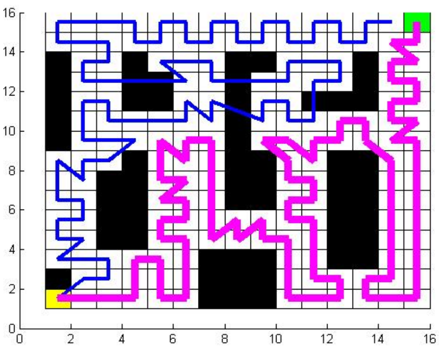


Figure 4. Results using linear type placement

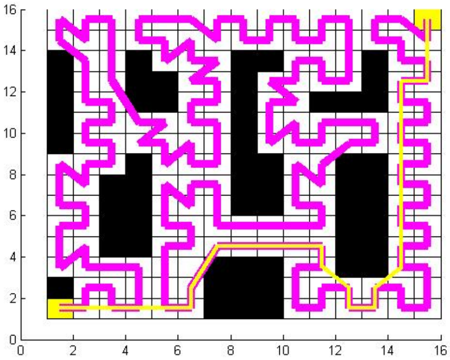


Figure 5. Results using linear type placement with overwriting of the first and last neuron weights with the start and end node coordinates

In Figure 3 as network topology a ring-type topology was used. The neurons were positioned on a circle. In Figure 4 and Figure 5 a linear placement was used and in Figure 5 the first and last neuron's weights were forced (overwritten) with the start and end node coordinates, as the result is how it was expected. In Figures 6, 7 and 8 the evolution of the training process is presented in different training cycles. A neighbourhood degree was gradually decreased during the training cycles. In the first phase a large scale rearrangement of the neuron was allowed, finally reducing to the immediate vicinity of the winner.

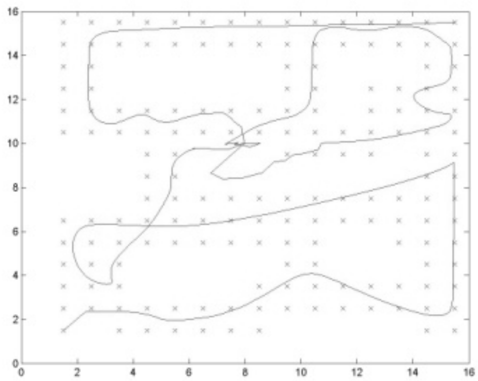


Figure 6. Network training results using a ring type topology

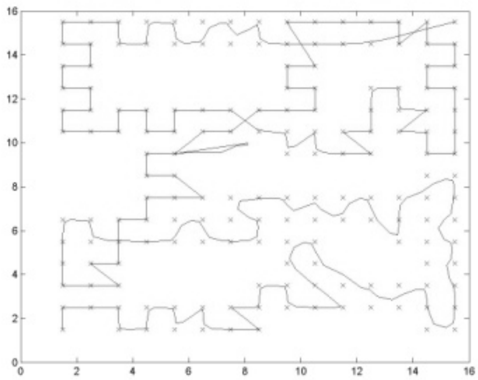


Figure 7. Example of one-column width figure

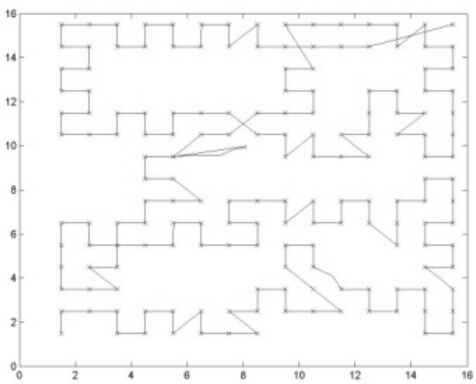


Figure 8. Example of one-column width figure

For 167 input nodes, for all of three variants of network topology (Figure. 2) a number of 1000 neurons were used. The weight update was processed based on the normalized Hebb rule. For the presented experiment a value of 0.7 of the training factor and as neighbouring function the Gaussian were used. The standard deviation of the Gaussian function was gradually decreased narrowing the weight update to the closely neighbour of the winner neuron.

In the following figures the evolution of certain weights corresponding to the first input (Figure. 9) and the second input (Figure 10) during network training results for ring type topology respectively in Figure 12 and Figure 13 for line type topology with fixed start and end node are presented. From the evolution of the weight the advancement of the training process can be concluded. On the figures, for 10 of the 1000 neurons the weight values are plotted. In case of supervised learning, the training process can be stopped based on the advancement of the training error. In case of unsupervised training the error cannot be calculated. A criteria function for detecting the end of the network training process, which characterizes the evolution of the training, is presented in equations 5 and 6. In equation 5 the criteria for the weight corresponding to the first and second input of the network are processed separately. According to the criteria values, when the criteria value is close to zero the training of the network can be stopped.

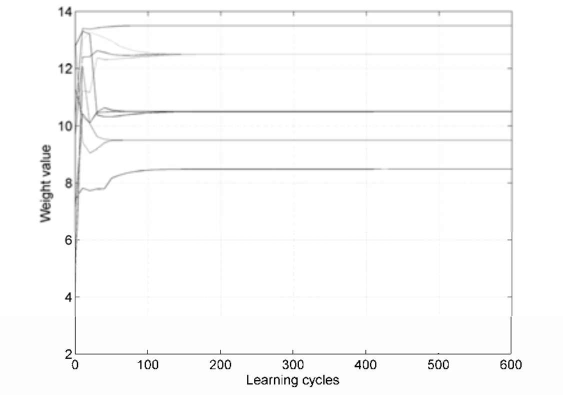


Figure 9. Evolution of certain weights corresponding to the first input during network training using a ring type topology

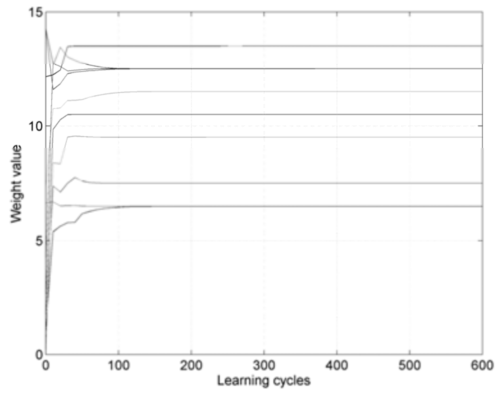


Figure 10. The evolution of certain weights corresponding to the second input during network training a ring type topology

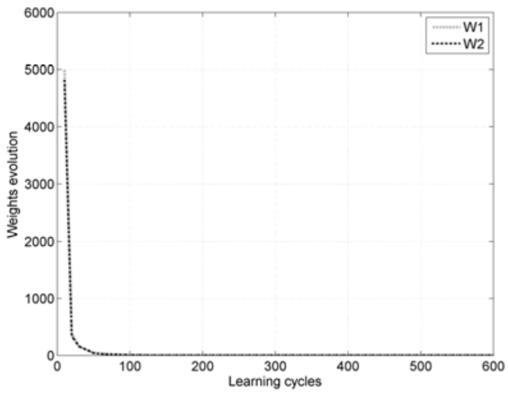


Figure 11. Criteria function evolution (eq. 5) of the weights corresponding to the first and second input during network training using a ring type topology

$$q_j[k] = \sum_{i=1}^{N} \left| w_{i,j}[k] - w_{i,j}[k-1] \right| \quad j = 1..M\,, \tag{5}$$

$$q[k] = \sum_{j}^{M} \sum_{i=1}^{N} \left| w_{i,j}[k] - w_{i,j}[k-1] \right|, \tag{6}$$

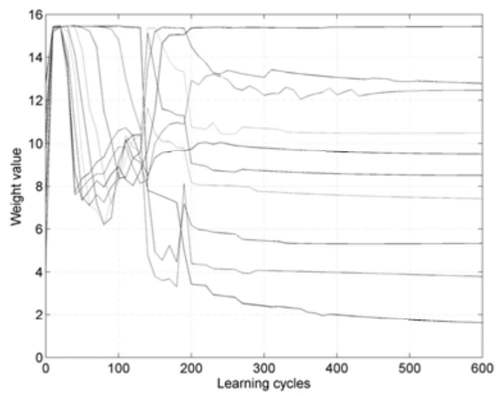


Figure 12. Evolution of certain weights corresponding to the first input during network training using a linear type topology with fixed start and end nodes

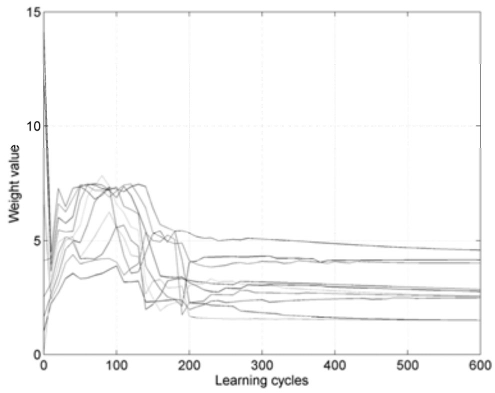


Figure 13. Evolution of certain weights corresponding to the second input during network training using a linear type topology with fixed start and end node

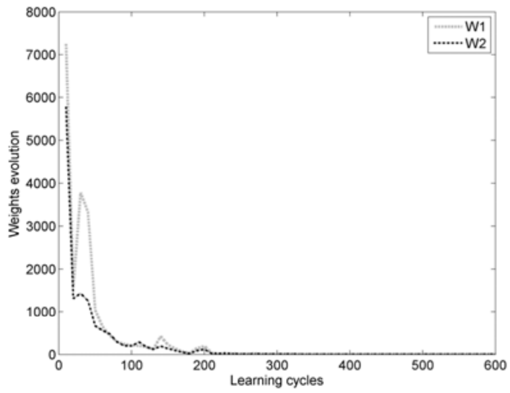


Figure 14. Criteria function evolution (equation 5) of the weights corresponding to the first and second input during network training using a linear type topology with fixed start and end node

In the table**s** from the annex the advancement of the weights values corresponding to the three variants: ring type, linear type and linear type with fixed start and end node topology are presented.

In tables 1 and 2 the weight values for the ring type topology, in table**s** 3 and 4 for the linear type topology respectively in table**s** 5 and 6 for the linear type topology with fixed start and end node are presented. In the first column of each table appears the number of training cycles, in the other columns every one hundredth weight value. In the case of the linear type topology with fixed start and end node there is a need for more training cycles than in the other cases, in the same circumstances. Evolution of the weights presented in the tables also illustrates the cycle when the network training was completed.

4. Conclusions

In this paper the TSP problem solving was discussed, using the self-organizing map with applicability in robotic agents' application which represents a subtask of the mobile robot navigation.

A multiple criteria function was proposed for the winner neuron selection. All four used criteria functions work well during the training. If the penalty part of the cost function prevails, the number of learning cycles increases. At the tuning phase it must be taken into account that the teaching is started with neighbouring values high enough. If the neighbouring degree is low, most of the nodes will not be part of the solution. The calculation of distance with the Euclidean and Manhattan norms should be completed with the infinite norm and tested in the future. Using a simplified cost function, the complexity of the output processing is reduced, reducing the network output processing time.

The achieved results are promising in terms of solving the path planning subtask of the robotic mobile agent navigation.

The research results based on the multiple type of artificial neural networks (Radial Basis Function ANN [20], [21] Cerebellar Model Articulation Controller [22], [23], [24]) implementation in FPGA circuits [25] can be used in the Kohonen network hardware implementation. The future research objective is the implementation of parallel pipeline architecture in hardware with real time functionality of the Kohonen network.

Acknowledgment

This paper is a result of the project `Transnational Network for Integrated Management of Postdoctoral Research in Communicating Sciences. Institutional building (postdoctoral school) and fellowships program (CommScie)" - POSDRU/89/1.5/S/63663, financed under the Sectorial Operational Programme Human Resources Development 2007-2013.

REFERENCES

1. FILIP, E., M. OTAKAR, **The Travelling Salesman Problem and its Application in Logistic Practice**, WSEAS Transactions on Business and Economics, Issue 4, Volume 8, 2011, pp. 163-173
2. RAJA, R., S. PUGAZHENTHI, **Optimal Path Planning of Mobile Robots: A Review**, International Journal of Physical Sciences, 2012, pp. 1314-1320
3. SIEGWART, R., I. R. NOURBAKHSH, D. SCARAMUZZA, **Introduction to Autonomous Mobile Robots**, The MIT Press Cambridge, Massachusetts, London, England, 2011.
4. HERNANDEZ-MARTINEZ, E. G., E. ARANDA-BRICAIRE, **Decentralized Formation Control of Multi-agent Robot Systems based on Formation Graphs**, Studies in Informatics and Control, vol. 21(1), 2012, pp. 7-16.
5. CHATTI, A., P. BORNE, M. BENREJEB, **On the Use of Neural Techniques for Path Following Control of a Car-like Mobile Robot**, Studies in Informatics and Control, Vol. 14(4), 2005, p. 221.
6. LATORRE, H., K. HARISPE, R. SALINAS, G. H. LEFRANC, **Ontology Model of a Robotics Agents Community**, Int. J. of Comp., Comm. & Ctrl., ISSN 1841-9836, Vol. VI (1), 2011, pp. 125-133.
7. BRAUNL, T., **Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems** Springer-Verlag Berlin, Heidelberg, New York, 2003.
8. MATAI, R., S. P. SINGH, M. L. MITTAL, **Travelling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches**, InTech, 11/2010; ISBN: 978-953-307-426-
9. PARKER, L. E., **Path Planning and Motion Coordination in Multiple Mobile Robot Teams**, Encyclopedia of Complexity and System Science, Springer, 2009.
10. JIANN-HORNG, L., H. LI-REN, **Chaotic Bee Swarm Optimization Algorithm for Path Planning of Mobile Robots**, Proc. 10th WSEAS Intl. Conf. on Ev. Comp., Prague, Czech Republic, 2009, pp. 84-89.
11. PICHPIBUL, T., R. KAWTUMMACHAI, **New Enhancement for Clarke-Wright Savings Algorithm to Optimize the Capacitated Vehicle Routing Problem,** Eur. J. of Sc. Res., Vol. 78(1), 2012, pp.119-134, ISSN 1450-216X
12. ZHAO, J., X. FU, Y. JIANG, **An Improved Ant Colony Optimization Algorithm for Mobile Robot Path Planning**, 3rd Intl. W-shop on Intell. Sys. and App. (ISA), 28-29 May 2011.
13. PINTEA, C. M., P. C. Pop, D., DUMITRESCU, **An Ant-based Technique for the Dynamic Generalized Travelling, Salesman Problem**, Proc. 7th WSEAS Intl. Conf. on Sys. Theory and Sc. Comp., Athens, Greece, Aug., 2007, pp. 255-259
14. NOUARA A., C. MOHAMED, **Mobile Robots Path Planning using Genetic Algorithms**, ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems, pp 111-115
15. KAO-TING, H., L. JING-SIN, ,C. YAU-ZEN, **A Comparative Study of Smooth Path Planning for a Mobile Robot by Evolutionary Multi-objective Optimization**, Proc. 2007 IEEE Intl. Sym. on Comp. Intel. in Rob. and Autom.
16. YOGITA G.,G. KUSUM, **Artificial Intelligence in Robot Path Planning**, Intl. J. of Soft Comp. and Eng. (IJSCE), Vol. 2(2), 2012.
17. SIVARAM, K. M. P.,S. RAJASEKARAN, **A Neural Network based Path Planning Algorithm for Extinguishing Forest Fires**, IJCSI Intl. J. of Comp. Sc. Iss., Vol. 9(2), No 2, 2012.
18. SITI, N., A. P. PUTRA, RENDYANSAH, **Intelligent Navigation in Unstructured Environment by using Memory-Based Reasoning in Embedded Mobile Robot**, Eur. J. of Sc. Res. ISSN 1450-216X Vol. 72(2), 2012, pp. 228-244.
19. KOHONEN, T., **Self-Organizing Maps, Third Edition**, Springer, 2001,
20. BRASSAI, S. T., L. BAKÓ, G. PANA, Şt. DAN, **Neural Control Based on RBF Network implemented on FPGA,** 11th Intl. Conf. on Opt. of El. and Elec. Eq., OPTIM 2008, pp.41-46.

21. BRASSAI, S. T., L. BAKÓ, **Visual Trajectory Control of a Mobile Robot Using FPGA Implemented Neural Network**, Pollack, Intl. J. for Eng. and Inf. Sc., Vol. 4(3), 2009, pp. 129-142.

22. BRASSAI, L. D., L. BAKÓ, **Hardware Implementation of CMAC based Artificial Network with Process Control Application**, Trans. on El. and Comm., Sc. Bul., „Politehnica” Univ. of Timisoara, 2004, p209-213, ISSN 1583-3380.

23. BRASSAI, S. T., L. BAKÓ, Şt. DAN, **FPGA Parallel Implementation of CMAC Type Neural Network with on Chip Learning**, SACI 2007, Budapest Tech, Hungary, 2007, 111-115, ISBN: 142441234X

24. BRASSAI, S. T., L. BAKÓ, **Hardware Implementation of CMAC Type Neural Network on FPGA for Command Surface Approximation**, J. of Ap. Sc. Budapest Tech Hungary, Vol. 4, No. 3, 2007, ISSN 1785-8860

25. BRASSAI. S. T. , **Neuroadaptive Systems Based on FPGA Circuits with Application in Automatic Control**, Phd thesis at Transilvania University from Brasov, 2008 Brassai Macro Conference.

26. BRAMSON, B., **New Applications for Mobile Robots**, Robotics Online, 2011

Annex

Table 1. Weight evolution corresponding to first input (ring type topology)

Training cycle	Weights corresponding to first input (ring top)									
	W_1	W_{100}	W_{200}	W_{300}	W_{400}	W_{500}	W_{600}	W_{700}	W_{800}	W_{900}
1	5.6941	3.7343	10.762	13.7838	9.4586	6.4059	1.8799	11.1667	4.4094	6.7183
40	4.5134	10.471	6.6255	15.3871	13.6954	6.5042	10.5527	8.1619	3.6613	2.1804
70	4.5	10.4993	6.5058	14.9292	13.506	6.5	10.5001	8.099	3.5047	2.0993
100	4.5	10.5	6.5	14.7293	13.5	6.5	10.5	8.0431	3.5	2.0431
130	4.5	10.5	6.5	14.5943	13.5	6.5	10.5	8.0175	3.5	2.0175
160	4.5	10.5	6.5	14.5297	13.5	6.5	10.5	8.0088	3.5	2.0088
190	4.5	10.5	6.5	14.5073	13.5	6.5	10.5	8.0066	3.5	2.0066
220	4.5	10.5	6.5	14.5014	13.5	6.5	10.5	8.0062	3.5	2.0062
250	4.5	10.5	6.5	14.5002	13.5	6.5	10.5	8.0061	3.5	2.0061
280	4.5	10.5	6.5	14.5	13.5	6.5	10.5	8.0061	3.5	2.0061
310	4.5	10.5	6.5	14.5	13.5	6.5	10.5	8.0061	3.5	2.0061

Table 2. Weight evolution corresponding to second input (ring type topology)

Training cycle	Weights corresponding to second input (ring top)									
	W_1	W_{100}	W_{200}	W_{300}	W_{400}	W_{500}	W_{600}	W_{700}	W_{800}	W_{900}
1	4.786	5.7963	2.0015	9.0381	14.7819	14.3306	5.4672	12.5902	0.4029	9.7687
40	2.3793	12.3375	15.2857	14.8926	6.0106	7.142	5.1014	1.5368	10.6436	6.4892
70	2.466	12.4199	15.3459	14.4936	5.8092	7.0536	4.8108	1.5004	10.5046	6.4999
100	2.4969	12.4812	15.4211	14.4992	5.6229	7.0175	4.6229	1.5	10.5	6.5
130	2.4999	12.4977	15.4733	14.5	5.5348	7.0091	4.5348	1.5	10.5	6.5
160	2.5	12.4998	15.4933	14.5	5.5079	7.008	4.5079	1.5	10.5	6.5
190	2.5	12.5	15.4983	14.5	5.5019	7.0079	4.5019	1.5	10.5	6.5
220	2.5	12.5	15.4994	14.5	5.5007	7.0079	4.5007	1.5	10.5	6.5
250	2.5	12.5	15.4996	14.5	5.5004	7.0079	4.5004	1.5	10.5	6.5
280	2.5	12.5	15.4997	14.5	5.5004	7.0079	4.5004	1.5	10.5	6.5
310	2.5	12.5	15.4997	14.5	5.5004	7.0079	4.5004	1.5	10.5	6.5

Table 3. Weight evolution corresponding to first input (linear type topology)

Training cycle	Weights corresponding to first input linear top									
	W_1	W_{100}	W_{200}	W_{300}	W_{400}	W_{500}	W_{600}	W_{700}	W_{800}	W_{900}
1	5.6941	3.7343	10.762	13.7838	9.4586	6.4059	1.8799	11.1667	4.4094	6.7183
40	15.0972	7.6873	2.1464	7.9492	6.8128	2.4596	8.9911	15.0579	7.514	15.2665
70	15.1727	5.5883	1.9869	8.3534	3.8266	6.5182	3.3603	9.7427	15.3634	14.0279
100	15.3682	5.8242	2.4851	6.6486	3.715	8.3901	2.4234	5.882	12.4805	14.8329
130	15.2291	6.3704	2.6008	7.2907	3.5754	9.3834	3.3657	6.8084	11.4624	15.47
160	15.2409	6.4944	2.6345	7.5471	3.4998	9.3965	3.4077	7.4375	11.2079	15.4614
190	15.2946	6.5304	2.562	7.5005	3.4726	9.408	3.4083	7.3658	11.0088	15.4996
220	15.3443	6.6745	2.5247	7.477	3.4952	9.4252	3.4228	7.2732	10.8516	15.4999
250	15.3742	7.1753	2.6118	7.4724	3.496	9.4438	3.4362	7.206	11.3159	15.5
280	15.3954	7.4884	2.5661	7.4761	3.4979	9.4606	3.4489	7.1381	11.4903	15.5
310	15.4147	7.4992	2.5365	7.4822	3.4992	9.4741	3.4608	7.1141	11.4951	15.5

Table 4. Weight evolution corresponding to second input (linear type topology)

Training cycle	Weights corresponding to first input									
	W_{1}	W_{100}	W_{200}	W_{300}	W_{400}	W_{500}	W_{600}	W_{700}	W_{800}	W_{900}
1	4.786	5.7963	2.0015	9.0381	14.7819	14.3306	5.4672	12.5902	0.4029	9.7687
40	15.3826	15.3463	14.7741	13.8128	11.5091	11.4412	11.0032	11.6343	5.7967	2.2433
70	15.4303	15.5	13.6102	11.2637	10.1007	5.7031	4.6192	4.5223	11.5664	2.6679
100	15.1576	15.5	13.8502	10.9903	10.4719	6.8125	6.3098	2.2501	11.7502	2.0151
130	15.458	15.5	14.3828	11.3444	10.4957	6.7721	5.3131	2.0572	10.0533	2.7934
160	15.4669	15.5	14.4762	11.2228	10.4996	6.7364	5.1719	1.5747	9.9481	3.8436
190	15.4815	15.5	14.49	11.29	10.5	6.6506	5.1505	1.5381	8.8686	3.0986
220	15.4915	15.5	14.4967	11.3313	10.5	6.5873	4.832	1.5214	8.8967	3.0144
250	15.4963	15.5	14.4995	11.3586	10.5	6.5459	4.7619	1.5108	9.7054	2.9496
280	15.4985	15.5	14.4999	11.3793	10.5	6.522	4.6995	1.505	9.5827	2.894
310	15.4994	15.5	14.5	11.3972	10.5	6.5096	4.6464	1.502	9.5559	2.8424

Table 5. Weight evolution corresponding to first input (linear type topology with fixed start and end node)

Training cycle	Weights corresponding to first input									
	W_{1}	W_{100}	W_{200}	W_{300}	W_{400}	W_{500}	W_{600}	W_{700}	W_{800}	W_{900}
1	5.6941	3.7343	10.762	13.7838	9.4586	6.4059	1.8799	11.1667	4.4094	6.7183
70	1.5	8.6642	14.9179	15.3683	10.4293	6.357	2.2908	3.1545	7.4041	12.092
130	1.5	10.4087	15.3288	12.9871	10.4378	6.1834	3.272	1.632	7.0823	11.7415
190	1.5	9.5657	15.3074	12.7897	10.4972	7.4531	3.4529	1.5153	6.9384	11.5615
250	1.5	9.8068	14.9712	12.6385	10.4998	7.4733	3.5003	1.5006	6.8079	11.5084
310	1.5	9.6289	14.8487	12.5526	10.5	7.4913	3.4998	1.5	6.6992	11.5007
370	1.5	9.5515	14.748	12.5159	10.5	7.498	3.5	1.5	6.6177	11.5
430	1.5	9.5204	14.6663	12.5039	10.5	7.4997	3.5	1.5	6.5641	11.5
490	1.5	9.5071	14.6042	12.5008	10.5	7.5	3.5	1.5	6.5338	11.5
560	1.5	9.5022	14.5609	12.5001	10.5	7.5	3.5	1.5	6.5189	11.5
610	1.5	9.5006	14.5331	12.5	10.5	7.5	3.5	1.5	6.5124	11.5

Table 6. Weight evolution corresponding to second input (linear type topology with fixed start and end node)

Training cycle	Weights corresponding to first input									
	W_{1}	W_{100}	W_{200}	W_{300}	W_{400}	W_{500}	W_{600}	W_{700}	W_{800}	W_{900}
1	4.786	5.7963	2.0015	9.0381	14.7819	14.3306	5.4672	12.5902	0.4029	9.7687
70	1.5	7.1013	5.7711	12.164	10.6822	11.2775	11.887	15.4116	15.3259	14.9019
130	1.5	7.3279	8.5055	10.4981	11.9371	7.538	10.6921	15.3122	15.4758	14.6988
190	1.5	5.5375	7.6104	10.4991	12.3482	7.2816	10.6023	15.1996	15.4987	14.5587
250	1.5	5.1049	7.4946	10.4999	12.3099	7.1437	10.8195	15.1951	15.5	14.5084
310	1.5	5.1715	7.4984	10.5	12.3358	7.1096	10.6949	15.211	15.5	14.5007
370	1.5	5.1063	7.4998	10.5	12.3787	7.0792	10.6057	15.2441	15.5	14.5
430	1.5	5.0362	7.5	10.5	12.4206	7.0549	10.5506	15.2867	15.5	14.5
490	1.5	4.9714	7.5	10.5	12.4535	7.0366	10.5214	15.3298	15.5	14.5
560	1.5	4.9114	7.5	10.5	12.4752	7.0237	10.508	15.3654	15.5	14.5
610	1.5	4.8554	7.5	10.5	12.4876	7.0149	10.5027	15.3898	15.5	14.5