Neural Labyrinth Robot –
Finding the Best Way in a Connectionist Fashion

Marvin Oliver Schneider[1], **João Luís Garcia Rosa**[1]

[1]Mestrado em Sistemas de Computação – Pontifícia Universidade Católica de Campinas
Rodovia D. Pedro I, km. 136, Caixa Postal 317, CEP 13012-970, Campinas-SP, Brasil.

moschneider@ig.com.br, joaoluis@puc-campinas.edu.br

Abstract. *The system "Neural Labyrinth Robot" is a connectionist approach to robot guidance in small experimental labyrinths. It is based on a two-layer perceptron, which learns the correct path from a symbolic algorithm. Even when the robot comes along to unknown situations, the neural network obtains satisfying results. The system has proven to be very efficient in its environmental tests, so it can provide a valuable contribution to nowadays' path finding systems.*

1. Introduction

Perhaps in analogy to human development that also starts with locomotion and only later comes to other features, several studies on robotics have taken correct movement in space as a goal: from animal mimics [Benyus, 1997] to futuristic walking machines (e.g. [Filiat *et al.*, 1999]) and even agents with complex tasks [Simmons *et al.*, 1997].

One of the main problems concerning computational resources consumption is the perception of the environment in its several constituents: stationary and moving objects, light, odor, temperature as well as abstract items like the cause of an action or – regarding this application – the best way from one point to another (e.g. [Satchok, 1997] and [Auf der Heide and Vöcking, 1995]).

The project "Neural Labyrinth Robot" intends to be a contribution to the perception problems research, offering a simple but efficient algorithm of learning the best way in a simulated labyrinth.

2. Description of the System

2.1. Basics

The system "Neural Labyrinth Robot" is a system that simulates a robot in an 8 by 8 matrix. Each position may contain an obstacle, the destination or be simply an empty space.

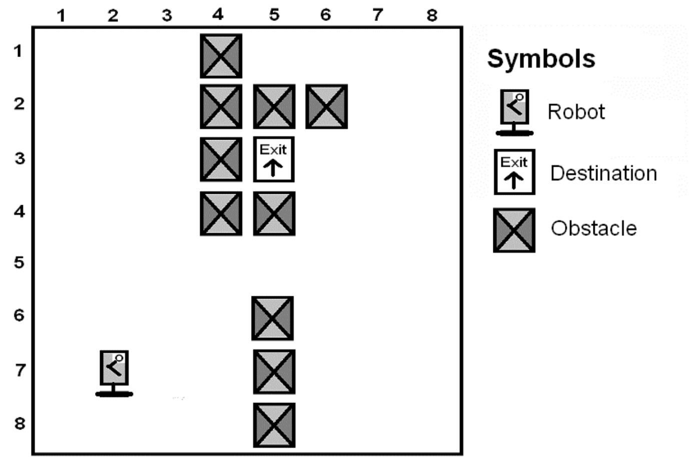


Figure 1: An example of a simple labyrinth

The robot's goal is to find its way without any external help from its starting point up to its destination.

"Neural Labyrinth Robot" employs a symbolic recursive algorithm to establish an analysis of the current labyrinth in order to teach the results to an artificial neural network.

2.2. Preprocessing

The system makes use of preprocessing routines. The scheme may be called by its steps of execution: "*vision – turning-points – move!*".

- *Vision*: The robot makes use of an artificial vision device, so it can go to all directions, providing that there are no walls in the path, because it is not allowed to see through walls. The labyrinth is completely unknown to the robot when it is inserted into it. As it moves and establishes vision throughout the maze, it automatically remembers everything discovered previously. This feature is not difficult to implement computationally, considering that it is one of the most important strategies of the symbolic algorithm to find the best way in a recursive manner.
- *Turning-points*: After establishing vision in its current situation, the robot automatically detects possible turning points. These are points that imply in a change of direction, making the robot move into an area not covered by its current vision or memory.
- *Move!*: The path of the robot is computed by its turning points, meaning that a simple straight path is not explicitly treated because it is considered simple to conceive with a real robot. The system also assumes that the destination point is achieved when the robot finds it, because moving towards it means simply adjusting the direction to it.

2.3. Symbolic Teaching Algorithm

The symbolic teaching algorithm finds the best way in a recursive manner, i.e., a tree diagram is formed, all eventual solutions are considered and the shortest way is achieved. Starting with the initial situation, all possible next turning points are considered, which themselves lead to all of the next turning points etc. until finding the destination. The fact that each path is described by its turning points comes in handy, because considering every single step would not only be useless, but also mean very deep recursion levels and stack overflows.

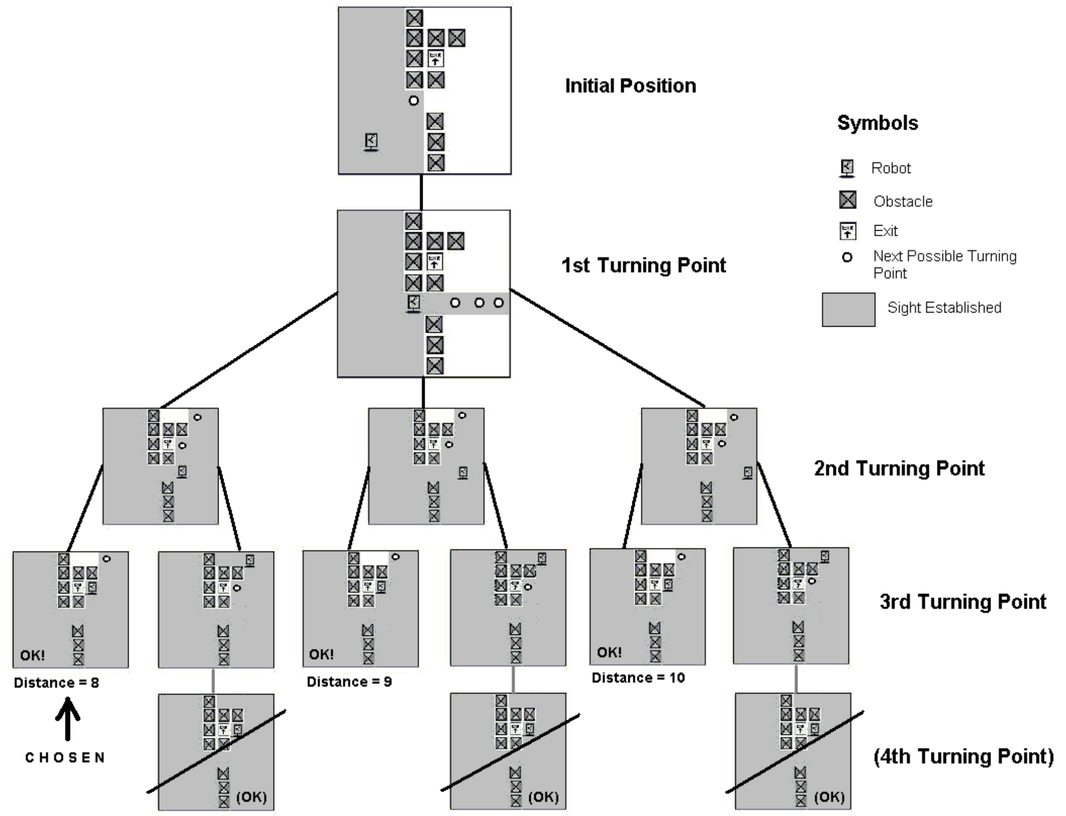


Figure 2: An example of processing using the symbolic algorithm

Throughout the recursion a maximum depth is considered. This depth is void until the first way to the destination is found. Afterwards, all ways having more turning points and thus representing a greater depth of recursion are cut off (as shown in figure 2, where all possibilities with 4 turning points are not considered after having found a path with only 3 turning points first).

Although there might be paths with fewer turning points being actually longer than some with more turning points, turning points instead of total distance are considered because of:

- The computational cost: every turning point means one more level of recursion (see figure 2) and a possible risk of memory problems (stack overflow etc.).

- General simplicity: human beings seem to prefer straight ways in unknown situations because they are much easier to remember – and this applies not only to humans but also to computational systems!

However, in order to decide between paths with the same number of turning points, the total distance is evaluated and the path with the smallest distance chosen (as shown in figure 2).

As turning points describe turns into ways that have not been explored yet, the robot keeps exploring until it finds the destination. If the entire labyrinth has been explored, there are no more turning points to reach, i.e. no more alternatives to go on and thus the algorithm detects that there is no solution (this happens only in labyrinths without any destination point or without any possible path to it!).

2.4. Neural Network

“Neural Labyrinth Robot” uses an artificial neural network, which takes hold of the knowledge being transmitted by the symbolic algorithm during training.

Topology: “Neural Labyrinth Robot” uses a basic two-layer perceptron structure with backpropagation as its learning algorithm. This topology was chosen because of its simplicity and computational efficiency [Haykin, 1999].

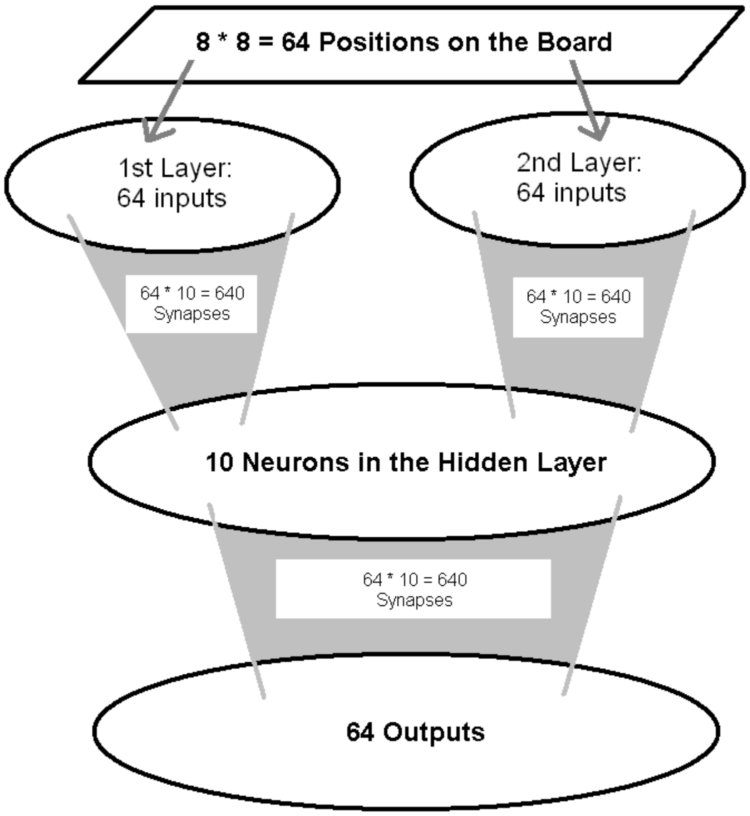


Figure 3: The Network Topology used in “Neural Labyrinth Robot”

Initialization: As a preliminary step, the synapses of the neural network are initialized with small random values between –0.1 and 0.1.

Processing: In the beginning of the neural network processing, the current position on the board is first mapped into the two input layers. This procedure (described below in detail) is performed because handing over the plain board to the net produces the following problems. Firstly, consider figure 4.

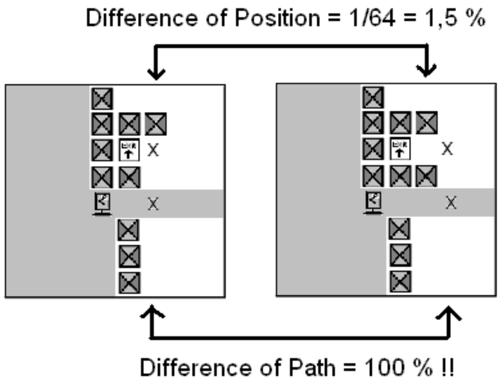

Difference of Position = 1/64 = 1,5 %
X
X
X
X
Difference of Path = 100 % !!

Figure 4: Problem with similar positions and different paths

The problem displayed by figure 4 is the reason that the network starts to oscillate when trying to treat similar labyrinths with slightly different (or even completely different) solutions in a straightforward manner (simply passing the situation on the board unprocessed to the network). The difference at the input-level is not significant enough to justify the desired difference at the output.

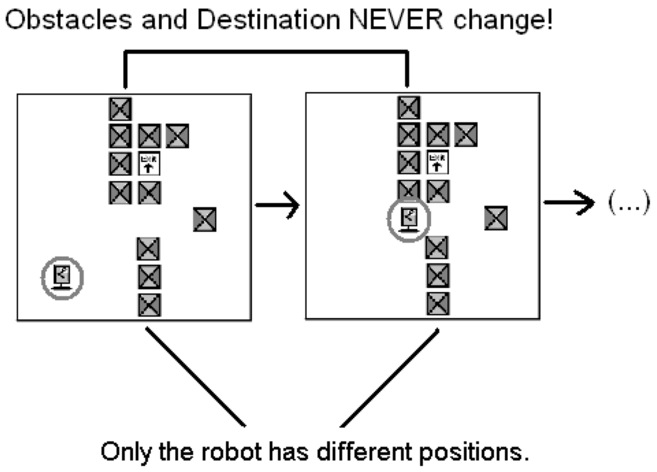
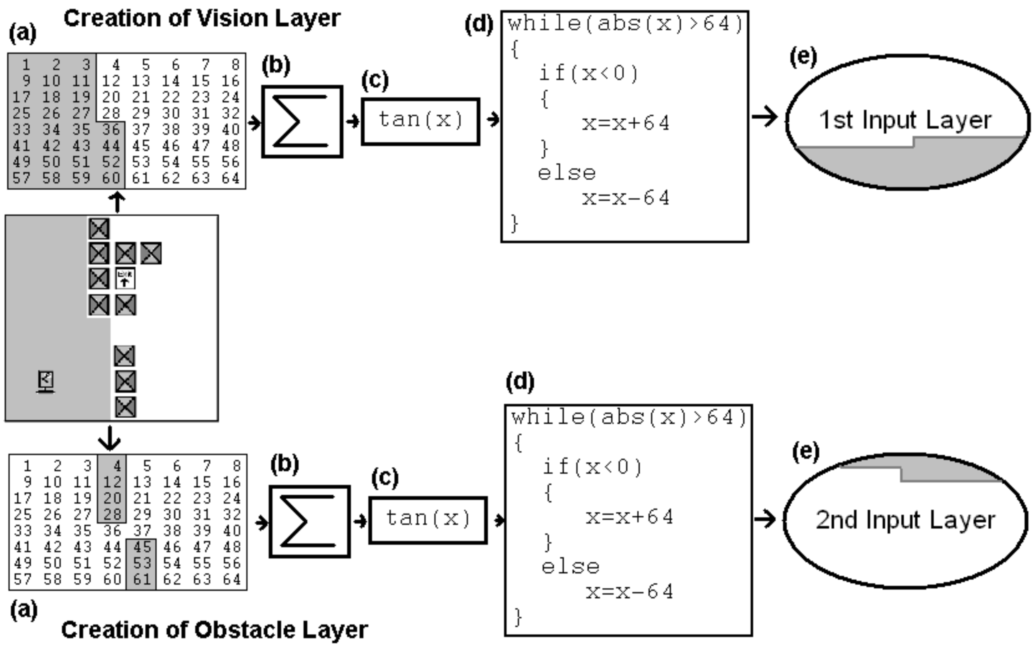

Obstacles and Destination NEVER change!
(...)
Only the robot has different positions.

Figure 5: Demonstration of the problem of dealing with different positions of the robot

Beside the fact that similar positions produce problems of oscillation, it also has to be ensured that the progress of the robot inside a single labyrinth is correctly treated. As shown in figure 5, only the obstacles and the position of the robot produce little difference at the input layer. Thus the path itself, meaning the sequential behavior only providing the next point at every step (and not the whole solution at once), is not correctly treated. This means that the position of the robot has to be clearly amplified at the input pattern.



```
while(abs(x)>64)
{
  if(x<0)
  {
     x=x+64
  }
  else
     x=x-64
}
```

```
while(abs(x)>64)
{
  if(x<0)
  {
     x=x+64
  }
  else
     x=x-64
}
```

Figure 6: Extraction of characteristics and translation into the two input layers

Therefore a more sophisticated way of preprocessing the information coming from the labyrinth is adopted. This is done in order to make sure that similar positions produce different entries and that the position of the robot has more influence on the pattern itself.

The preprocessing scheme extracts the main characteristics of the labyrinth: visible empty fields, visible obstacles and position of the robot. It works on each one of them and maps them to the input layers.

The following steps are observed (see figure 6):

- The board is mapped to an 8 by 8 matrix. Vision of empty spaces and visible obstacles are numbered in the matrix with fields from 1 to 64 (a).
- The corresponding numbers of the visible fields are summed in a first step of processing (b). This pattern is used to ensure that identical numbers of visible fields do not automatically produce identical results (as this would potentially bring about problems especially in well explored labyrinths).
- The number obtained is passed on to a tangent function (c). This function was chosen, because it is non linear and does not necessarily produce similar results for values that are relatively close to each other.
- The results tend to be very different with frequent low values and with seldom very high values. Thus, they have to be normalized in order to be fitted into an

input layer. This is done by a simple function (d) leaving the respective value between -64 and 0 if the original value is negative and between 0 and 64 if it is positive.

- Afterwards the values are finally supplied to the input layers (e). This is achieved by counting positions from above if the value is positive (e.g. if the final value is 15 the first 15 positions of the matrix starting with [0,0] will be set true, the others false). If the value is negative the positions from below are filled with true (e.g. -27 will fill the 27 positions from [8,8] upwards with true and the others with false).

Finally, the position of the robot is considered. All fields in the two layers with the x-position or the y-position equal to that of the robot are inverted producing a large cross.

Thus a typical input layer may look like the following:

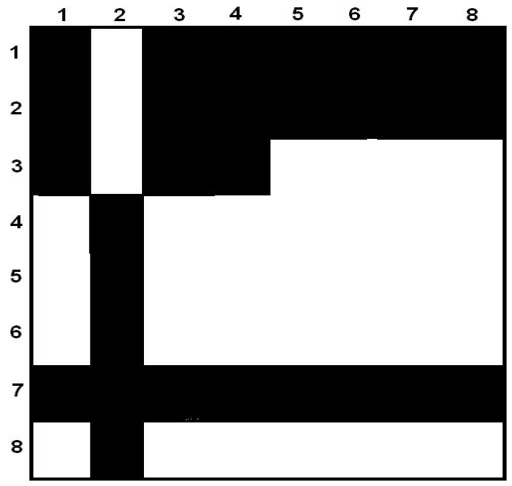


Figure 7: Example of an input layer produced by the number of visible obstacles and the position of the robot at (2,7)

From the input patterns onwards the information is processed in a standard perceptron manner. “Neural Labyrinth Robot” makes use of 10 neurons in the hidden layer. This number has been chosen out of an empirical evaluation with 2 to 64 neurons (see figure 8) as well as considering the computational cost, which rises considerably with the increase of the number of hidden neurons.

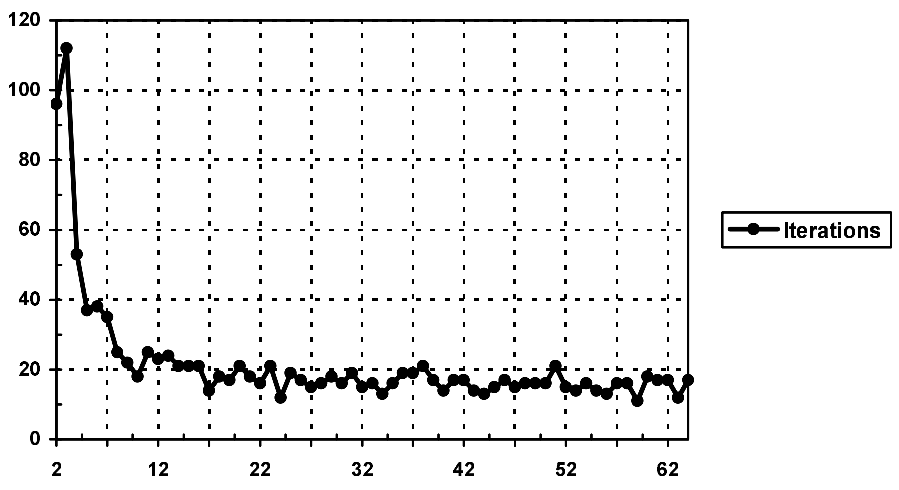


Figure 8: Number of necessary iterations (y-axis) in relation to the number of hidden neurons (x-axis) treating 20 labyrinths

At the end of processing, a matrix 8 by 8 with decimal values from 0 to 1 is produced at the output of the network. "Neural Labyrinth Robot" interprets this matrix in order to find the correct position to move to. In this case, it only considers the positions that correspond to possible turning points. Between these turning points, the one with the highest attributed value is chosen as illustrated in figure 9.

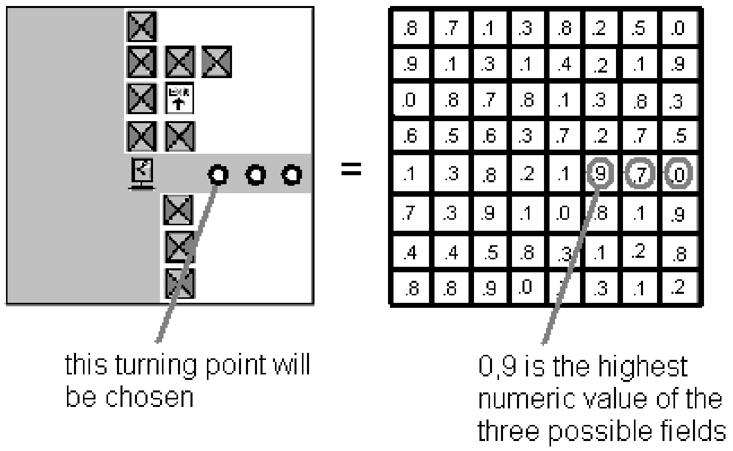


Figure 9: Method of choice between the turning points

Certainly, there is a lot of information being produced, which is not at all being used in the present position. However, depending on the situation, the network has to be able to process a much higher quantity of turning points – the nominal limit, which will never be reached, is 64 because this is the number of positions on the board.

A second reason for this output-layout is a higher flexibility and computational power, which enables "Neural Labyrinth Robot" to process several labyrinths sequentially.

3. Learning

As stated above, the backpropagation algorithm is used, however employing the system of small changes (adopted also in [Schneider and Rosa, 2002]). This system tends to look at the output rather than the real values, meaning that adjustments are only made if the final decision produced is incorrect. In this case the alternative that should have been chosen receives the value 1, the alternative incorrectly provided receives the value 0 and the entire scheme is learned without changing any other value. This produces smoother results and steadier learning as shown in [Schneider and Rosa, 2002].

The learning rate of "Neural Labyrinth Robot" was set to 0.85 as the system obtained a minimum value at 20 (or less) labyrinths (see figure 10). This minimum is true in a topology with 10 neurons and was also validated with 64 neurons.

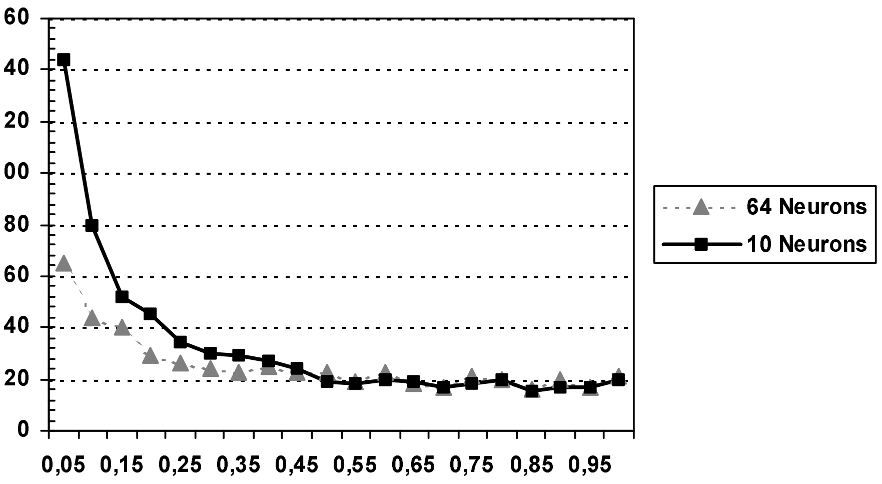


Figure 10: Comparison of Learning Rates 0,05 to 1,0 with 20 Labyrinths

The system successfully treats 20 labyrinths, which can be modified at the user's choice. In this case the system reaches error zero (100% of correctness) with only 16 to 20 iterations. So, it is clearly estimated that the system has the ability to treat many more labyrinths in a reasonable period of time.

4. Known Problems

The inputs' preprocessing produces 64 * 64 * 64 = 262,144 (distinct vision layers * distinct obstacle layers * distinct robot positions) unique positions at most – not considering eventually repeated values being provided by the tangent function.

The matrix-size has not yet been expanded to more than 8 by 8 because of memory problems. Though not being of any theoretical difficulty, the practical solution has yet to be compressed in order to be able to consider larger or even real life mazes.

5. Future Developments

Firstly, a huge number of different labyrinths could certainly be implemented in order to offer the possibility of better evaluation of the system performance. This is done at no extra computational cost, however demands a reasonable amount of time for development, since each labyrinth has to be created manually.

Secondly, the size of the labyrinths must be expanded further in order to adapt the system slowly to real life situations. In this case problems are expected and must be treated, mainly because of limited computational resources.

6. Conclusions

"Neural Labyrinth Robot" is a system based on a simple feedforward neural network that adequately learns several paths from an initial point to the final destination in 20 test labyrinths. The system gives a suggestion of how future guidance systems may process their way through more difficult situations.

References

Benyus, J. M. (1997), "Biomimicry", William Morrow Inc., New York

Filiat, D., Kodjabachian, J., Meyer, J.A. (1999), "Evolution of Neural Controllers for Locomotion and Obstacle Avoidance in a 6-legged Robot", http://citeseer.nj.nec.com/274294.html

Haykin, S. (1999), "Neural Networks: A Comprehensive Foundation", 2nd Ed., Prentice Hall, New Jersey

Auf der Heide, M.F. and Vöcking, B. (1995), "A Packet Routing Protocol for Arbitrary Networks", in: Proceedings of the 12th Symposium on Theoretical Aspects of Computer Science, pp. 291-302

Simmons, R. G., Goodwin, R., Zita, K. H., Koenig, S., O'Sullivan, J., Veloso, M. M., (1997) "Xavier: Experience with a Layered Robot Architecture", http://www-2.cs.cmu.edu/~reids/bibliography.html

Schneider, M.O. and Rosa, J.L.G. (2002), "Neural Connect Four – A Connectionist Approach to the Game", in: Proceedings of the 7th Brazilian Symposium on Neural Networks (SBRN'02) in Recife, IEEE-Computer Press, pp. 236-241

Satchok, G. (1997), "Metropolitan In-street On-route Passenger Transport: Monitoring and Control", The United Nations University, Report Nr. 110, Tokyo/Japan