Artificial Neural Network based Local Motion Planning of a Wheeled Mobile Robot

I. Engedy* and G. Horváth*
* Budapest University of Technology and Economics, Department of Measurement and Information Systems, Budapest, Hungary
engedy@mit.bme.hu, horvath@mit.bme.hu

Abstract—**In this paper we present an artificial neural network based motion and path planning system of a wheeled mobile robot navigating among stationary and moving obstacles. The neural network is aware of its distance sensor readings and its relative position from the target. The neural network is used in this system as a controller, and it is trained using a previously proposed extension of the backpropagation through time algorithm, which uses potential fields for obstacle avoidance. The operability of this method is presented in a series of simulation results.**

I. INTRODUCTION

The navigation system of a mobile robot must handle numerous well separable subtasks for proper operation. These tasks are the localization, motion planning and mapping. Localization is the process of determining the position of the robot in the configuration space. It is a cardinal problem in robot navigation, because the location of the robot is the initial step in the other two problems. Mapping, on the contrary, is the process of determining the surrounding environment, relative to the position of the robot. Later the robot is able to locate itself based on this map. When these two tasks are performed simultaneously, we are talking about simultaneous localization and mapping (SLAM).

The other task of robot navigation, including both mobile robots and robot arms, is motion planning. In this problem the task is to determine the path from one point to another one (called target), and to control the movement, to make the robot follow the previously computed path. Thus motion planning is often discussed as two separate subtasks, the path planning and the movement.

A. Path Planning

Path planning is the procedure, which plans the path from the current location to the end point, of which the robot will have to go through. Obviously, the map of the intermediate area is needed to be known, so the obstacles and unreachable areas could be avoided. There are different types of maps, and based on the type of the map, there are also different path planning algorithms. There are two basic approaches found in the literature, topological landmark based maps, and metric maps [1].

In case of metric maps, the area, in which the robot must operate, is continuous in most time; its range could be represented with real numbers, however most algorithms of path planning work only on finite graphs. The discretization of the area is needed for these graph-based algorithms to be used in path planning on metric maps. There are many methods for that, like cell decomposition, skeletonization using Voronoi diagram, or making probabilistic road-map [1].

Algorithms, that are working on finite graphs, are for example the well-known Dijkstra, Bellman-Ford, or A* algorithms [2].

There are also soft-computing methods for path planning [3]. They usually utilize fuzzy logic controllers, artificial neural networks, and reinforcement learning methods. They are used mostly because solving this problem with classical methods could be computationally inefficient. They do not require an exact mathematical description of the problem, and therefore they are able to operate in a broader scale. Due to their abstraction skills, they better tolerate the noisy input data. Also a usual reason for their utilization is their adaptability to changing environments, like moving obstacles, or dynamically changing target. Some of the methods presented in the sequal are also capable of solving the two sub-problems of motion planning, the path planning and the movement planning. They are able to take the possible kinematic constraints of the robot into account, thus planning a path that the robot is actually able to follow, like a nonholonomic car-like robot cannot follow a path with curves sharper than the maximal wheel angle, like in [4].

Artificial neural network (ANN) based motion planning techniques can be divided into two groups by the basics of their operation. One of these groups contains solutions that explicitly render the configuration space and the robot state-space into an array of locally connected neurons. This array is then trained or used with different kind of methods, resulting in a trajectory that connects the present state of the robot with the target state. Usually the ANN is trained with an analytical or unsupervised training algorithm. This includes Hopfield-network in [5] and [6], self organizing map (SOM) in [7] and dynamic wave expansion neural network (DWENN) in [8] and [9]. The other group of techniques contains methods which don't follow the previous paradigm, at least not explicitly. This group contains solutions for example like ANN based robot motor controller, that tells the robot in which direction to turn based on the state of the robot. In these solutions the ANNs must be trained with an either static or dynamic training method. In the followings we will discuss a system, which is part of the second group.

B. Movement

The movement is the procedure of keeping the robot on the path. The motion could be complex [10], because the movement of the robot must meet various physical constraints, like the momentum of the car, or the slip of the wheels, or the minimum turning arch. There are many different ways to keep the robot on its path. The most common solution is to use various controllers, such as P, PD or PID controllers. These provide the system input in real-time, by looping back the measured system output, which is the robot location. The disadvantage of this is that there is a need for a precise model of the system, the robot and the environment in which the robot moves.

The movement control could be carried out using soft computing methods, including ANNs. Some of these methods are able to simultaneously perform the path planning and the movement problems too [2].

In the sequel we will briefly present our former solution of a motion planning system that is such an ANN based mobile robot navigation solution, which is able to avoid static and moving obstacles. We will describe the problems that have risen during the testing of the former system. We will propose a modified and extended version of this system. At the end of this paper, we will present simulation results of the new system, and conclusion.

II. Former Results

In our former work [11], we have shown a solution of the mobile robot motion planning problem. The robot was a nonholonomic car-like robot, with two independently driven wheels, and one steady wheel. It was able turn the wheels in both direction, but during the tests of the navigation system, we decided to use the robot only in forward mode. The target of the robot was a predefined position and orientation in the working area. There were also static and moving obstacles that the robot had to have to avoid.

We have presented an ANN based mobile robot controller for obstacle avoidance. The ANN was trained with the classical backpropagation through time (BPTT) training approach. We have shown that it can be extended using regularization, where through the regularization term additional constraint can be taken into consideration, and how this is used for avoiding static or moving obstacles in a navigation problem. It was also shown that real time online training is possible if the speed of the moving objects – the mobile robot and the obstacles – is rather small.

A. Details of the former motion planning system

In that work, the robot localization was performed with a camera on the ceiling, and an image processing algorithm, which recognized a marker on the top of the robot [12]. The position of the robot was described with a pair of coordinates in a Descartes coordinate system, and its orientation was described with the two coordinates of a two dimensional unit vector. The orientation was the angle between the x axis of the coordinate system and this vector.

Because we used online training, the target position and orientation could be changed during operation. These were also described in the same Descartes coordinate system as the robot.

There were also moving obstacles in the working area. Their presence was only known for the system by the regularization term added to the error from the target. This was backpropagated through the whole unfolded chain of controllers and system models during the online training.

B. Problems with the former system

Although the online training is a good solution to adapt to changing environment, e.g. moving obstacles, it has also a couple of drawbacks. The most serious one is the increased need of computational power. The only solution for this problem is to use faster or more processors.

Another problem with online training is that the ANN has to learn over and over the same movements, for example to turn left, when the obstacle is on the right side. The method is not capable of remembering the already learned movements, if a new movement in a new situation is trained. This problem could be solved with another neural network, or some other knowledge-based system, that is trained with these rules.

Similar problem is that the description of the configuration space (coordinates of the robot and target) are relative to a coordinate system. This way the ANN has to learn the same movements in different positions. It doesn't matter if the robot has to avoid an obstacle in the same situation at the centre of the workspace or at the edge of the workspace.

There were also some problems with the camera based localization. Most of the time it worked well, determined the position of the robot, but it was not robust enough, so sometimes there were incorrect recognitions. Without checking for these errors, the online training also takes these faulty recognitions as input, taking a big noise into the training procedure.

III. Modified Navigation Method

The modified version of our motion planning method is using an ANN as a controller of the robot just like in the originally proposed system. It is also trained with the backpropagation through time method. The output of the neural network is the same as it was before: the turning angle of the robot in the next iteration. However the input of it is slightly different. Now the state of the robot is described with the relative position and orientation of the target to the robot. The description of the state is also changed: it is described with a distance and angle coordinate pair in a polar coordinate system. The input of the ANN is extended too, with a polar coordinate system grid, which is a grid map of the obstacles in front of the robot.

A. The BPTT method in robot navigation

The backpropagation through time method [13] is a training method of dynamic feedback artificial neural networks (ANN) [14]. The main idea of the method is to train the ANN with the usual backpropagation algorithm by opening the feedback loop: the feedback neural network is unfolded through many iteration steps. This way we get a chain of the same ANNs. The training is done in the following way: the input is given to the first ANN, which calculates its own output. This output would be the input of the second ANN, and so on. We get the output of the chain at the output of the last ANN. This output is used to calculate the error, which is then propagated back through the entire chain using the chain-rule. After calculating the partial derivatives of the error with respect to every weights, the weights can be modified for example with the Delta-rule, but only in the original ANN, which has been copied many times all along the chain, because of opening the loop.

The BPTT could be used for robot navigation, as D. Nguyen and B. Widrow showed it [15], “Fig. 1,” where the controller of the robot is an artificial neural network and the whole control loop is opened, including a model of the controlled system.

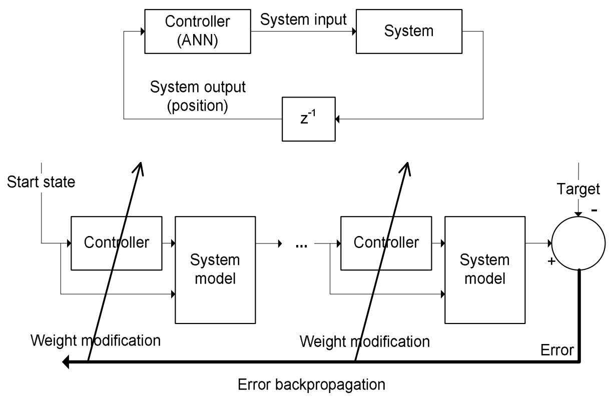

Figure 1. BPTT in use of training in a control loop

In this situation not only ANNs take place in the chain, but numerous models of the system, which are not needed to be trained, but the error must be propagated back through them. The models can be constructed using different approaches: e. g. a mathematical model can be used, or the model itself can be constructed by training a properly selected neural network. The former approach is useful if a quite accurate mathematical model can be constructed rather easily, while the latter if the motion equations of the mobile robot are rather complex. In [11] the model was also constructed using a neural network, while in this work the use of a mathematical model was more practical. The input of the model is the output of the controller, which is the control signal, for example the voltage of the motors. In our case it is the turning angle of the robot. The output of the model is the input of the controller, which is the state of the robot. In this case it is the position of the target related to the robot.

B. *The description of the state of the robot*

As it was mentioned before, the state of the robot is described with its relative position to the target. It was also mentioned that this relative position is described in a polar coordinate system.

As it is shown in “Fig. 2.” this state description contains the following values: α, the angle between the direction the robot is facing and the direction of the line pointing to the target from the robot, *d*, the distance of the robot and the target, and β, the angle between the direction the robot is facing and the direction the robot should face in the target.

Describing the robot state this way, the target of the training, where we wish the neural network controller to guide the robot, is the origo of the coordinate system. So we want to train the neural network that at the end of the unfolded chain, the state of the robot will be zero in all three values. This way there is no need to subtract anything from the state of the robot at the end of the chain, because numerically the state is the error we want to backpropagate through the unfolded chain and minimize. This is a much more similar approach to control theory, than the former navigation system was.

It could be easily seen that the backpropagation through time is not capable by itself to work with obstacles. The only aspect of the training is to get to the target position at the end of the chain.

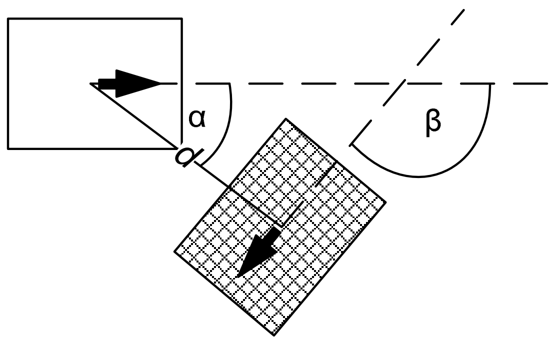

Figure 2. State of the robot relative to the target, in polar coordinate system. The white rectangle is the robot, and the checked one is the target position. The arrows show the direction of the robot

C. *The extension of the BPTT method*

It could be easily seen that the backpropagation through time is not capable by itself to work with obstacles. The only aspect of the training is to get to the target position at the end of the chain.

To make the BPTT method able to take account of the obstacles, we have elaborated the following solution. Based on the location and the size of the obstacles, a potential function can be defined (1), which can be used to repel the robot away from the obstacles. To actually repel the robot, this function must be used to extend the cost function of the Delta-rule. The cost function (2) is regularized with the potential field, so the goal of the weight modification is not only to minimize the error at the end of the simulation chain, but also to minimize the potential of the path, to get the robot the farthest from the obstacles. This way the obstacles could be avoided.

$$U_i = \begin{cases} \dfrac{1}{(d_i - r_i)^2}, d_i > r_i + \varepsilon \\ \dfrac{1 + r_i - d_i}{\varepsilon^2}, otherwise \end{cases} \tag{1}$$

$$C_R(t,y) = \|t - y_n\|^2 + \lambda \sum_{i=0}^{n} \sum_{j=0}^{k} U_j(y_i) \tag{2}$$

U_i is the potential field of the i^{th} obstacle, d_i is the distance of the centre of the obstacle from the robot, r_i is the radius of the obstacle, y is the position of the robot. The parameter ε is a finite positive real number; it ensures the value of the potential field to be always finite.

$C_R(t,y)$ is the regularized cost function (3). Its first term is the usual squared deviation of the simulated end position y_n from the target position t. In this case t is zero. This term is responsible to get the robot to the target at the end of the simulation. The end position is the output of the simulation chain, so it is the position of the robot at the end of the simulation. Only this robot position is used to calculate the error for the weight modification of the ANNs. The second term of the cost function is the regularization term. Here n is the number of iteration steps during BPTT, and k is the number of obstacles. It is responsible for the obstacle avoidance. It uses each ANNs' outputs, each robot positions along the simulation chain to calculate the potential of each obstacle, because the collision must be avoided all along the path, not just at the last step. This is why the potential field is summarized through *i*. The summarizing through *j* refers to the superposition of the potential field of multiple obstacles.

This cost function is minimized with the weight modifying algorithm of the neural network. In this case we use the backpropagation algorithm with the delta-rule to do so. The partial derivative of the cost function with respect to each weight $\partial C/\partial w$ is needed, which can be expressed by the chain-rule as the product of the partial derivative of the cost function with respect to the actual robot position $\partial C/\partial y_i$, and the partial derivative of the actual robot position with respect to each weights $\partial y_i/\partial w$ (3).

$$\frac{\partial C}{\partial w}=\frac{\partial C}{\partial y_i}\frac{\partial y_i}{\partial w} \tag{3}$$

The first term of the cost function, the squared deviation of the end position from the target, is propagated back in an ANN the same way as it was described in the disclosure of the BPTT method, so the regularization term could be added to the error during the backpropagation at the appropriate places through the simulation chain, and the weights of the ANN can be modified using the usual way as it can be seen in "Fig. 3".

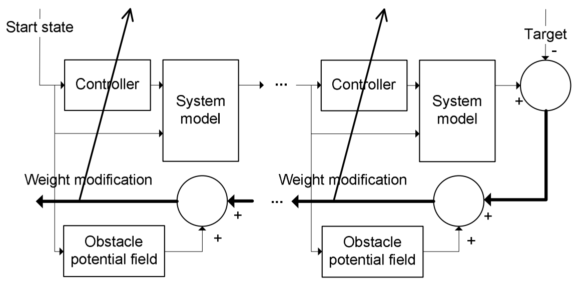

Figure 3. Adding the obstacle potential to the error

The backpropagated error, which has the same dimension as the state of the robot, contains five values, the distance of the robot and the target, the two coordinates of the orientation vector of the robot-target line, and the target orientation, which is also a unit vector with two coordinates. The orientations are described with the two coordinates of a two dimensional unit vector. The orientations are the angle of these vectors. This way the angles can be described with continuous functions, which are needed by the ANN.

However the potential function depends only on the location of the robot, so the derivative of the potential field with respect to the target orientation is zero. To determine the partial derivative of the potential field with respect to the location of the robot, first it is needed to discuss the description of the obstacles.

The obstacles, similar to the target, are described in the polar coordinate system of the robot. Each obstacle has two values, its distance to the robot, and the angle between the direction of the robot and the line pointing from the robot to the obstacle. In this description it is hard to see how the potential field depends on the polar coordinates of the target. In "Fig. 4" it is shown how the distance and angle of the obstacle can be calculated if the distance of the target and the obstacle is known. The equations for this calculation are shown in (4).

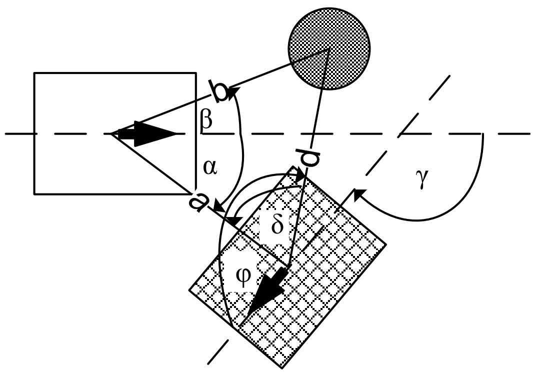

Figure 4. The obstacle (dotted circle) and its relative position to the robot (white rectangle) and the target (checked rectangle) in the polar coordinate system

The *d* distance and the φ angle of the obstacle relative to the target are independent from the robot position or orientation. So we can express the *b* distance and β angle of the obstacle relative to the robot using these values, and the *a* distance and α angle of the robot relative to the target, and the γ target orientation (4).

$$\begin{aligned}\delta&=(\alpha-\gamma)-(\pi-\varphi)\\ \beta&=\alpha+cos^{-1}\left(\frac{a^2+b^2-d^2}{2ab}\right)sgn(\delta)\\ b&=\sqrt{a^2+d^2-2ad\,cos(\delta)}\end{aligned} \tag{4}$$

The potential fields only depend on the *b* distance, so the partial derivates of the distance with respect of the robot states must be calculated. From that, the partial derivates of the potential field with respect to the robot positions are easy to calculate (5).

$$\begin{aligned}\frac{\partial U_i}{\partial a}&=-U_i^{-3/2}\left(a-d\,cos(\delta)\right)\\ \frac{\partial U_i}{\partial \alpha}&=-U_i^{-3/2}ad\,sin(\delta)\end{aligned} \tag{4}$$

These partial derivates now can be used to regularize the cost function, this way repel the robot away from the obstacles.

D. The extension of the neural network input

So far this system is quite similar to our former one [11], only this system is using polar coordinates relative to the robot to describe the world. This polar coordinate description has solved the problem of learning the same movement in different places in the working area. Now the ANN has to learn each different situation only in one place, because the absolute position of the robot is not an input of the neural network.

The polar coordinate description partially solves the localization problem too, because this way there is an opportunity to sense the world using distance sensors on the robot. Since the position of the environment (obstacles and the target) is relative to the robot, the sensor values are not needed to be transformed to an absolute coordinate system.

However the increased need of computational power of the online training is still a problem, just like that the robot has to learn all the movements over and over again. For these problems we have proposed the following solution.

The movements of the robot could be stored in a knowledge based system. Neural network is such a system, and we are already using one. We could use this ANN to remember to the different movements in the different situations. To do that, we must somehow differentiate the different situations from each other at the ANN's input too.

Until this point, the input of the ANN was only the state of the robot, its distance and angle to the target. The network was not aware of the obstacles, it was just simply trained that way that it will avoid the obstacles coded in the cost function we used for training.

To make the ANN aware of the obstacles, we have extended its input with a polar grid "Fig. 5.", where each cell in the grid is an input of the ANN. If there is an obstacle in the grid cell, its value will be 1, if the cell is empty, the value will be 0. The grid is only in front of the robot, since it is moving only forward, we do not need to know what is behind the robot. The movement only depends on the direction and distance of the target, and the obstacles in front of the robot.

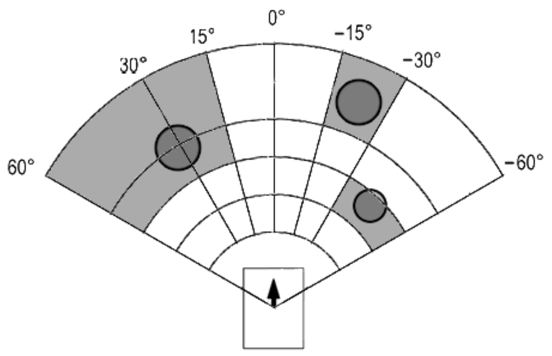

Figure 5. The polar grid in front of the robot. The gray cells are the occupied cells, in which there is at least 20% of an obstacle, the dark circles are the obstacles, and the rectangle is the robot.

The grid is a bit modified. The closest cells are merged into one big cell in front of the robot, because these would be so small, that they won't give any relevant information to the robot, and they are so close, that if there is an obstacle in any of them, the robot must turn around.

Also the cells far from the robot are merged in distance, because these are so far away, that there is no much difference if only one of them contains an obstacle. This is the reason for the merging of the side cells. If the robot turns that way, it will get a better resolution of the obstacle field.

With the use of this polar grid, the ANN is able to learn and remember the movements for each situation in which it was trained with the regularized BPTT algorithm. The weights of the neurons in the first hidden layer from the polar grid input values are functioning as the memory for this purpose.

This way the neural network can be used in offline training mode too, so before using the network, it can be trained with a training set of motion planning problems, which are solved by the extended BPTT training method. The solutions of these problems are stored in the neural network. After the training, the trained ANN can be used to navigate the robot among static and moving obstacles. It should be noted, that the speed of the moving obstacles should be slightly lower than the speed of the robot. If there are faster obstacles than the robot, it should be considered using obstacle position prediction, like we showed in [11].

The offline training needs a set of problems that is going to be solved. We used randomly generated situations for the robot. Since the robot state is described with the relative distance and angle of the target, these values can be changed randomly. The angles could take any value from [-π; π), and we restricted the robot–target distance to the [200 mm; 3000 mm] interval. To compare, the shaft length of the robot was 120 mm. We used 1–3 obstacles. They were placed randomly between the robot and the target. For each situation, we trained the ANN only for one BPTT step, to prevent overfitting to any situation. With these randomly selected situations and one BPTT training step per situation, we achieved, that the network was trained to solve these kinds of situations in general.

IV. SIMULATION RESULTS

The method was tested in simulation only. The former simulator application was modified with the previously mentioned changes. First, the extended BPTT method was tested with the new polar coordinate description approach.

The tests showed that this polar coordinate description and robot centric approach is more suitable for the neural network. The convergence of the training was considerably faster than it was with the Descartes-coordinate system. This way problems that took very long for the former system to solve, were solved easily with the new approach "Fig. 6". The obstacle avoidance of the system remained the same as it was in the former version "Fig. 7".

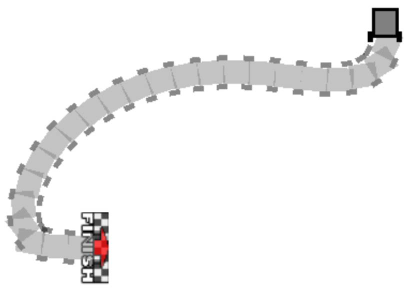

Figure 6. A problem and its solution in the new polar coordinate system description by our method. The dark rectangle is the robot, the checked area and the FINISH text is the target, facing the direction of the arrow. The light gray trajectory is the path of the robot.

After the training of the neural network, we have also tested its capabilities without training it anymore. We have placed some obstacles between the simulated robot and the target. One of the obstacles was a moving obstacle. We started the robot, to see, if it is actually capable to avoid colliding with the obstacles and get to the target. As it is shown in “Fig. 8”, the robot was able to get to the target without collision.

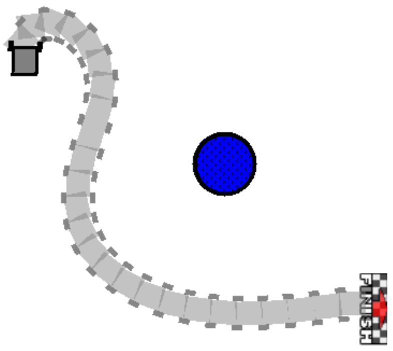

Figure 7. Obstacle avoidance capability of our method. The blue circle is the obstacle.

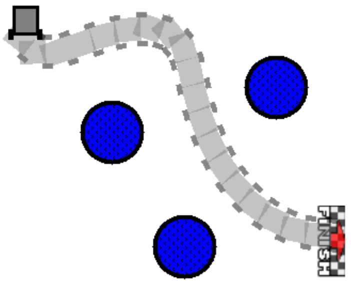

Figure 8. The path of the robot after the neural network controller was trained.

V. CONCLUSION

In this paper we have shown that the classical BPTT training approach can be extended using regularization, to solve the motion planning problem and to take additional constraints, like obstacles, into consideration. We have shown an effective way to describe the robot and its environment using a polar coordinate system. It was also shown that it is possible to handle moving obstacles as well, by extending the inputs of the neural network with a polar grid, without training the neural network online.

This offline trained ANN is able to navigate the robot on a collision free course towards the target. However there are some limitations on its capabilities.

It is not guaranteed, that the path of the robot will be optimal by any means, or that the robot will reach the target, and it won’t get stuck in a local minima. With this offline training only a little, local part of the environment is taken into account, on which it is not possible to compute a globally optimal path. This method is useful when the navigation system doesn’t have any information on the rest of the environment, only the local surroundings of the robot, for example during exploring or map making.

If a bigger part or the whole map of the environment is known, other path planning algorithms can be used to compute a globally optimal path. Using our method with waypoints on the path, the robot is able to follow it, without colliding with any static or moving obstacle. This way path planning algorithms that are incapable of taking the constraints of the robot or dynamic changes in the environment into account, could be also used for mobile robot motion planning.

ACKNOWLEDGMENT

The authors gratefully acknowledge the support of the Hungarian Fund for Scientific Research (OTKA), Grant #73496

This work is connected to the scientific program of the "Development of quality-oriented and cooperative R+D+I strategy and functional model at BME" project. This project is supported by the New Hungary Development Plan (Project ID: TÁMOP-4.2.1/B-09/1/KMR-2010-0002).

REFERENCES

[1] Sebastian Thrun, Wolfram Burgard, Dieter Fox: Probabilistic Robotics, MIT Press 2005

[2] Stuart Russel, Peter Norwig: Artificial Intelligence A Modern Approach, Prentice Hall 2003

[3] Pratihar, D.K.: Algorithmic and soft computing approaches to robot motion planning. Mach. Intell. Robot Control 5,1-16 (2003)

[4] S. X. Yang and M. Meng, “Neural Network Application in Robot Motion Planning”, IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, pp 611-614, 1999.

[5] R. Glasius, A. Komoda and S. A. M. Gielen, Neural Network Dynamics for Path Planning and Obstacle Avoidance, Neural Networks, 8(1), 1995, 125-133.

[6] N. Sadati and J. Taheri, "Solving robot motion planning problem using Hopfield neural network in a fuzzified environment," in IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'02). vol. 2, 2002, pp. 1144-1149.

[7] A.J. Knobbe, J.N. Kok, and M.H. Overmars. Robot motion planning in unknown environments using neural networks. In Proceedings of the ICANN'95, volume 2, pages 375-380, 1995.

[8] D.V. Lebedev, J.J. Steil, and H. Ritter, “A neural network model that calculates dynamic distance transform for path planning and exploration in a changing environment,” in Proc. IEEE Int. Conf. on Robotics and Automation, 2003, pp. 4201-4214.

[9] D.V. Lebedev, J.J. Steil and H.J. Ritter, the Dynamic Wave Expansion Neural Network Model for Robot Motion Planning in Time-Varying Environments, Neural Networks, 18(3), 2005, 267-285.

[10] J.-C. Latombe, Robot Motion Planning. Boston , MA : Kluwer, 1991.

[11] I. Engedy and G. Horváth, “Artificial Neural Network based Mobile Robot Navigation,” in Proc. of the IEEE International Symposium on Intelligent Signal Processing, pp. 241-246, Budapest, Hungary, Aug. 2009

[12] I. Engedy and G. Horváth, “A Global, Camera-based Mobile Robot Localization”, in Proc. of the International Symposium of Hungarian Researchers on Computational Intelligence and Informatics, pp. 217-228, Budapest, Hungary, Nov. 2009

[13] M. Minsky and S. Papert. Perceptrons: An Introduction to Computational Geometry. The MIT Press, Cambridge, Massachusetts, 1969.

[14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, Parallel Distributed Processing: Explorations in the microstructure of cognition; Vol. 1: Foundations, Cambridge, Massachusetts, 1986. The MIT Press.

[15] D. Nguyen and B. Widrow, ``The Truck Backer-Upper: An Example of Self-Learning in Neural Networks," Proceedings of the International Joint Conference on Neural Networks, 2:357-362, 1989.