Some improvements in the reinforcement learning of a mobile robot

Marcin Pluciński

Faculty of Computer Science and Information Technology,
Westpomeranian University of Technology, Żołnierska 49, 71-210 Szczecin, Poland,
mplucinski@wi.zut.edu.pl

Abstract: The paper presents the application of the reinforcement learning for the autonomous mobile robot moving learning in an unknown, stationary environment. The robot movement policy was represented by a probabilistic RBF neural network. As the learning process was very slow or even impossible for complicated environments, there are presented some improvements, which occurred very effective in most cases.

Keywords: reinforcement learning, probabilistic RBF neural network, mobile robot

1. Introduction

One of the most interesting machine learning methods is a reinforcement learning. The learning is based on gaining a procedural knowledge (skill) during intermediate interactions with an environment in which this skill will be used to perform a given task. A learning system/agent doesn't need any a priori knowledge about an environment and it even doesn't need to know explicitly a task that it learns to perform. During interactions with an environment an agent receives only a scalar "reward" or "reinforcement" feedback signal indicating how good or bad its action was and on that base it tries to adapt its future action policy to receive better rewards [11,13].

The reinforcement learning (RL) is very close to human learning, because a man obtains a lot of his skills by a trial and error method, so RL is potentially one of the best approaches to creating really intelligent systems. The RL was successfully used for many practical applications from a strategy learning in board games [13,14] to learning given behaviours of mobile robots [1,4,5,6,7]. The paper presents the application of the RL to the autonomous mobile robot moving learning in an unknown, stationary environment.

2. Reinforcement learning

Generally, we can say that during the RL we want to find an optimal action policy in an un-known environment to solve a given task. A learner can observe a state of an environment and on the base of its value it chooses its action according to its current policy. At the beginning, a policy is taken arbitrarily, so it is very typical for a learning process that learner makes a lot of wrong actions. The learner makes errors and receives a reinforcement/reward feedback signal from an environment. On the base of this information it tries to improve its policy [11].

A policy depends only on an environment state, so during learning a learner is to find an optimal mapping from perceived states to action to be taken when in those states:

$$\pi: \quad X \to A,$$

where: π – a policy, X – a set of environment states, A – a set of possible learner actions [3,13]. For a current learner policy π, we can define a value function that is a mapping from states to the total amount of reward an agent can expect to accumulate over the future, starting from the state x:

$$V^{\pi}(x) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t} \cdot r_t \mid x_0 = x\right], \tag{1}$$

where: E_{π} – expected (for the policy π) sum of future rewards r_t, $\gamma \in (0,1]$ – discount rate, which determines that rewards received in the future are less worth for the state value.

During the RL a learner looks for an optimal policy π^{*} – the policy for which it will always receive the best rewards from an environment. For such a policy, the value function $V^{\pi^*}(x)$ is always biggest or equal to the value function $V^{\pi}(x)$ for any policy π. If the optimal value function $V^{\pi^*}(x)$ is known, we can easily find an optimal action policy π^{*} as a greedy policy to $V^{\pi^*}(x)$. A greedy policy to $V(x)$ always chooses its action to maximize an immediate reward and a discounted next state value of a value function $V(x)$ [13].

One of the most important RL methods is a temporal difference (TD) algorithm. In this algorithm we'll try to find the optimal value function $V^{\pi^*}(x)$. At the beginning, the $V(x)$ function is initiated arbitrarily (often in a random way). During succeeding learning steps, the function is modified on the base of observed experiences $< x_t, a_t, r_t, x_{t+1}>$. The modification rule can be written as:

$$V_{t+1}(x_t) = V_t(x_t) + \eta \cdot \Delta, \tag{2}$$

where: η – step-size parameter, which influences the rate of learning, and:

$$\Delta = r_t + \gamma \cdot V_t(x_{t+1}) - V_t(x_t). \tag{3}$$

$V_t(x_t)$ – is the value of the current value function and $r_t + \gamma \cdot V_t(x_{t+1})$ – is a sum of a reward received in a step t and a discounted value of the next state – it is probably better evaluation of real value of a $V^{\pi}(x)$ (for a current policy π) than $V_t(x_t)$ [11,12,13].

If actions will always be chosen such that a value of the next value function is maximal, then our policy will be greedy to the function $V(x)$. Equations (2) and (3) can be written as:

$$V_{t+1}(x_t) = (1-\eta)\cdot V_t(x_t) + \eta \cdot \left(r_t + \gamma \cdot \max_a V_t(x_{t+1}) \right). \tag{4}$$

An operator $\max_a$ means choosing the greedy policy. Iterative improving of the value function while applying the greedy policy leads to a convergence to the optimal value function $V^{\pi^*}(x)$, for which the optimal policy can be easily found as greedy to it.

Described method of finding the optimal policy works well for a finite number of states and actions. In the case of a continuous set of states and actions, the learning becomes more complicated. We must use continuous mapping $\pi : X \to A$, and in practice it is easiest to model continuous value function $V(x)$ and to choose actions according to the policy that is greedy to $V(x)$. For modelling of a policy by the value function $V(x)$ we can use any modelling method, but it must have: a possibility of an iterative improving (a possibility of learning in an incremental mode), a possibility of learning on the base of an infinite number of data, a small amount of calculations needed for a function actualisation [3]. Neural networks are often preferred method of a modelling and in experiments there was applied the probabilistic RBF neural network (PRBF NN). The network had the constant number of neurons with constant width. Only weights of an output layer were parameters that were adapted during learning process.

The main advantages of the PRBF NN are easiness of learning and interpretation of its parameters values – differently from a perceptron multilayer NN. The network enables modelling of any complex continuous mapping. It can be used in an incremental learning mode and its adaptation process is fast and not complicated. The main disadvantage of the PRBF NN can be possible slow work in the case of a large amount of RBF neurons [8,9,11].

3. Reinforcement learning of the mobile robot movement policy

A RL task will be the learning of a policy of a mobile robot moving in an unknown, stationary environment with a given, constant (and known to the learner) target point.

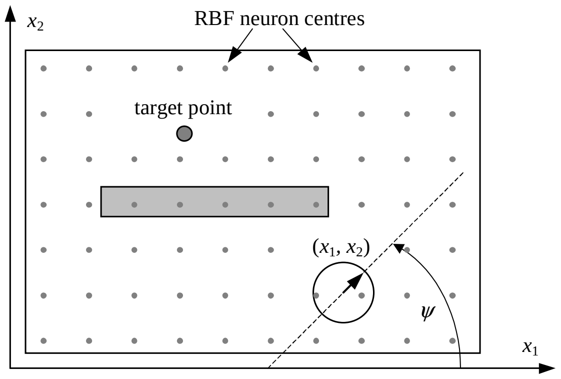

Fig. 1. The state and action of the robot and exemplary locations of RBF neurons [11].

A robot state is defined by its location in a stationary coordinate system $\mathbf{X}=(x_1,x_2)$ (simplified notation x will be used later) and its action is defined by chosen given movement direction ψ, Fig. 1. Both the state and the action are continuous signals. The value function $V(x)$ is modelled by the PRBF NN. Centres of neurons are placed uniformly in the entire environment and are constant, Fig. 1. One additional neuron is placed in the robot target point. During learning, only output layer weights (connected with each neuron) will be changed.

The learning process is very sensitive to a value function $V(x)$ initialisation method. As it was said before, $V(x)$ can be initiated arbitrarily. For example it can be random or equal for the entire environment, but by proper initialisation of $V(x)$ we can introduce to the learning system same a priori knowledge. A good solution is to cause that function maximum is in the target point and thanks to that, choosing the policy that is greedy to $V(x)$ will always produce the effect of robot moving towards that point. Beginning robot location is chosen randomly but of course it can’t be placed inside obstacles [11].

In each step (t_s) the robot reads its sensor values and checks if there is any obstacle on its movement way. If the robot detects anything it stops, next it turns randomly left or right by 135° and it moves straight for a given steps number.

In each learning step t (taken every tenth step t_s in experiments), if there is no obstacle, the robot chooses new action (direction of its movement) and the $V(x)$ function is adapted (learned). The learning process is described exactly below [11].

1. In each learning step the robot tests values of $V(x_{t+1})$ in a radius defining possible robot positions in a next learning step, Fig. 2. The radius can be easy counted when we know the robot speed. After testing, we know the state x_{t+1} for which the $V(x_{t+1})$ takes the maximum value and in that way we know the new movement direction (according to the greedy policy).

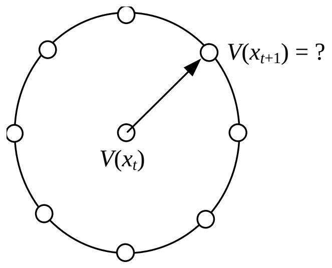

Fig. 2. Searching for a new greedy action [11].

2. The robot checks the environment. After reaching the target point (with a given accuracy) it receives a large positive reinforcement r – a reward.
3. A value of the value function in a previous learning step $V(x_{t-1})$ is modified on the base of the received reward r_{t-1} (0 or r) and the current value $V(x_t)$ taken earlier in a greedy way:

$$V_{t+1}(x_{t-1}) = (1-\eta)\cdot V_t(x_{t-1}) + \eta \cdot \left(r_{t-1} + \gamma \cdot \max_{\psi} V_t(x_t) \right). \quad (5)$$

Formula (5) describes the way of $V(x)$ adaptation according to a TD algorithm. It could be difficult to apply in practice but we can use approximate formula in which the weight y_p of the RBF neuron is changed instead of the exact $V(x)$ value. Finally, the learning process can be described as:

$$y^*_{p,t+1}\left(x^*_{t-1}\right) = (1-\eta)\cdot y^*_{p,t}\left(x^*_{t-1}\right) + \eta \cdot \left(r_{t-1} + \gamma \cdot \max_{\psi} V_t\left(x_t\right)\right), \tag{6}$$

where variables with "star" mean state x and weight y_p connected with the RBF neuron which is closest to real state x_{t-1}.

4. The robot saves the current state and takes the new action/direction found in point 1.

After realisation of points 1-4 in each learning step, the robot continues moving in the taken direction. After reaching the target point, a new start point is taken randomly and the learning process is continued. The entire stage of learning (from start point to end point) can be called a learning epoch or an episode.

The learning process is very sensitive to parameters taken by a system designer. Wrong parameter values can even make that the learning process becomes impossible. The most important learning parameters are: the way of value function $V(x)$ initialisation, reinforcement value r, RBF neurons width, step-size parameter η, discount rate γ, learning step.

4. Improvements of the reinforcement learning

There were made a lot of experiments of learning of the robot in environments with different difficulty degree. Parameters described in previous sections were set as follow: r = 10, η = 0.1, γ= 0.9. Results of experiments are described in details in [11].

As it was mentioned above, reinforcement learning is performed with a trials and errors method. In the case of a searching of the robot policy in the given environment, it consists in many repeated trials of moving from the randomly chosen start point to the given (known to the robot) target point.

In very simple environments such trials usually stop with a success, after which there are reinforced correct directions of a movement to the target point. But in more complicated environments, the single episode of moving to the target point can take very long time or it can never finish with a success. A complicated environment can also require repeating of greater number of episodes. So, all this together can cause that learning time can become unacceptable long even for simulation experiments. In the case of experiments with real robots, usually there is no possibility of performing great amount of trials, which can last very long and not always must finish with a success.

For these reasons, the basic RL algorithm needs improvements.

4.1. Negative reinforcement

The first solution that was tested was applying negative reinforcements in cases when the robot approached an obstacle. While it is recommended [13,14] to give the learner a reinforcement different from 0 only in the end of the episode (it should guarantee a learning algorithm convergence to the really optimal policy) – it is intuitively obvious that small negative reinforcements generated by obstacles can cause, that the value function in their neighbourhood will be smaller. Thanks to that, the robot will not even try to approach them and in that way there will be diminished a number of possible actions from which the robot must choose the correct one leading to a task solution.

In very simple environments, such idea works well. Fig. 3 shows example results of an experiment in which small negative reinforcement was equal 0.0001 and a positive reinforcement (equal 10) was assigned only once in the end (finished with a success) of the whole episode.

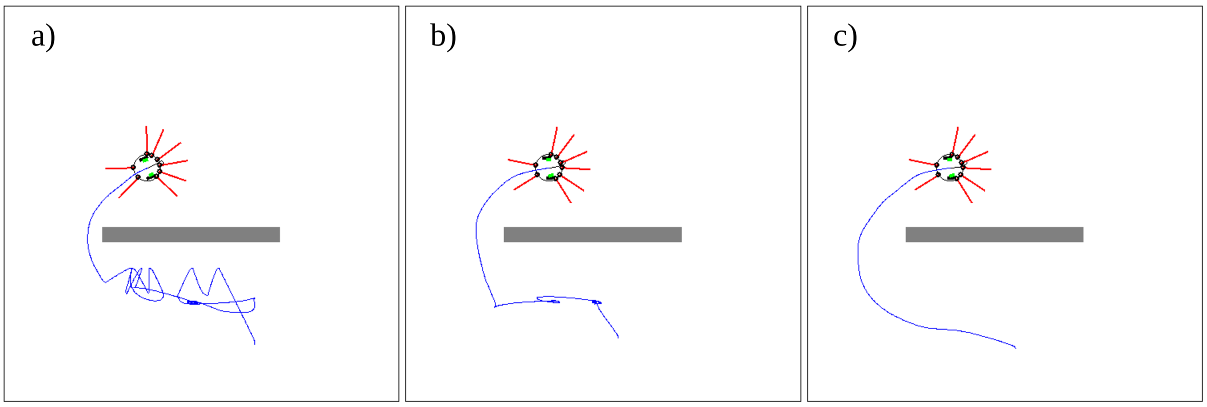

Fig. 3. Successive learning phases with a negative reinforcement application.

Fig. 3 presents robot movement trajectories in different learning phases. In the Fig. 3a we can see a chaotic movement of the robot which tries (with a trail and error method) to reach the target point through the obstacle (first episode of learning). In that phase the robot learns the obstacle position. Fig. 3b presents the second learning phase (after 10 episodes) in which the robot “knows” the obstacle position and doesn’t try to move close to it. Fig. 3c presents the trajectory of the robot movement after learning (50 episodes). Learning without negative reinforcements lasted about 500 episodes (10 times longer).

For more complicated environments, applying negative reinforcements fails. It is caused by creating of value function $V(x)$ local maxima. As the robot realises a greedy policy, such local maxima always attract it and cause that the learning process gets stuck and the robot starts to move around the local maximum. Such effect we can observe especially for environments in which there are bounded rooms. It is illustrated in Fig. 4.

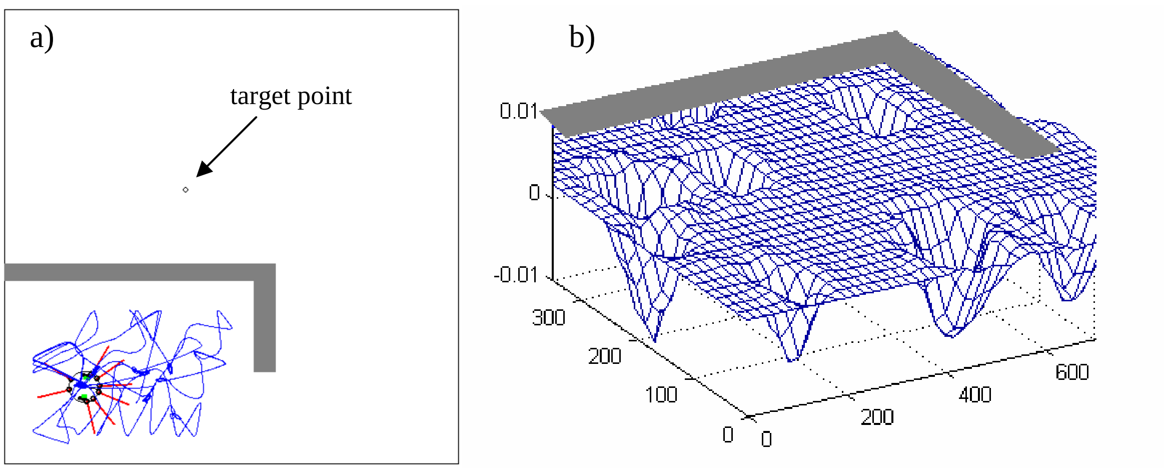

Fig. 4. Trajectory of the robot movement (a) and the surface of $V(x)$ function (b).

Fig. 4b presents a part of the $V(x)$ function surface – it can be seen that local minima in the "entrance" cause leaving the room completely impossible.

We can easy prevent such effects by testing the robot location every some given time (500 learning steps in experiments). If the state change is lower than a given threshold it can mean that the robot has got stuck and moves around $V(x)$ maximum. Then a greedy policy can be changed for a random movement direction for some time and the robot can go away from the maximum point. After some time the robot returns to the previous learning mode. Such solution, however, is effective only for a moment and if the robot finds itself in a local maximum neighbourhood it will get stuck again.

As a conclusion, it can be said that negative reinforcements are good solution only in very simple environments (small amount of small obstacles), but in most cases they make the learning process worse – completely blocking it.

4.2. Heuristic behaviour

An another approach may be attempt of prompting the robot any heuristic behaviour during solution searching. A very good effect were obtained by stopping the learning in a moment of approaching an obstacle and starting moving around it in a randomly chosen direction. After finishing bypassing phase, the robot went back to the described earlier learning mode.

One of the problems occurred here was time duration for which the robot should switch into a bypassing mode. As the robot has no information about size, shape and location of obstacles in the environment and the range of its sensors is rather limited, it can't determine whether the obstacle is bypassed yet or not. We can choose one of two solutions here:

- time of bypassing is chosen randomly,
- time of bypassing is set in advance by supervisor of the learning as an additional information about the environment (for small obstacles time should be shorter and for bigger ones – longer).

Exemplary results of experiments are presented in Fig. 5.

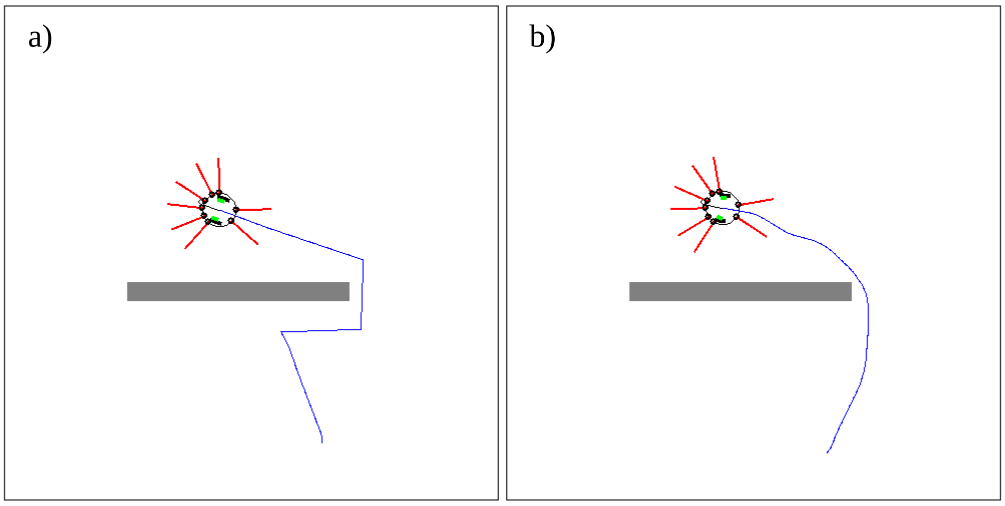

Fig. 5. Trajectories of the robot movement in the first learning episode (a) and after learning (b).

Fig. 5a presents the robot trajectory in a first learning step. After approaching the obstacle it is bypassed. Fig. 5b shows the trajectory after 40 learning episodes.

4.3. Gradually learning

A man learns to solve problems gradually, slowly increasing a level of difficulty. First, he tries to solve simple tasks and slowly moves to more and more complicated. A trial of a hard problem solving by an unlearned person probably will fail and what is more important an effort spent to overcome the task will usually not change into improving learner skills in next learning steps.

It is similar with robots. If the robot has to move in a complicated and unknown environment it probably will not be able to reach the target point moving randomly in acceptable long time. So, it seems reasonable to graduate the difficulty level. Such solution was applied for example by G. Tesauro [14] for backgammon policy learning. In the beginning, only the policy of the end phase of the game was learned, next the game started in the medium phase and in the end the learning process included the entire game.

Similar approach can be applied in a learning of a robot movement policy. We can assume, that it is easier to reach the target point from start points, which lies close to it. Each episode finished with a success will reinforce correct paths to the target.

So, in the beginning of a learning process start points should be chosen randomly in a close neighbourhood of the target point. A radius of the neighbourhood should gradually increase with each next episode. Effects of such approach are illustrated in Fig. 6.

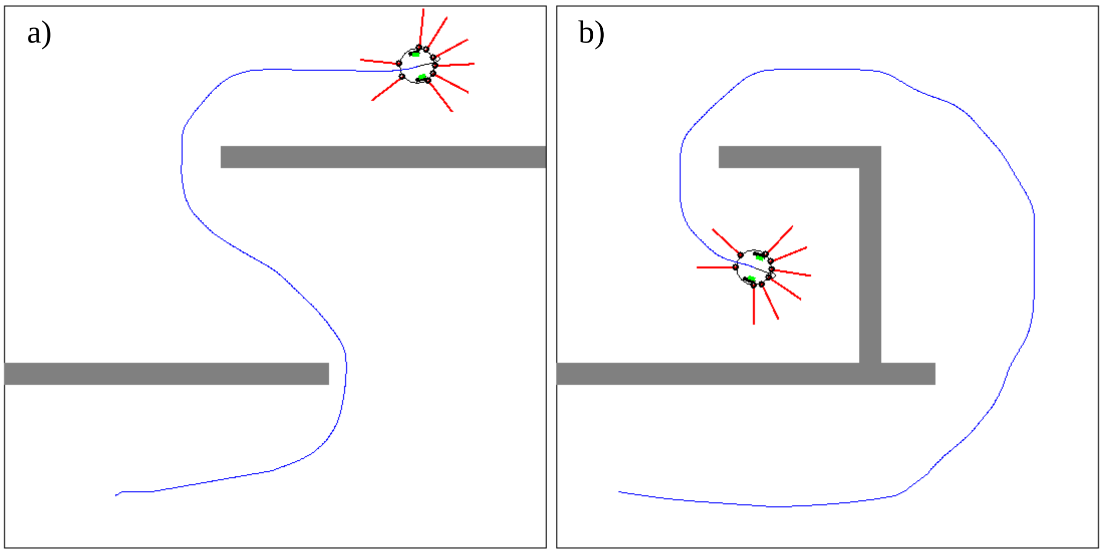

Fig. 6. Trajectories of the robot movement after gradually learning.

Fig. 6 presents exemplary trajectories of the robot movement after finishing the learning process (about 500 episodes). An experiment without applying gradually learning in the same environments completely failed. If start points were chosen far from the target, the robot wasn't even able to finish single episode.

5. Conclusions

There exist a lot of environment learning (mapping) and path planning (in known environment) methods. The connection of RL and PRBF NN, described in the paper, enables solving of both mentioned tasks at the same time – the robot can simultaneously learn the environment and look for the optimal trajectory.

Learned movement policies are characterised by very safe and smooth movement trajectories, which are located in a large distance from obstacles. Trajectories presented in Figs. 5 or 6 are very close to optimal ones and their shapes are similar to trajectories found by a potential field’s method [2,10].

Negative reinforcements didn’t fulfil expectations. The local maxima problem makes this modification rather unuseful. But, an introduction of heuristic behaviours and gradually learning works very well. It accelerates the entire learning process many times and in complicated environments it makes this process really possible.

References

1. BEKEY G.E., *Autonomous robots* (*from biological inspiration to implementation and control*), The MIT Press, **2005**.
2. CHOSET H. at all, *Principles of robot motion* (*theory, algorithms and implementation*), The MIT Press, **2005**.
3. CICHOSZ P., *Learning systems*, Wyd. Naukowo-Techniczne, Warszawa, **2000** [in Polish].
4. CONNELL J., MAHADEVAN S., *Rapid task learning for real robots*, In Robot Learning, Kluwer Academic Publishers, **1993**.
5. KAELBLING L.P., LITTMAN M.L., MOORE A.W., *Reinforcement learning: A survey*, Journal of Artificial Intelligence Research, vol. 4, pp. 237-285, **1996**.
6. LONG-JI LIN, *Hierarchical learning of robot skills by reinforcement*, Proceedings of the International Conference on Neural Networks, **1993**.
7. MILLAN J.R., *Rapid, safe, and incremental learning of navigation strategies*, IEEE Transactions on Systems, Man, and Cybernetics, **26**(3), **1996**.
8. MOORE A.W., ATKESON C.G., *An investigation of memory-based function approximators for learning control*, Technical report, MIT Artificial Intelligence Laboratory, Cambridge, MA, **1992**.
9. PLUCIŃSKI M., *Application of the probabilistic RBF neural network in multidimensional classification problems*, In Advanced Computer Systems, Kluwer Academic Publishers, pp. 49-57, **2002**.
10. PLUCIŃSKI M., KORZEŃ M., *Application of the Peano curve for the robot trajectory generating*, Proceedings of the 13th International Multi-Conference on "Advanced Computer Systems", Międzyzdroje, Poland, pp. 43-52, **2006**.
11. PLUCIŃSKI M., *Application of the probabilistic RBF neural network in the reinforcement learning of a mobile robot*, Polish Journal of Environmental Studies, vol. 16, no. 5B, pp. 32-37, **2007**.
12. SUTTON R.S., *Learning to predict by the methods of temporal differences*, Machine Learning, vol. 3, pp. 9-44, **1992**.
13. SUTTON R.S., BARTO A.G., *Reinforcement learning: An introduction*, The MIT Press, **1998**.
14. TESAURO G., *Practical issues in temporal differences learning*, Machine Learning, vol. 8, pp. 257-277, **1992**.