Application of the probabilistic RBF neural network in the reinforcement learning of a mobile robot

Marcin Pluciński

Faculty of Computer Science and Information Technology,
Szczecin University of Technology, Żołnierska 49, 71-210 Szczecin, Poland,
mplucinski@wi.ps.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. The network proved to be very attractive tool, which enabled efficient, simple and fast approximation of the state value function.

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 [12].

The reinforcement learning is a very universal approach but it is often very hard in a practical realisation. It is the reason why other learning methods, like supervised learning for example, are much more popular. One of the main difficulties is a delay (sometimes very large) of an environment reward. Another problem is frequent stochastic and nonstationary character of an environment.

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 [12,13] 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 unknown 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 the environment. On the base of this information it tries to improve its policy.

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:

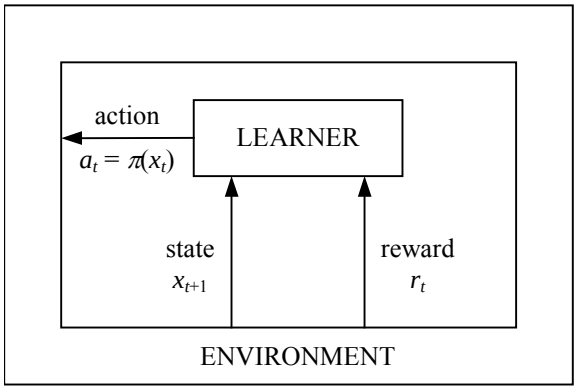
$$\pi : \quad X \to A,$$

where: π – a policy,

X – a set of environment states,

A – a set of possible learner actions [3,12].

A schematic diagram of the RL is presented in Fig. 1.



for all learning steps t
observe an environment state x_t
choose an action a_t to be taken in a state x_t
(according to current policy)
perform an action a_t
observe a reinforcement r_t
observe a next state x_{t+1}
on the base of an experience $< x_t, a_t, r_t, x_{t+1} >$
improve a policy
end

Fig. 1. A schematic diagram and the basic algorithm of the RL [3].

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 that state:

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

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 $V(x)$.

2.1. TD algorithm

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:

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

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 $V_t(x_t)$ [11,13].

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:

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

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 the paper there is applied the probabilistic RBF neural network (PRBF NN).

2.2. Probabilistic RBF neural network

The PRBF NN can be successfully applied for classification and approximation tasks. It can be classified to memory-based function approximation methods, but it has some parameters that can be tuned too, so it is possible to easy control an approximation quality [8,9].

The PRBF NN works with a data set in which each sample consists on a pair $<\mathbf{X}_p, y_p>$, where: $\mathbf{X}_p$ – a neural network input vector and y_p – a given network output, $p = 1 \ldots L$. The number of RBF neurons in the network is equal to the number of samples from the data set and a centre of each neuron "lies" in the input vector $\mathbf{X}_p$. A parameter that can be tuned in the PRBF NN is a width (σ_p) of neurons [9]. It can be the same or different for each neuron. Another parameter can be the y_p value.

A PRBF NN output is calculated from the formula:

$$y(\mathbf{X}) = \frac{\sum_{p=1}^{L} y_p \cdot \exp\left[-w_p \sum_{j=1}^{n} \left(x_j - x_{pj}\right)^2 \right]}{\sum_{p=1}^{L} \exp\left[-w_p \sum_{j=1}^{n} \left(x_j - x_{pj}\right)^2 \right]}, \tag{5}$$

where: n – a network inputs number,

w_p – a weight which defines a RBF neuron width: $w_p = 1/\left(2\sigma_p^2\right)$.

In the paper the PRBF NN was applied for the modelling of the value function $V(x)$. The network has the constant number of neurons and the constant weight w_p. y_p values will be parameters that will adapt during learning process.

The main advantages of the PRBF NN are easiness of learning and interpretation of its parameters values (both: w_p and y_p) – differently from 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 samples/RBF neurons.

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.

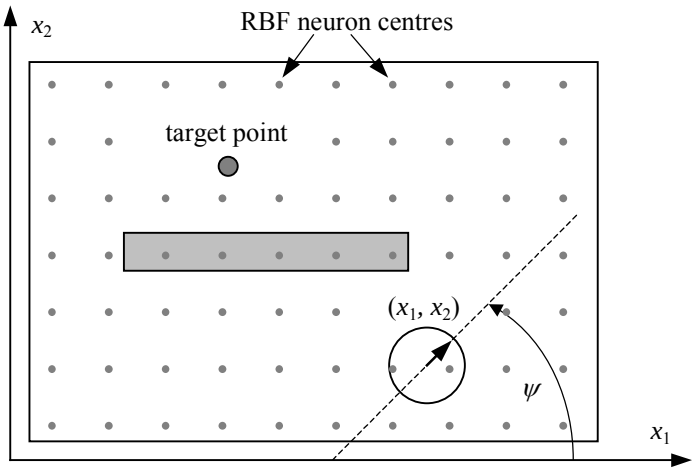


Fig. 2. The state and action of the robot and exemplary locations of RBF neurons.

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. 2. 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. 2. One additional neuron is placed in the robot target point. During learning, only y_p values (connected with each neuron) will be changed. Neurons with positive value of y_p have their width two times bigger than neurons with negative y_p value.

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. In our case, to initialise $V(x)$ we must set a certain beginning value y_p for all RBF neurons. It can be calculated as:

$$y_p = B \cdot \exp\left(\frac{\left\| \mathbf{X}_p - \mathbf{X}_T \right\|^2}{2\sigma_R^2} \right), \tag{6}$$

where: $\mathbf{X}_T$ – coordinates of the target point,

σ_R – a Gauss function width (dependent on the environment size),

B – a constant, taken empirically.

Such method of $V(x)$ initialisation cause that function maximum is in the target point and 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) and next the robot starts to move according to a greedy policy.

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.

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. 3. 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).

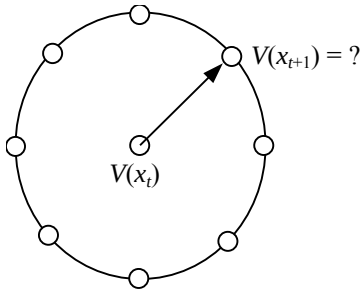


Fig. 3. Searching for a new greedy action.

2. The robot checks the environment. If it finds any obstacle in its neighbourhood it receives a small negative reinforcement (r_1) and after reaching the target point (with a given accuracy) it receives a large positive reinforcement (r_2) – 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_1 or r_2) and the current value $V(x_t)$ taken earlier in a greedy way:

$$V_{t+1}(x_{t-1}) = (1-\eta)\cdot V_t(x_{t-1}) + \eta\cdot\left(r_{t-1} + \gamma\cdot\max_{\psi} V_t(x_t)\right). \tag{7}$$

Formula (7) 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 y_p value of the RBF neuron is changed instead of the exact $V(x)$ value. Finally, the learning process can be described as:

$$y^*_{p,t+1}(x^*_{t-1}) = (1-\eta)\cdot y^*_{p,t}(x^*_{t-1}) + \eta\cdot\left(r_{t-1} + \gamma\cdot\max_{\psi} V_t(x_t)\right), \tag{7}$$

where variables with "star" mean state x and value 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.

During learning the robot can get stuck in a local value function maximum. We can easy prevent it 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 gets 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.

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 values r_1 and r_2,
- RBF neurons weights w_p (widths σ_p),
- step-size parameter η,
- discount rate γ,
- learning step.

4. Experiments

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: $B = 10$, $r_1 = -0.1$, $r_2 = 2.5$, $\eta = 0.1$, $\gamma = 0.9$.

Fig. 4 presents robot movement trajectories in different movement phases. In the Fig. 4a we can see a chaotic robot movement which tries (with a trail and error method) to reach the target point through the obstacle. In that phase the robot learns the obstacle position. Figs. 4b and 4c present the second learning phase (after 10 and 20 epochs) in which the robot "knows" the obstacle position and doesn't try to move close to it. Fig. 4d presents the trajectory of the robot movement after learning (50 epochs).

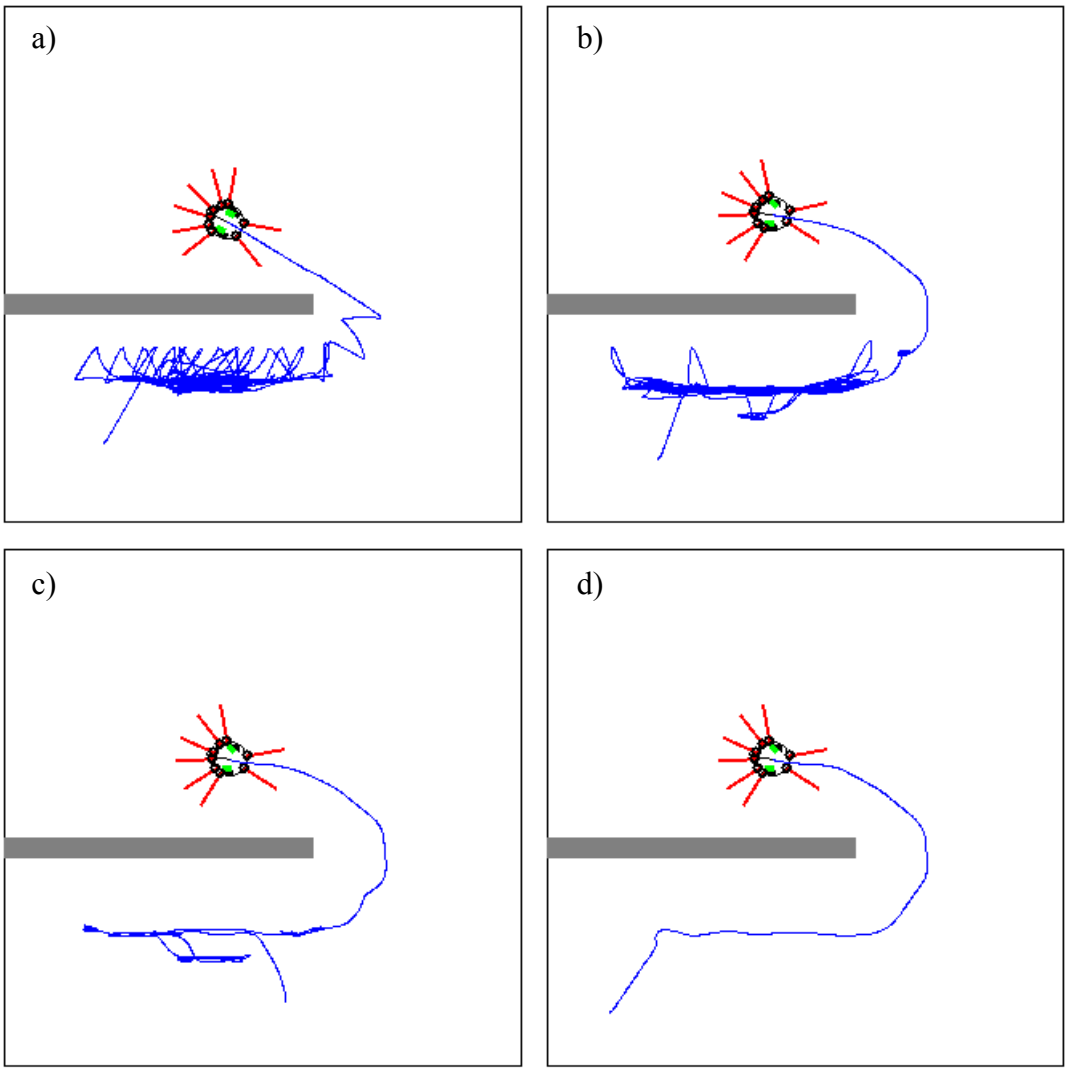


Fig. 4. Successive learning phases in the environment no 1.

Fig. 5 presents trajectories of the robot movement in the environment with a bigger number of obstacles (after 100 learning epochs). In the Fig. 6 we can see a much more complicated environment and exemplary trajectories of movement after 500 learning epochs.

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. 4-6 are very close to optimal ones and their shapes are similar to trajectories found by a potential fields method [2,10]. Fig. 7 presents the surface of the value function before (a) and after (b) learning in the environment no 3.

5. Conclusions

The PRBF NN turned out to be an efficient way of the value function $V(x)$ approximation. The network learns fast and its parameters can be easily interpreted.

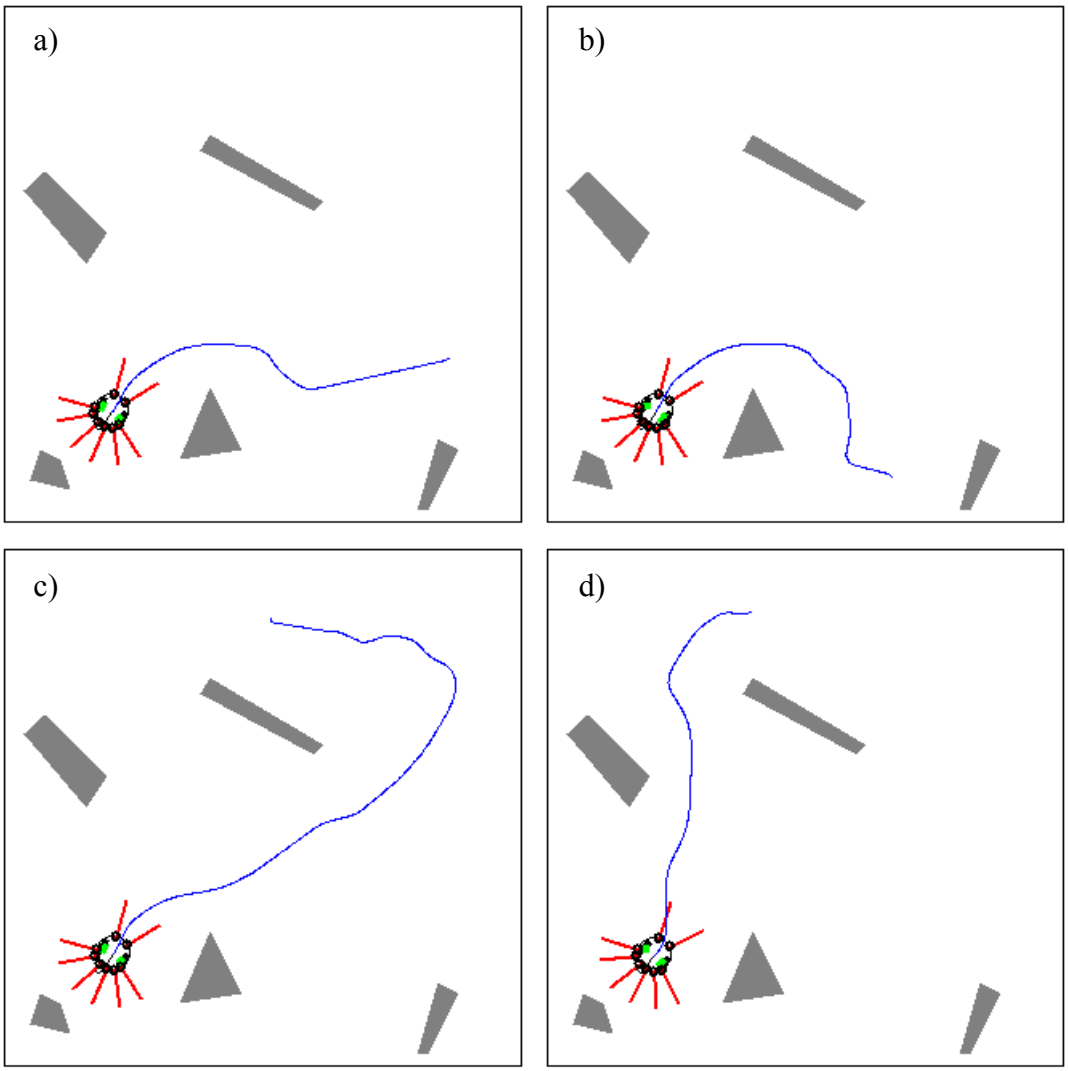


Fig. 5. Trajectories of the robot movement in the environment no 2.

The network can work with problems for large environments where large number of RBF neurons is needed to ensure good mapping quality. In such cases, the network can work slowly and it can't be used in controlling the robot movement in real time. So, the next research direction will be connected with the reduction of the RBF neurons number. Some neurons can be removed from the network without a (sensible for controlling) loss of quality of the approximation.

Mentioned PRBF NN disadvantage doesn't reduce its usability. The PRBF NN is a very attractive tool which enables efficient, simple and fast approximation of value function and in that way it can represent an information about the optimal policy which the robot can learn with the RL application.

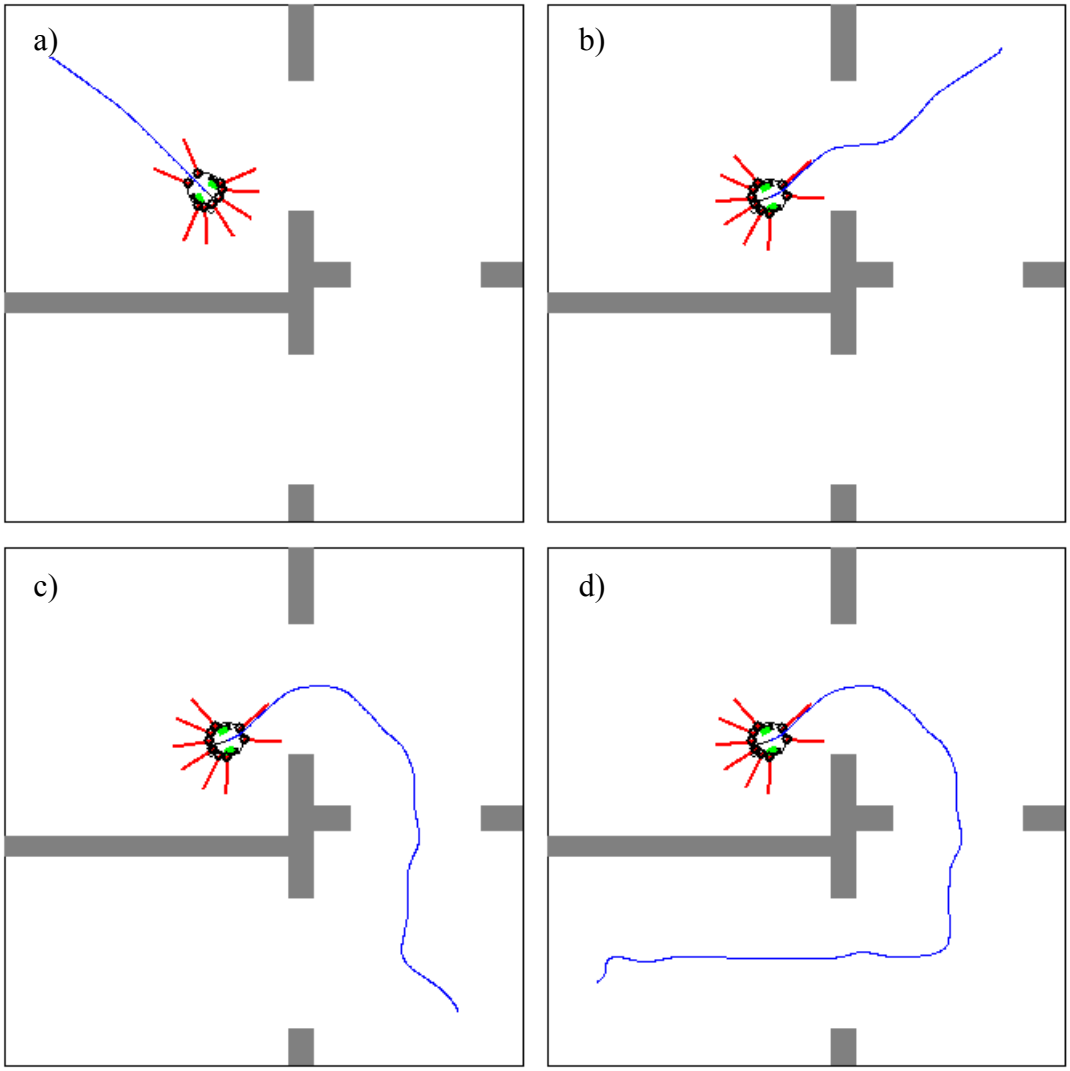


Fig. 6. Trajectories of the robot movement in the environment no 3.

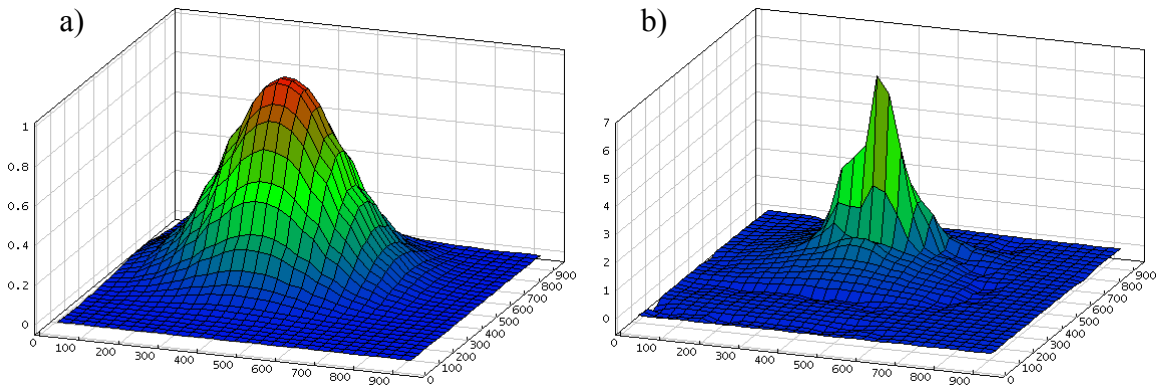


Fig. 7. The surface of the value function before (a) and after (b) learning in the environment no 3.

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*, Wydawnictwa 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. SUTTON R.S., *Learning to predict by the methods of temporal differences*, Machine Learning, vol. 3, pp. 9-44, **1992**.
12. SUTTON R.S., BARTO A.G., *Reinforcement learning: An introduction*, The MIT Press, **1998**.
13. TESAURO G., *Practical issues in temporal differences learning*, Machine Learning, vol. 8, pp. 257-277, **1992**.