ISSN 1330-3651
UDC/UDK 004.85:004.421

Q-LEARNING BY THE *n*th STEP STATE AND MULTI-AGENT NEGOTIATION IN UNKNOWN ENVIRONMENT

Josip Job, Franjo Jović, Časlav Livada

Original scientific paper

This work will show a new procedure of *Q*-learning in which the agent's decision, regarding the next step, is not based on the optimal action at that moment but on the usefulness of a future state. A near agent communication has been implemented so that the agents signal each other their future actions which contribute to a better choice of actions for each of the agents. The new method is named *Q*-learning by the *n*th step and multi-agent negotiation. The results of the testing of this algorithm are compared with the basic *QL* algorithm which is also graphically demonstrated and the advantages of the new algorithm are listed too. An average of 40 % collision decrease is obtained during learning procedure.

Keywords: *agent, learning from reward and punishment, q-learning, reinforcement learning*

Q-učenje prema stanju *n*-tog koraka i dogovaranjem više agenata u nepoznatom okruženju

Izvorni znanstveni članak

U ovom radu je predstavljen novi postupak *Q*-učenja kod kojega agent odluku o sljedećoj akciji donosi na osnovu korisnosti nekog budućeg stanja, a ne na osnovu trenutno optimalne akcije. Implementirana je komunikacija agenata u okolini koji si međusobno javljaju svoje buduće akcije što doprinosi kvalitetnijem odabiru akcija pojedinog agenta. Nova metoda nazvana je *Q*-učenje prema stanju *n*-tog koraka i dogovaranjem više agenata. Uspoređeni su rezultati testiranja ovdje predstavljenog algoritma s osnovnim *QL* algoritmom što je i grafički prikazano te su navedene prednosti novog algoritma. Postignuto je prosječno smanjenje od 40 % sudara tijekom postupka učenja.

Ključne riječi: *agent, pojačano učenje, q-učenje, učenje iz nagrade i kazne*

1 Introduction

Tracing agent behaviour on its way to the destination in an unknown environment with obstacles is a complex problem. The difficulty of the problem lies in the fact that the set of all the possible actions given to all the possible inputs is too complex to be described by the usual program languages. From this arises the need to use one of the methods of machine learning. The problem of seeking a destination while avoiding obstacles can be solved in a variety of ways. Some of approaches are artificial potential field method [1, 2], fuzzy logic approach [3, 4], probabilistic roadmap method [5, 6], used in autonomous robots navigation, proportional navigation [7, 8] for air target missiles navigation, and reinforcement learning approach [9, 10] which will be described with more details in this article.

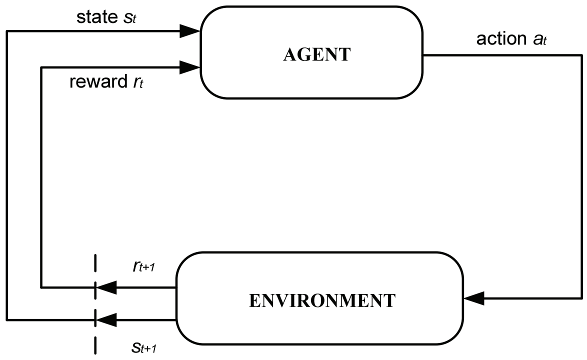

Figure 1 Step-wise interaction between agent and environment in reinforcement learning

The approach presented in this work is based on a process of reinforcement learning. Reinforcement learning (*RL*) is learning through interaction with the environment (depicted in Fig. 1) and it is a method of unsupervised learning. In *RL* systems, in every time step, the agent observes the state and then it chooses an action from a set of allowed actions based on the current state and policy.

The working environment is the setting in which the agent observes and acts. The set of states which an agent can occupy represents a formal bond between the agent and the environment. In this article, the concept of an agent is used for a software simulated object which moves in a working environment, which is divided to a finite number of fields, and it is seeking for a destination field while avoiding a collision with other objects, i.e. other agents. The agent is learning from the rewards and penalties and it is trying to improve its repetitive performance during the time.

2 Existing methods

2.1 *Q*-learning

Q learning is a technique of reinforcement learning [11] which functions in the following way: the agent learns the value function of the action which gives an expected value by executing particular actions in a particular state while following the policy**.** A great advantage of this method is that it does not require the model of the environment to compare the expected value of available actions. In the *QL* method, the agent chooses the action a_t according to the policy π and the *Q*-value in the state s_t. After executing the action a_t in the state s_t and transition into the state s_{t+1}, the agent gets an immediate reward or a punishment r_{t+1}. After that it updates the *Q*-value for the action a_t in a state s_t using the *Q*-value of the state s_{t+1} and the reward r_{t+1} according to the update rule, Eq. (1):

$$Q(s_t,a_t)\leftarrow Q(s_t,a_t)+\alpha\left[r_{t+1}+\gamma\max_{a_{t+1}}Q(s_{t+1},a_{t+1})-Q(s_t,a_t)\right]. \quad (1)$$

This method is proven to converge to the optimal solution [12, 13] with the probability one if each of the actions in every state is performed infinite number of times.

2.2 Double Action *Q*-learning

Q-learning is one of the methods derived from the Markov decision process where the agent tries to perform the actions according to the policy and the *Q*-values in a particular state. In the next step the agent evaluates the action according to the received reward or punishment and according to the expected value of the current state. In dynamically changing environments – DCE, the model of the Markov decision process can be insufficiently efficient in describing the dynamics of the system. The reason for this may be that in dynamically changing environments the next state is not only determined by the action of an agent, but it can be a consequence of changes in working environment. As the traditional *Q*-learning is based on the Markov decision process, it is not appropriate tool for use in dynamically changing environments. In working environments such as this, the update rule of the *QL* method may cause fluctuation of the *Q*-value. For problem solution in such working environments a new method has been suggested by [14, 15], named Double Action *Q*-learning – *DAQL*, which has two basic differences in comparison with the Markov decision process. Firstly, a state is defined by object and obstacle relation in a working environment. The working environment may have different obstacles, both static and dynamic, with their own properties and states towards the agent. The mutual relations of the agent and the obstacles are the states of an agent in a working environment. The second difference is in that the environment can change independently of the action of the agent and according to this both the agent and the obstacle can undertake actions and cause changes in the states. In the *DAQL* method the *Q*-values are iteratively updated and the agent learns how to act in different states. After modifying of the (1), corresponding modified update rule for the *Q*-value is:

$$Q(s_t,a_t^1,a_t^2)\leftarrow Q(s_t,a_t^1,a_t^2)$$
$$+\alpha\left[r+\gamma\max_{a_{t+1}^1,a_{t+1}^2}Q^*(s_{t+1},a_{t+1}^1,a_{t+1}^2)-Q(s_t,a_t^1,a_t^2)\right], \quad (2)$$

where a_t^1 is the action of the agent and a_t^2 is the action of the environment in time step *t*. If there are more agent-obstacles the process of updating the *Q*-values is repeated for each agent–obstacle. There is a similar solution called parallel *Q*-learning which was used in the problem of solving the block-pushing problem [16, 17]. This rule for updating the *Q*-value can be used only when the agent can accurately predict the action of the working environment, but in cases when this is not possible the *Q*-value may not converge so the agent may not act as expected. In cases such as these a less aggressive approach may be by using the mean value of $\max_{a_{t+1}^1}Q(s_{t+1},a_{t+1}^1,a_{t+1}^2)$ overall a_{t+1}^2 i.e.:

$$Q(s_t,a_t^1,a_t^2)\leftarrow Q(s_t,a_t^1,a_t^2)$$
$$+\alpha\left[r+\gamma\frac{\sum_{a_{t+1}^2}\max_{a_{t+1}^1,a_{t+1}^2}Q(s_{t+1},a_{t+1}^1,a_{t+1}^2)}{\mathrm{count}(a_{t+1}^2)}-Q(s_t,a_t^1,a_t^2)\right], \quad (3)$$

where $\mathrm{count}(a_{t+1}^2)$ represents the number of actions a_{t+1}^2 available in the system in the moment $t+1$.

In the original *DAQL* algorithm the agent is expected to predict the actions of other agents (environment) and based on that prediction to decide which action it will choose. If the agent has successfully predicted the action of the working environment then it knows what will be the next step of the working environment so it can avoid collision with other agents. Figs. 2a, 2b, 3a and 3b show the usual example of the difference between *QL* and *DAQL* algorithm.

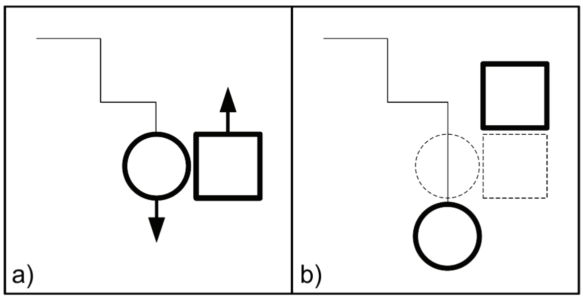

Figure 2 a) *QL* method at time *t*; **b)** *QL* at time *t* + 1

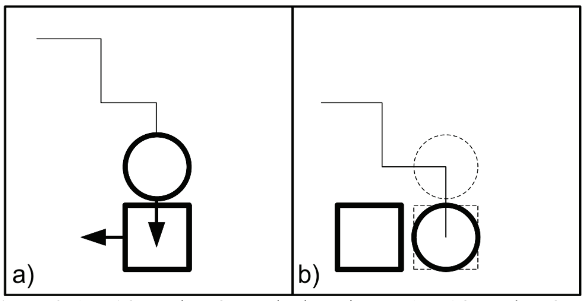

Figure 3 a) *DAQL* and *nnQL* methods at time t; **b)** *DAQL* and *nnQL* at time *t* + 1

In the *QL* algorithm the agent chooses the action DOWN due to collision evasion, because there is another agent in the adjacent field. What the agent does not know is that the second agent/obstacle moves upwards in the same step, so the agent could in fact execute the optimal action for the current step and that action is RIGHT. *DAQL* algorithm will choose the action DOWN in a similar situation, in the case when there is another agent below it and if it successfully predicts action of the obstacle to the left.

3 A *Q*-learning by *n*th step state and multi-agent negotiation

3.1 Basic idea

Unlike the regular *QL* method, in the *DAQL* method, besides the agent action, the selection of actions is done according to environment changes (alterations), i.e. according to other agents-obstacles actions. In calculating the actions of other agents a module for predicting their actions is used. It is difficult to predict the action of another agent in some situations, and historical and empirical data do not always involve the current intention and the circumstances of an agent, while presuming agent cooperation the suggested approach is announcing its future actions, i.e. states, then it can signal other surrounding agents so each of them can make a decision about the optimal action towards the *n* next states of the environment while presuming their compatibility. These are the basic presumptions of the proposed method. The agent may base its decision about the optimal action according to the value of a future state because during the process of learning each action in every state is not repeated for an infinite number of times which is a condition for a convergence of the method. In this way the procedure of learning is accelerated. The decision making by the nth step is calculated by:

$$a_t = \arg\max_{a_t} Q\left(s_{t+n}(a_t), a\right), \tag{4}$$

where a_t is the action of the agent in the moment t, $s_{t+n}(a_t)$ is any possible state which is in n next steps if the first action was a_t and a is any action in the moment $t + n$. In the learning algorithm it is necessary to set the parameters and to initialize the *Q*-value table. After that, the procedure is repeated for each episode in which the initial state of the agent is determined. The agent chooses the action a_t for each step of single episode in which will have the maximum *Q*-value in the nth step. After completing the action a_t, the maximal *Q*-value of a future state is searched for while the *Q*-value of the current state is being updated by the regular equation:

$$\begin{aligned} &Q(s_t, a_t) \leftarrow Q(s_t, a_t) \\ &+\left[r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]. \end{aligned} \tag{5}$$

The next state (s_{t+1}) is exchanged for the present state (s_t) and the procedure is repeated for the next step in cases when the agent has not come to the destination or for the next episode in cases when the agent has come to the destination.

3.2 Multiple goals learning

The agent does not have to be limited to only one goal, but it can have more goals simultaneously. The navigation of the basic robot, which should avoid collisions and seek the target, would be the simplest example of multiple goals. The robot would determine its action according to the optimized path to the destination simultaneously avoiding collisions.

First step of an algorithm in a *Q*-learning by an nth step state and a multi-agent negotiation is the *Q*-table initialization. The initial state of each episode is randomly selected. In every step of the single episode the agent is searching for an optimal action. The agent will choose its action taking into account other agents' actions because in this method agents announce their future actions.

Common *Q*-value for all agent-obstacles is calculated by equation:

$$Q_{CA} = \sum_i \frac{1}{d_i} q_{CA}\left(s_{i,t+n}, a_t\right), \tag{6}$$

where $q_{CA}(s_{i,t+n}, a_t)$ is *Q*-value of agent's state and ith agent-obstacle in the moment $t + n$, and d_i Euclidean distance between the agent and ith agent-obstacle. It is obvious from the formula above that the most significant agent-obstacle in the action-choosing process is the nearest one, and by increasing the distance the significance of an agent decreases respectively.

Final *Q*-value for both goals is calculated after Q_{CA} calculation [18]:

$$Q_{\text{final}} = \beta Q_{DS} + (1-\beta) Q_{CA}, \tag{7}$$

where β is parameter which gives more significance to one of the goals, and Q_{DS} is *Q*-value for destination seeking goal.

Optimal action for both goals is chosen with:

$$a_t = \arg\max_{a_t} \left[Q_{\text{final}}\left(s_{DS,t}, s_{CA,t+n}, a_t, a\right)\right], \tag{8}$$

where $s_{DS,t}$ and $s_{CA,t+n}$ are agent's state in the sense of destination seeking and collision avoidance respectively in the moment of t, and $t + n$.

Q-values are updated after execution of action a_t:

$$\begin{aligned} &Q_{DS}(s_t, a_t) \leftarrow Q_{DS}(s_t, a_t) \\ &+\alpha\left[r_{DS,t+1} + \gamma \max_{a_{t+1}} Q_{DS}(s_{t+1}, a_{t+1}) - Q_{DS}(s_t, a_t)\right] \end{aligned} \tag{9}$$

and

$$\begin{aligned} &Q_{CA}(s_t, a_t) \leftarrow Q_{CA}(s_t, a_t) \\ &+\alpha\left[r_{CA,t+1} + \gamma \max_{a_{t+1}} Q_{CA}(s_{t+1}, a_{t+1}) - Q_{CA}(s_t, a_t)\right]. \end{aligned} \tag{10}$$

Next states are becoming current states, and all of the procedure is repeated until the agent finds the destination or until the completion of the single episode, i.e. after the last step. The procedure is repeated episode by episode until the last episode.

Each agent can find out other agent's actions in the moment of choosing its own action and that is the main reason why this method allows better collision avoidance performance than the regular *QL* method. It allows better reliability in determining consequences of its actions. In cases when an agent should choose action which decreases distance to destination, and if in front of the agent is another agent, this method allows the agent to know if the other agent is leaving its path, resulting in decrease of average steps to destination.

3.3 Multi-agent negotiation

Agents negotiate actions (Fig. 4) in the way that agent *B* answers to agent *A*'s request by its next actions, and after choosing and taking optimal action agent *A* updates its *Q*-table.

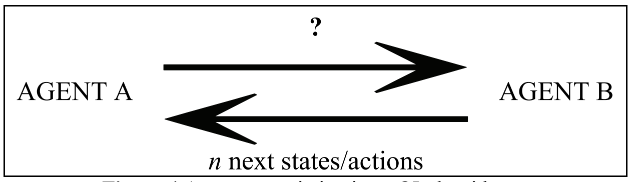

Figure 4 Agents negotiation in *nnQL* algorithm

Some presumptions are made prior to the agents' negotiation, like:

1) Agent *A* is rational and does not go against agent *B* – cooperation
2) Agent *B* is independent from agent *A*, but does not change the chosen action – consistency

The other way of negotiating is by making decisions in a hierarchical way. Every agent gets a priority level and the agent with the highest priority chooses its actions, then the next agent does the same, and so does every other agent until the last one which will coordinate its actions with all other agents.

4 Simulation

The simulation environment was made with the MATLAB numerical computing software tool. The use case functions of different scenarios for both algorithms were coded, respectively, *QL* and the method proposed here. The set of data with initial locations has been generated to achieve the same initial conditions for each algorithm. The data in this set is generated in the random order. Beside the numerical results, graphs and charts were made which show performance of learning according to the number of episodes. Testing of the proposed method was made with the usage of different parameters and its results were compared with *QL* method.

4.1 Destination seeking and collision avoidance learning procedure

The testing was executed to evaluate destination seeking and collision avoidance performance of the two algorithms. In the testing scenario the agent is positioned in the centre of coordinate system and it can see the final destination and the other agents, which are no more than ten cells away from it (ten cells to the left or the right and ten cells up or down). The agent can be in one state out of 441 possible states towards the destination cell and in one state out of 441 states toward any other agent. Possible directions of moving, i.e. possible actions, for the agent in the single state are UP, DOWN, LEFT, RIGHT and DON'T MOVE. One of the goals is to seek the wanted location which can be positioned on any possible location. The agent has to learn the optimal path to the destination cell during the learning phase while the other agents are moving through its environment. Collision is a possible situation when two agents are trying to occupy the same cell in the same time step. Therefore, the second goal is for an agent to learn how to avoid other agents, i.e. collision avoidance.

For the purpose of seeking the destination field the regular *QL* method has been used, while for learning collision avoidance a standard *Q*-learning algorithm was used (*QL*), in one case, and in the other case a *Q*-learning algorithm which bases its decisions on the state of the *n*th step and multi-agent negotiation (*nnQL*). In the second case the *n*th state is the next step and the agents signal their next action, i.e. what their next position will be. As to ensure equal parameters and data of the learning process, a set of data about the starting positions of the destination field for each episode is generated, as are the starting positions of all obstacles and all actions for each individual step during the episode.

Figs. 2a and 2b depict a situation in which a difference between the two approaches can be seen. If the agent, whose learning is based on the *QL* algorithm, is in the situation seen in Fig. 2a, where the obstacle is positioned in the same direction as the destination field, it will not choose the action which leads it towards the destination differing from *nnQL* algorithm which will choose optimal action because of communication between agents (Figs. 3a and 3b).

The function of the reward is set so that during the learning of objective seeking the agent gets a reward which is equal to the reciprocal value of the distance if the agent is getting closer to the objective, half the reciprocal value of the distance if the distance has not changed and no reward if the agent is moving away from the destination. In learning collision avoidance goal, the agent receives a reward ten if there were no collisions and no reward if a collision occurred. As there are two goals a mechanism has been made by which the agent chooses the optimal action for both aspects of learning. This was done in such a way that the *Q*-value of each visible agent/obstacle is calculated and then multiplied with the reciprocal value of their distance from the agent and after that the final *Q*-value is calculated by combining both the behaviour in destination seeking and the behaviour in collision avoidance. The merging of *Q*-values is done by the equation (7).The optimal action will be the one with the maximal *Q*-value and it is calculated by equation (8).

4.2 Algorithm comparison

The comparison was done for the destination seeking and collision avoidance goals during 500 episodes. The number of steps and collisions per episode, the sum of rewards and the total number of collisions were measured. The maximum allowed number of steps per episode is 100.

The learning rate α is 0,9, and the discount factor is 0,6. The random and optimal actions ratio is 0,05, i.e. every twentieth action is randomly chosen.

In this example the agent chooses a random action in every twentieth step (ε = 0,05) and chooses the optimal action in nineteen steps. The *QL* algorithm finishes the first phase of learning after the 230th episode (Fig. 5). In the second phase of learning there are three unsuccessful episodes in the sense of not finding the destination field. The sum of rewards during the learning is 4000 (Fig. 7). The total number of collisions is 350 (Fig. 9). Six collisions per episode is the maximum value of the number of collisions during the learning (Fig.11).

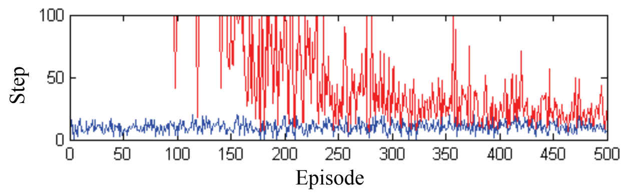

Figure 5 Number of steps per episode and minimal number of steps for *QL* method

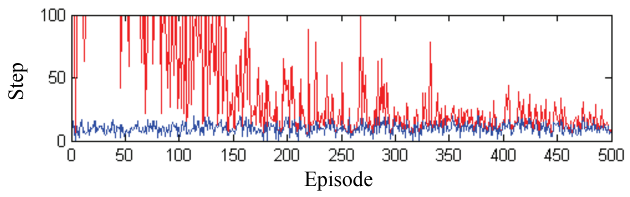

Figure 6 Number of steps per episode and minimal number of steps for *QL* method

In the *nnQL* algorithm the first phase of learning ends after the 140th episode (Fig. 6) which is earlier than in the *QL* algorithm. The sum of rewards in the end of learning is nearly 4000 (Fig. 8). The total number of impacts is 150 (Fig. 10) and the maximum number of collisions per episode is nine (Fig. 12).

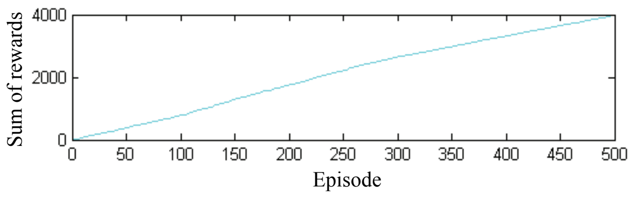

Figure 7 Sum of rewards for *QL* method

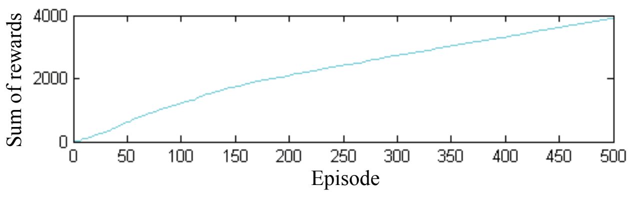

Figure 8 Sum of rewards for *nnQL* method

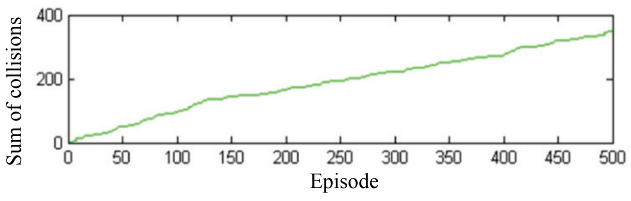

Figure 9 Cumulative number of collisions for *QL* method

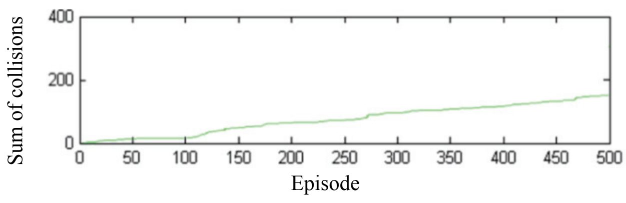

Figure 10 Cumulative number of collisions for *nnQL* method

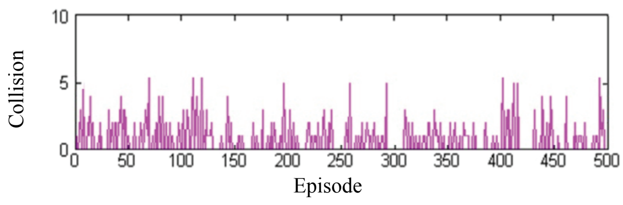

Figure 11 *Number of collisions for QL method*

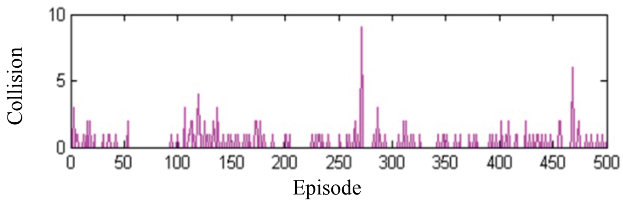

Figure 12 Number of collisions for *nnQL* method

Table 1 Average number of collisions for both algorithms and percentage of decrease

Number of random step	Average number of collisions / *QL*	Average number of collisions / *nnQL*	Decrease in number of collisions / %
1	860,4	849,5	1,27
2	632,9	417,9	33,97
3	534,8	302,6	43,42
4	499,0	252,1	49,48
5	486,8	241,0	50,49
6	439,8	216,5	50,77
7	441,6	195,4	55,75
8	425,1	209,0	50,84
9	408,5	189,6	53,59
10	413,3	202,3	51,05
	514,2	307,6	40,18

Tab. 1 and Fig. 13 show average number of collisions for both algorithms when changing the number of random steps. Number of random step denotes the period of randomly chosen actions.

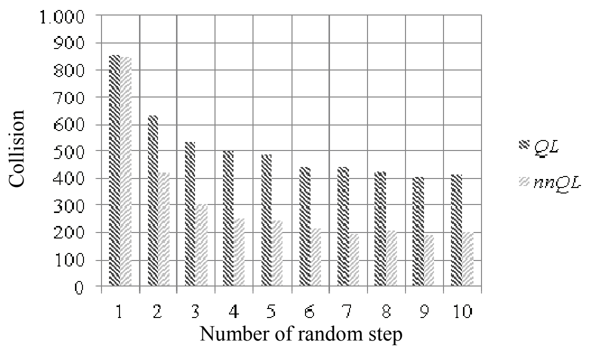

Figure 13 Number of collisions vs number of random steps for both algorithms

The *nnQL* algorithm was proven better because it has a lower total number of collisions. The reason lies in the fact that the *nnQL* algorithm, unlike the *QL* algorithm, knows where the other agent/obstacle is in the next step because of communication so it can reach a better decision in the action it is going to take.

5 Conclusion

Q-learning algorithm which decides on the next action by the state of the n^{th} step and by multi-agent negotiation is presented in this work. The algorithm was tested and its performance was compared with the performance of the regular *Q*-learning algorithms performance. The results of the test have shown a better performance of the new algorithm in the sense of decrement of the number of steps necessary for seeking the destination by about 60 % and decreasing the number of collisions during the learning process in relation to the existing solutions by about 40 %.

As the existing methods of evaluating the performance of *Q*-learning have in some cases shown a certain inconsistency and illogicalities the authors hold it necessary to suggest new methods for evaluating the agents learning performance, collision avoidance performance and to impose a measure of the environment complexity which will present a measure for difficultness of the working environment. The new methods should give a more appropriate evaluation tool considering the successful performance of algorithms corresponding to the dynamics of changes in environment and different situations which an agent experiences during the learning. The authors plan to conduct further research in this direction so as to get a precise tool for evaluating processes of learning.

6 References

[1] Khatib, O. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. // International Journal of Robotics Research. 5, 1(1986), pp. 90-98.

[2] Dunias, P. Autonomous Robots Using Artificial Potential Fields. // Technische Universiteit Eindhoven. 1996.

[3] Sugeno, M.; Nishida, M. Fuzzy Control of a Model Car. // Fuzzy Sets and Systems. 16(1985), pp. 103-113.

[4] Takeuchi, T.; Nagai, Y.; Enomoto, N. Fuzzy control of a mobile robot for obstacle avoidance. // Information Sciences. 43(1988), pp. 231-248.

[5] Nilsson, N. J. A Mobile Automation: An Application of Artificial Intelligent Techniques. // Proceedings of the International Joint Conference on Artificial Intelligent, 1969. pp. 509-520.

[6] Kavraki, L.E.; Svestka, P.; Latombe, J.-C.; Overmars, M.H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. // IEEE Transactions on Robotics and Automation. 12, 4(1996), pp. 566-580.

[7] Adler, Fred P. Missile Guidance by Three-Dimensional Proportional Navigation. // Journal of Applied Physics. 27, 5(1956), pp. 500-507.

[8] Yang, C. D.; Yang, C. C.; A unified approach to proportional navigation. // IEEE Transactions on Aerospace and Electronic Systems, 33(1997), pp. 557-567.

[9] Sutton, R. S. Reinforcement Learning. // Boston: Kluwer Academic Publishers. 1992.

[10] Sutton, R. S.; Barto, A. G. Reinforcement Learning-An Introduction. // The MIT Press. Cambridge, 1998.

[11] Watkins, C. J. C. H., Learning from delayed rewards. // Doctoral thesis, Cambridge University. Cambridge, England, 1989.

[12] Watkins, C. J. C. H.; Dayan, P. Technical Note: Q-learning. // Machine Learning, 8(1992), pp. 279-292.

[13] Kaelbling, L. P.; Littman, M. L.; Moore, A. W. Reinforcement Learning: A Survey. // Journal of Artificial Intelligence Research. 4(1996), pp. 237-285.

[14] Ngai, D. C. K.; Yung, N. H. C. Double action Q-learning for obstacle avoidance in a dynamically changing environment.// Proceedings of the 2005 IEEE Intelligent Vehicles Symposium. Nevada, USA, 2005. pp. 211-216.

[15] Ngai, D. C. K.; Yung, N. H. C. Performance Evaluation of Double Action Q-Learning in Moving Obstacle Avoidance Problem. // Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Waikoloa, USA, 2005. pp. 865–870,.

[16] Laurent, G.; Piat, E. Parallel Q-Learning for a blockpushing problem. // Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Maui, USA, 2001.

[17] Laurent, G.; Piat, E. Learning Mixed Behaviours with Parallel Q-learning. // Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems. Lausanne, Switzerland, 2002.

[18] Ngai, D. C. K.; Yung, N. H. C. Automated Vehicle Overtaking Based on a Multiple-Goal Reinforcement Learning Framework. // Proceedings of the IEEE International Conference on Intelligent Transportation Systems. Seattle, USA, 2007. pp. 818–823.

Authors' Addresses

Josip Job
Faculty of Electrical Engineering
J. J. Strossmayer University of Osijek
Cara Hadrijana bb
31000 Osijek, Croatia
josip.job@etfos.hr

Franjo Jović
Faculty of Electrical Engineering
J. J. Strossmayer University of Osijek
Cara Hadrijana bb
31000 Osijek, Croatia
franjo.jovic@etfos.hr

Časlav Livada
Faculty of Electrical Engineering
J. J. Strossmayer University of Osijek
Cara Hadrijana bb
31000 Osijek, Croatia
caslav.livada@etfos.hr