POLISH MARITIME RESEARCH Special Issue S1 (74) 2012 Vol 19; pp. 31-36
10.2478/v10012-012-0020-8

Reinforcement Learning in Discrete and Continuous Domains Applied to Ship Trajectory Generation

Andrzej Rak, M. Sc.,
Witold Gierusz, Ph. D.,
Gdynia Maritime University

ABSTRACT


This paper presents the application of the reinforcement learning algorithms to the task of autonomous determination of the ship trajectory during the in-harbour and harbour approaching manoeuvres. Authors used Markov decision processes formalism to build up the background of algorithm presentation. Two versions of RL algorithms were tested in the simulations: discrete (Q-learning) and continuous form (Least-Squares Policy Iteration). The results show that in both cases ship trajectory can be found. However discrete Q-learning algorithm suffered from many limitations (mainly curse of dimensionality) and practically is not applicable to the examined task. On the other hand, LSPI gave promising results. To be fully operational, proposed solution should be extended by taking into account ship heading and velocity and coupling with advanced multi-variable controller.

Keywords: ship motion control; trajectory generation; autonomous navigation; reinforcement learning; least-squares policy iteration

INTRODUCTION

Development of control systems in recent years is determined by the expansion of relatively cheap computing power. This process is obviously present in the field of ship motion control systems too. As a result one can observe a progress from course keeping autopilots to trajectory tracking autopilots and further to integrated motion control systems with functionality of route planning, anti-collision subsystem and advanced multi-variable autopilot [3].

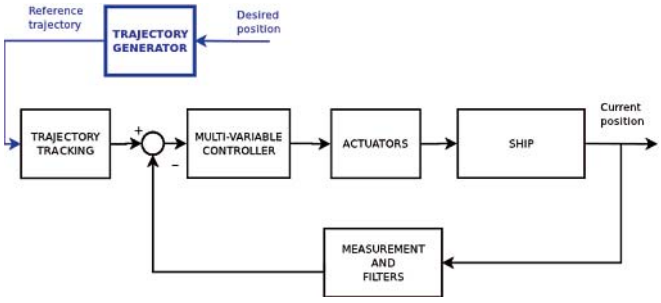

Fig. 1. *Block diagram of the proposed extension of the ship motion control system*

If the system is to be fully autonomous it should incorporate block of trajectory determination for in-harbour and harbour approaching manoeuvres. This task is significantly different from open waters or restricted areas anti-collision manoeuvres. The ship is moving with relatively small velocities and trajectory should be determined simultaneously for position, transversal and longitudinal velocities as well as heading signals. Trajectory generator has to cooperate with multi-variable autopilot capable to control all of mentioned signals [3, 4, 8]. The proposed location of such block in ship motion control system is shown in Fig. 1.

Present paper describes the results of application of reinforcement learning (RL) algorithms to the generation of a ship trajectory. RL applications in the ship motion control system are quite rare. There are only a few examples in the openly accessible bibliographical sources [7, 9, 12].

In the first section the idea of the reinforcement learning and Markov decision processes (MDP) as a formal notation of RL problem are discussed, They deliver theoretical background of the algorithms. In the second and third sections learning algorithms in the discrete and continuous domains are presented. The results of computer simulations are shown in section fourth. Fifth section concludes the paper.

REINFORCEMENT LEARNING AND MARKOV DECISION PROCESSES

In reinforcement learning procedure a controller (agent) interacts with the process (environment) by means of three signals: a state signal which determines current state of the process, action signal which is used by the controller to

influence the process and a reward given by the reward function which gives the measure of the controller performance (see Fig. 2). In each consecutive time step the controller observes a state of a process and issues an action to move the process to a next state. At the same moment the reward function, based on a state and chosen action, evaluates this move issuing value of reward. In a next time instance whole procedure is repeated. The goal of the reinforcement learning process is to find (learn) a strategy of action selection for the controller which maximises the cumulative reward in a long term.

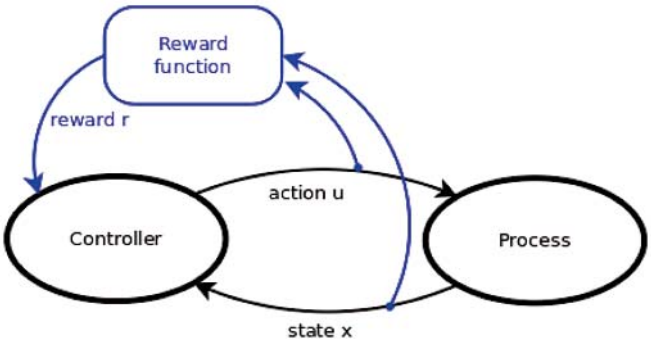

Fig. 2. *Interactions in the reinforcement learning process*

The dynamics of the process can be deterministic or stochastic. In this paper the deterministic case will be discussed: it means we assume that in particular state choice of certain action always gives the same next state[1)]. Extension to the stochastic case can be easy find in the references [1, 2, 10].

It is easy to notice that the problem of reinforcement learning can be described in the Markov decision process (MDP) formalism. Markov decision process is defined as a quadruple (X, U, f, p) where $X = \{x_1, x_2, ..., x_n\}$ is finite set of states; $U = \{u_1, u_2, ..., u_n\}$ is finite set of actions and $f: X \times U \rightarrow X$ is transitions function determining the state in a next time step:

$$x_{k+1} = f(x_k, u_k) \tag{1}$$

At the same time the controller receives the value of reward according to the reward function $\rho: X \times U \rightarrow [\ \mathbb{R}$:

$$r_{k+1} = \rho(x_k, u_k) \tag{2}$$

where we assume that $\|\rho\|_\infty = \sup_{x,u}|\rho(x, u)|$

The controller chooses action according to its own policy $h: X \rightarrow U$, using:

$$u_k = h(x_k) \tag{3}$$

Taking into account above definitions we can state that given functions f and ρ as well as current state x_k and action u_k are sufficient to determine the next state x_{k+1} and reward r_{k+1}. This fulfils Markov property.

LEARNING IN DISCRETE SPACE

As we mentioned in the previous section, the goal of RL is to find an optimal policy that maximises the return from any initial state x_0. The return is cumulative value of rewards collected along a trajectory originating in x_0. There exist a few types of return definitions. We will use one of them; infinite horizon discounted sum:

$$R^h(x_0) = \sum_{k=0}^{\infty} \gamma^k \rho[x_k, h(x_k)] \tag{4}$$

[1)] This condition is partly violated by the need of exploration of the discrete RL algorithms [10].

The infinite horizon return has better theoretical properties leading to the stationary optimal policies. But in a practical case the sum is limited by the number of steps in a trajectory between initial and final states.

Discount factor γ controls a trade-off between the quality of the solution and convergence rate of RL algorithm and usually is set by trial and error procedure.

A convenient way to represent policies are their value functions [10]. In an area of RL two types value functions are used: state value function (V) and state-action value function (Q). The latter one is more general and incorporates the first. In this section algorithms based on the Q-functions are presented. It is a mapping $Qh: X \times U \rightarrow [\ \mathbb{R}$ and represents a reward of choosing action u in a state x according to the followed policy h cumulated with the return from the next state [1].

$$Q^h(x, u) = \rho(x, u) + \gamma R^h[f(x, u)] \tag{5}$$

The optimal Q-function is defined as the best Q-function that can be obtained by any policy:

$$Q^*(x, u) = \max_h Q^h(x, u) \tag{6}$$

Any policy h^* that selects at each state an action with the largest optimal Q-value i.e., that satisfies;

$$h^*(x) \in \arg\max_h Q^*(x, u) \tag{7}$$

is optimal. In general, for a given Q-function Q, a policy h that satisfies:

$$h(x) \in \arg\max_h Q(x, u) \tag{8}$$

is said to be greedy in Q. So finding an optimal policy can be done by first finding Q^*, and then using (7) to compute a greedy policy in it. If a process for which the learning is applied (Fig. 2) is known (we have exact model of the process), Q-functions Q^h and Q^* can be easily found from the iterative Bellman equations [2, 10]. This leads to the dynamic programming Q-iteration algorithms [1].

In the case of RL the model of the process is unknown, so the next state and reward values are collected in the interactions with it. Therefore these algorithms are often called model-free. One of them, most widely used, is Q-learning:

$$\begin{aligned} Q_{k+1}(x_k, u_k) &= Q_k(x_k, u_k) + \\ &+ \alpha_k\left[r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1}, u') - Q_k(x_k, u_k)\right] \end{aligned} \tag{9}$$

where $a_k \in (0,1]$ is the learning rate. The term between square brackets is the temporal difference, i.e., the difference between the updated estimate $r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1}, u')$ of the optimal Q-value of (x_k, u_k) and the current estimate $Q_k(x_k, u_k)$.

As the number of transitions k approaches infinity Q-learning asymptotically converges to Q^* if the state and action spaces are discrete and finite, and under the following conditions [11]:

- the sum $\sum_{k=0}^{\infty} \alpha_k^2$ gives a finite value, whereas the sum $\sum_{k=0}^{\infty} \alpha_k$ produces an infinite value.
- all the state-action pairs are (asymptotically) visited infinitely often.

The first condition is easy to satisfy. To fulfil the second one, a stochastic parameter ε is introduced. It represents the probability of selection of any action in encountered state. This is called exploration. On the contrary, the controller should also exploit current knowledge to improve performance by

selecting greedy actions in the current Q-function. The balance of exploration-exploitation is usually implemented in a form of ε-greedy exploitation [10].

Algorithm 1. Q-learning with ε-greedy exploration

Input: discount factor γ,
exploration schedule $\{\varepsilon_k\}_{k=0}^{\infty}$, learning rate schedule $\{\alpha_k\}_{k=0}^{\infty}$

1: initialise Q-function, e.g, $Q_0 \leftarrow 0$
2: measure initial state x_0
3: **for** every time step k = 1, 2, ... **do**
4: $u_k \leftarrow \begin{cases} u \in \arg\max_{\bar{u}} Q_k(x_k, \bar{u}) & \text{with probability } 1-k \\ \text{a uniformly random in } U & \text{with probability } k \end{cases}$
5: apply u_k, measure next state x_{k+1} and reward r_{k+1}
6: $Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \alpha_k[r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1}, u') - Q_k(x_k, u_k)]$
7: **end for**

The complete algorithm, developed from equation (9) is presented in the frame Algorithm 1 in this section. This version of Q-learning was used in the computer simulations.

LEARNING IN CONTINUOUS SPACES USING FUNCTION APPROXIMATORS

Discrete algorithm for RL has a significant drawback. It requires an exact representation of a value function and policy. It means distinct values of the return estimates for each state-action pair has to be stored as well as actions for every state. When an action space or state space is large this can be extremely difficult or practically impossible. To reduce a number of parameters that has to be stored approximation techniques are used.

Generally in the RL algorithms approximation is used not only for function representation. Policy iteration which will be introduced in the subsequent parts of this paper must repeatedly solve potentially difficult maximisation problems over the action variables (policy evaluation). This can be done by sample-based approximation. In this research parametric approximation was used.

Parametric approximators are mappings from a parameter space into a space of functions [6]. The functional form of the mapping and the number of parameters are usually set by the skilled operator in advance and do not depend on data collected during the interaction with the process. The parameters are tuned using the data about target function.

Let as consider the Q-function approximator parameterised by an n-dimensional vector θ. The approximator is the mapping $F:\mathbb{R}^n \rightarrow \Im$ where $\mathbb{R}^n$ is the parameter space and $\Im$ is the space of Q-functions. Every parameter vector θ provides a compact representation of a corresponding approximate Q-function:

$$\hat{Q}(x,u) = [F(\theta)](x,u) \tag{10}$$

where $[F(\theta)](x,u)$ denotes the Q-function evaluated at the state-action pair (x,u). Therefore, instead of storing distinct Q-values for every pair (x,u), it is only necessary to store n parameters. But, it must be noticed that, since the set of Q-functions represent by F is only a subset of $\Im$ any arbitrary Q-function, it can be reproduced only up to the certain approximation error [5].

The mapping $F(\theta)$ can be generally non-linear. However, linearly parameterised approximators are preferred because they simplify an analysis of resulting RL algorithm. In presented algorithms we use linear approximators built with n Gaussian normalised radial basis functions (BF) $\phi_1, \ldots \phi_n: X \times U \rightarrow [\mathbb{R}$ and n-dimensional vector of parameters θ. Approximate values were therefore computed as:

$$[F(\theta)](x,u) = \sum_{l=1}^{n} \phi_l(x,u)\theta_l = \phi^T(x,u)\theta \tag{11}$$

Let us consider now model-based value iteration with parametric approximation. This procedure explains the ideas used in a final algorithm. One can observe that iteration formula (9) rewritten for model-based case can be generally stated as:

$$Q_{l+1} = T(Q_l) \tag{12}$$

for consecutive l iterations, where T is Q-iteration mapping [1]. In approximate Q-iteration Q_l cannot be represented exactly. Therefore, an approximation (10) has to be used:

$$\hat{Q}_l = F(\theta_l) \tag{13}$$

Using this approximation in iteration formula (12) leads to:

$$Q'_{l+1} = (T \circ F)(\theta_l) \tag{14}$$

But the function T cannot be stored explicitly either. Instead it is represented by approximation using new parameter vector θ_{l+1}. This vector is specified by the projection mapping $P : \Im \rightarrow \mathbb{R}^n$:

$$\theta_{l+1} = P\left(Q'_{l+1}\right) \tag{15}$$

which should keep $\hat{Q}_{l+1} = F(\theta_{l+1})$ as close as possible to $Q'\hat{Q}_{l+1}$. Usually a least-squares regression is chosen for P. Finally, approximate Q-iteration is a composition of mappings:

$$\theta_{l+1} = (P \circ T \circ F)(\theta_l) \tag{16}$$

The algorithm should be stopped when suitable parameter vector $\hat{\theta}^*$ is found. Eventually the estimate $\hat{\theta}^*$ should be kept as close as possible to the fixed point θ^* of iteration (16). The whole procedure is illustrated on Fig. 3. The cycle of mappings: Q-value approximation (F), Q-iteration (T) and projection back to parameter space (P) is repeated until fixed point θ^* is reached.

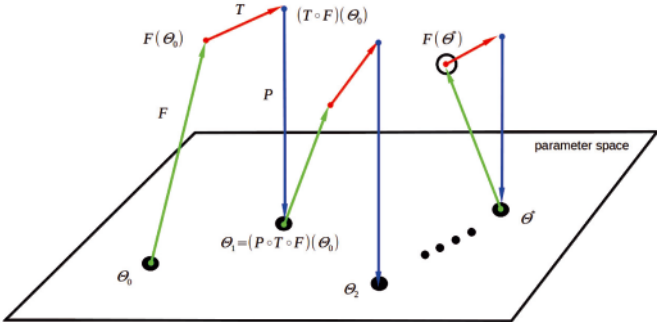

Fig. 3. *An idea of approximate Q-iteration*

Similar considerations lead to the analogous approximate policy evaluation algorithm for Q-functions. This algorithm starts from arbitrary chosen vector of parameters θ_0^h and updates this vector in every iteration τ using:

$$\theta_{\tau+1}^h = (P \circ T^h \circ F)(\theta_\tau^h) \tag{17}$$

As we mentioned before, approximators used in these considerations have linear properties. Based on this, it is possible to derive projected Bellman equation. Because state and action spaces are now finite by assumption we can rewrite policy evaluation mapping T^h.

$$[T^h(Q)](x,u) = \sum_{x'} \bar{f}(x,u,x')[\rho(x,u,x') + \gamma Q(x,h(x'))] \tag{18}$$

In a linear case approximate Q-function that has the form of (13) can be written as:

$$\hat{Q}^h(x, u) = \phi^T(x, u)\theta^h \tag{19}$$

where $\phi(x,u) = [\phi_1(x,u), ..., \phi_n(x,u)]^2$ is the vector of BFs and θ^h is the vector of parameters. This relationship satisfies approximate version of Bellman equation (projected Bellman equation):

$$\hat{Q}^h = (P^w \circ T^h)(\hat{Q}^h) \tag{20}$$

where P^w performs a weighted least-squares projection onto the space of approximate Q-functions spanned by the BFs. To derive proper algorithm let us rewrite (18) in a matrix form:

$$\mathbf{T}^h(\mathbf{Q}) = \bar{\boldsymbol{\rho}} + \gamma \bar{\mathbf{f}}\mathbf{h}\mathbf{Q} \tag{21}$$

Using symbols $\boldsymbol{\phi}$ to denote BF matrix and $\mathbf{w}$ to denote diagonal weight matrix of P^w we can write:

$$\hat{\mathbf{Q}} = \boldsymbol{\varphi}\boldsymbol{\theta} \tag{22}$$

and Bellman equation (20) as:

$$\mathbf{P}^w\mathbf{T}^h(\hat{\mathbf{Q}}^h) = \hat{\mathbf{Q}}^h \tag{23}$$

Rearranging this equation and substituting:

$$\begin{aligned} \Gamma &= \boldsymbol{\varphi}^T\mathbf{w}\boldsymbol{\varphi} \\ \Lambda &= \boldsymbol{\varphi}^T\mathbf{w}\bar{\mathbf{f}}\mathbf{h}\boldsymbol{\varphi} \\ z &= \boldsymbol{\varphi}^T\mathbf{w}\bar{\boldsymbol{\rho}} \end{aligned} \tag{24}$$

the projected Bellman equation can be written in a final form:

$$\Gamma\theta^h = \gamma\Lambda\theta^h + z \tag{25}$$

Solving this equation leads directly to the Least-Squares Policy Iteration (LSPI) algorithm presented in a frame Algorithm 2 [6].

This procedure was employed in simulation software used to the generation of the ship trajectory in a continuous state space. The next section presents some of the simulation results.

EXAMPLE OF SHIP TRAJECTORY GENERATION FOR NAVIGATION IN RESTRICTED WATERS

Algorithm 2. Offline least-squares policy iteration

Input: discount factor γ,
BFs $\phi_1,\ldots,\phi_n : X \times U \to R$, samples $\{(x_{l_s}, u_{l_s}, x'_{l_s}, r_{l_s}) | l_s = 1,\ldots,n_s\}$

1: initialise policy h_0
2: **repeat** at every iteration $l = 0, 1, 2, ...$
3: $\Gamma_0 \leftarrow 0, \Lambda_0 \leftarrow 0, z_0 \leftarrow 0$
4: **for** $l_s = 1, ..., n_s$ **do**
5: $\Gamma_{l_s} \leftarrow \Gamma_{l_s-1} + \phi(x_{l_s}, u_{l_s})\phi^T(x_{l_s}, u_{l_s})$
6: $\Lambda_{l_s} \leftarrow \Lambda_{l_s-1} + \phi(x_{l_s}, u_{l_s})\phi^T[x'_{l_s}, h(x'_{l_s})]$
7: $z_{l_s} \leftarrow z_{l_s-1} + \phi(x_{l_s}, u_{l_s})r_{l_s}$
8: **end for**
9: solve $\frac{1}{n_s}\Gamma_{n_s}\theta_l = \gamma\frac{1}{n_s}\Lambda_{n_s}\theta_l + \frac{1}{n_s}z_{n_s}$
10: $h_{l+1}(x) \leftarrow u$, where $u \in \text{argmax}_{\bar{u}}\phi^T(x, \bar{u})\theta_l, \quad \forall x$
11: **until** h_{l+1} is satisfactory
Output: $\hat{h}^* = h_{l+1}$

The simulation experiments were done in the MATLAB environment. First the Q-learning algorithm was tested. Two variants were implemented:

- Classic one: where the state transition can be done only to the neighbour-state in the discrete grid. It means that there are seven possible actions to chose. Six of them are "king moves" [10] to the neighbour state, and seventh is the loopback to the current state. The value function was set to -1 for all state transfers except last move to the goal state which yields 0 reward[2)].
- Modified one: prepared to overcome limitations of neighbour-state only transitions. In this case action chosen can "jump" from every state to the any of states that are not blocked by obstacles. The reward is equal to the negative Euclidean distance from k + 1 state to the goal state. In this case action value function has to store n^2 distinct values, where n is the number of states in the whole grid.

The results of the simulation for classic (step-by-step) variant are depicted on Fig. 4 and Fig. 5. Q-value function on Fig. 4 is marked by colours. More red for particular state means that the return (negative) associated with this state is bigger. It is easy to notice that trajectories from all states will tend towards the goal state; even starting point for the learning algorithm was not changed. This property is a result of ε-greedy exploration part of the Algorithm 1.

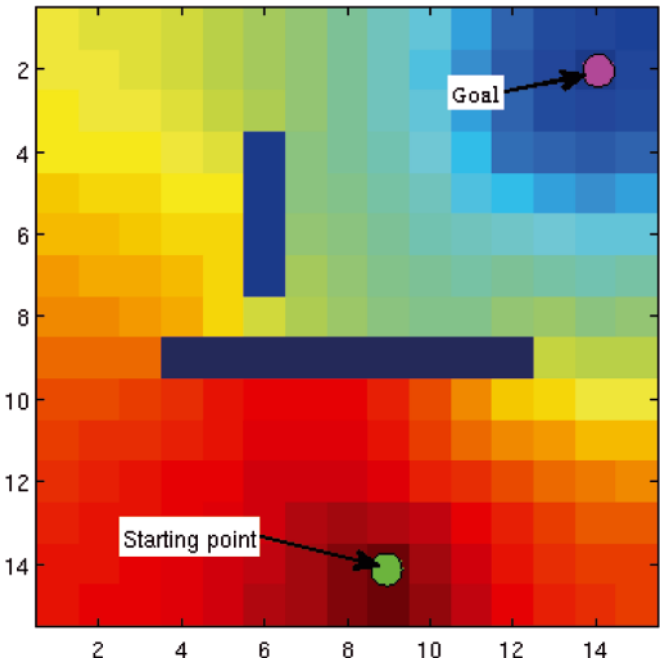

Fig. 4. *Q-value surface of Q-learning with step-by-step type strategy. Corresponding proposed trajectory is presented on the Fig. 5. Obstacles are dark blue. (2500 episodes)*

The corresponding greedy trajectory originating from the starting state (9, 14) is shown on Fig. 5. It can be easy noticed that it is not unique trajectory passing minimal number of state on the way to the goal.

Results of the second variant (point-to-point) of the Q-learning algorithm are presented in Fig. 6. In this and next figure the colour map for values is reversed. Except modifications mentioned above, one more change to the standard algorithm

[2)] Very often the reward is of negative value. It means it works as a "punishment", not a "reward". However it is a standard in RL not to change the term "reward" even for negative values.

was done. Along with the obstacle collision check in every step, there is a test if there exists straight, direct path from the current state to the goal. If so, then the state transition is done directly to the goal and algorithm finishes. This modification caused brown "shadows" over certain states of the grid. Because these states lie on the "open" side of the obstacles, learning algorithm did not "visit" them. The proposed modifications accelerated convergence of the solution.

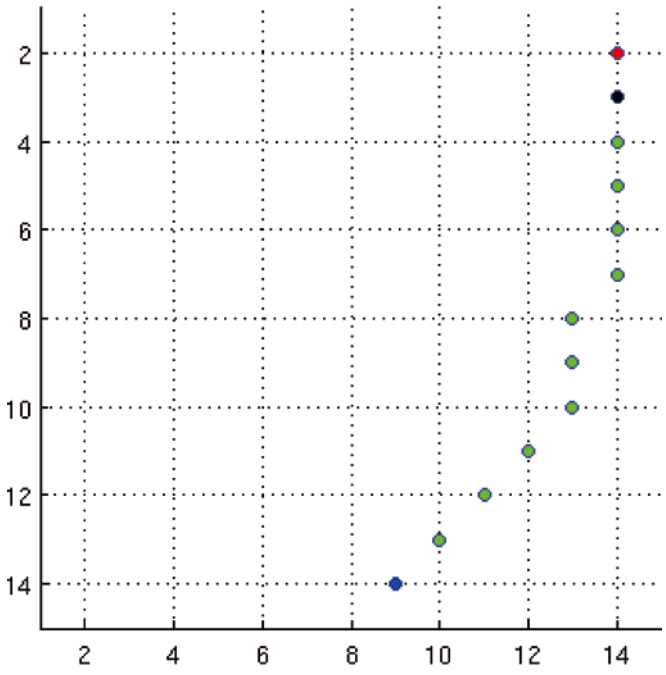

Fig. 5. *Final greedy trajectory of Q-learning with step-by-step type strategy - see Fig. 4 (2500 episodes)*

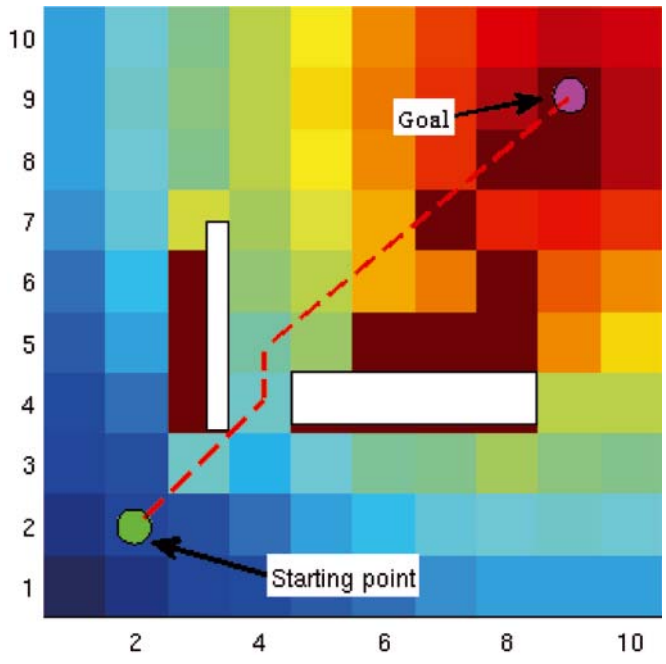

Fig. 6. *Results of Q-learning with point-to-point type strategy. Proposed trajectory is marked by the red doted line. Obstacles are white. (15 000 episodes)*

On the contrary to the step-by step variant this one produces unique, broken-line trajectory. Unfortunately, this version is exceptionally computing power demanding. On the Fig. 4 there are 15 x 15 = 225 states in the grid. Multiplying them by 7 possible actions yields 1575 state-action pairs for Q-function evaluation. The same grid for the second case will give $(15 \times 15)^2 = 50525$ state action pairs to process. This number will expand very quickly if one wants to improve the precision of the ship positioning by the grid refinement or supplement a heading angle.

It means that Q-learning algorithms in a discrete version are of limited usability for manoeuvring ship trajectory generation.

Last figure presents the LSPI approximate Q-value surface (precisely: one of the cross sections through it) given by linear approximation over a (5x5) grid of RBFs. Additionally, action space was also approximated by (11x11) grid of RBFs. As one can notice, there is not the starting point for the learning algorithm. Version of the LSPI algorithm implemented in the research works in a batch mode doing calculations off-line for all previously collected data samples. The starting point was set to show only an exemplary trajectory. Whole surface is parameterised, therefore the problem of the grid scale mentioned earlier vanishes. The same factor makes it susceptible to heading control extension.

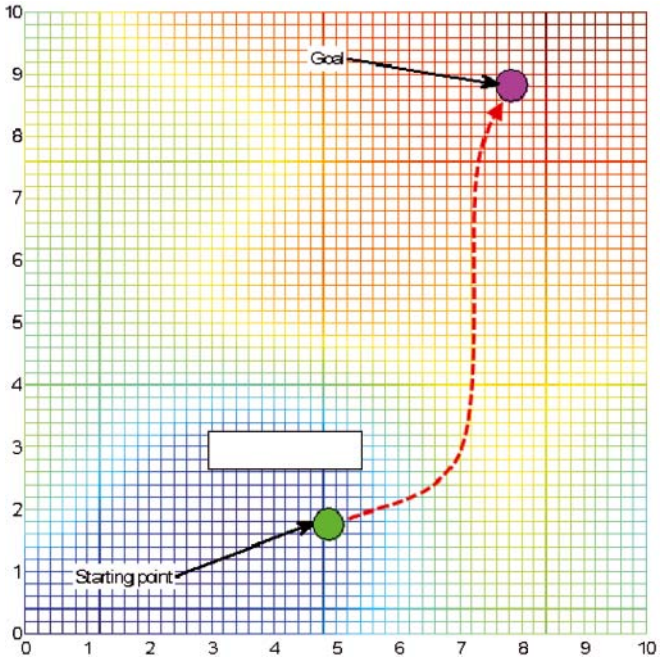

Fig. 7. *Least-Squares Policy Iteration. Surface of approximate Q-value function with proposed trajectory. Obstacle is white. (11 iterations for 17000 samples)*

CONCLUSIONS

Results of the research presented in a previous section allow conclude that:

- The discrete Q-learning algorithms can be used for ship trajectory generation only when precision of the motions is not of major importance or the maneuvring area is significantly small.
- Point-to-point version of the algorithm is very vulnerable to the curse of dimensionality.
- Off-line, continuous domain, LSPI algorithm seems to be good alternative to the discrete Q-learning. To test all its properties in a context of ship trajectory generation, it should be expanded to incorporate heading set-point values and, in a next step, longitudinal and transversal velocities.

BIBLIOGRAPHY

1. Busoniu L., Babuska R., De Schutter B., Ernst D.: *Reinforcement Learning and Dynamic Programming Using Function Approximators***.** CRC Press. *Automation and Control Engineering S*eries. 2010.
2. Cichosz P.: *Learning Systems*. WNT. Warszawa 2000. (in Polish).
3. Gierusz W.: *Synthesis of Multi-variable systems of Precise Ship Motion Control Using Selected Robust Control Design Methods.* Gdynia Maritime University. 2005. (in Polish).
4. Gierusz W., Nguyen Cong V., Rak A.: *Maneuvering Control and Trajectory Tracking of Very Large Crude Carrier*, Ocean Engineering, 2007. No 34, pp. 932-945.
5. Kudrewicz J.: *Functional Analysis for Control and Electronics Engineers*. PWN. Warszawa 1976. (in Polish).
6. Lagoudakis M. G., Parr R.: *Least-Squares Policy Iteration*. Journal of Machine Learning Research. 2003. Vol. 4, pp. 1107-1149.
7. Mitsubori K., Kamio T., Tanaka T.: *On a Course Determination based on the Reinforcement Learning in Maneuvering motion of a ship with the tidal current effect.* International Symposium on Nonlinear Theory and its Applications. Xi'an 2002.
8. Morawski L., Nguyen Cong V., Rak A.: *Full-Mission Marine Autopilot Based on Fuzzy Logic Techniques*. Gdynia Maritime University. 2008.
9. Rak A.: *Application of Reinforcement Learning to Ship Motion Control Systems.* Zeszyty Naukowe AM w Gdyni. 2009. No 62, pp. 133-140. (in Polish).
10. Sutton R. S., Barto A. G.: *Reinforcement Learning An Introduction*. MIT Press. 1998.
11. Watkins C. J. C. H., Dayan P.: *Q-learning*. Machine Learning. 1992. Vol. 8, no 3-4, pp. 279-292.
12. Zhipeng S., Chen G., Jianbo S.: *Reinforcement learning control for ship steering based on general fuzzified CMAC*. Proc. of 5-th Asian Control Conference. 2005. Vol. 3, pp. 1552-1557.

CONTACT WITH THE AUTHOR

Andrzej Rak, M. Sc.,
Witold Gierusz, Ph. D.,
Faculty of Marine Electrical Engineering,
Gdynia Maritime University,
Morska 81-87
81-225 Gdynia, POLAND
e-mail: anrak@am.gdynia.pl
e-mail: wgierusz@am.gdynia.pl