ISSN 0005-1179 (print), ISSN 1608-3032 (online), Automation and Remote Control, 2023, Vol. 84, No. 3, pp. 221–236.
Trapeznikov Institute of Control Sciences, Russian Academy of Sciences, 2023.

NONLINEAR SYSTEMS

Neural Network Algorithm for Intercepting Targets Moving Along Known Trajectories by a Dubins' Car

A. A. Galyaev[*,a], **A. I. Medvedev**[*,b], **and I. A. Nasonov**[*,c]

[*] *Trapeznikov Institute of Control Sciences, Russian Academy of Sciences, Moscow, Russia*
e-mail: [a] *galaev@ipu.ru,* [b] *medvedev.ai18@physics.msu.ru,* [c] *nasonov.ia18@physics.msu.ru*

Received July 28, 2022
Revised November 17, 2022
Accepted November 30, 2022

Abstract—The task of intercepting a target moving along a rectilinear or circular trajectory by a Dubins' car is formulated as a problem of time-optimal control with an arbitrary direction of the car's velocity at the time of interception. To solve this problem and to synthesize interception trajectories, neural network methods of unsupervised learning based on the Deep Deterministic Policy Gradient algorithm are used. The analysis of the obtained control laws and interception trajectories is carried out in comparison with the analytical solutions of the interception problem. Mathematical modeling of the target motion parameters, which the neural network had not previously seen during training, is carried out. Model experiments are conducted to test the stability of the neural solution. The effectiveness of using neural network methods for the synthesis of interception trajectories for given classes of target movements is shown.

Keywords: interception task, Dubins' car, DDPG algorithm, neural network synthesis of trajectories

DOI: 10.25728/arcRAS.2023.21.74.001

1. INTRODUCTION

The task of intercepting mobile targets moving along known trajectories has been of interest to researchers since mid-50s of the last century [1]. One of the basic models for describing the dynamics of an intercepting object is the Dubins' car model.

The first works on finding a line with a limited curvature and the minimum length connecting two given points belong to A.A. Markov. His first task in [2] was devoted to finding a curve connecting two points on a plane with minimal length and bounded curvature with a fixed exit direction from the first point. Such the task has found application in solving the problems of laying railways. In 1957, L. Dubins published a similar work [3] on finding a line of the shortest length with a limited radius of curvature connecting two points on a plane with a given direction of the exit from the first point and a given direction of entry into the second. The results proved to be useful in the study of objects with the limited turning radius and a constant speed of movement.

In [4] the non-game problem of the fastest interception of a moving target by a Dubins car is considered. It was assumed that the target was moving along the arbitrary and previously known continuous trajectory. To find the solution, the algebraic criterion of the optimality of the interception along the geodesic line and the optimal value of the interception time criterion were found.

In early studies of [5], sufficient conditions established that the optimal trajectory is curves "arc–line". These conditions impose restrictions on the ratio of the minimum radius of curvature of the trajectory of the car and the distance between the target and the car at the initial moment

of time. In [6], control has been synthesized to intercept a target along the geodesic line drawn from the beginning of the movement of the car to the intercept point, and it is assumed that the target is moving in a straight line with a constant speed.

The practical applications of the tasks of interception by the Dubins' car are quite extensive: the construction of optimal trajectories of unmanned aerial vehicles that monitor several ground targets [7], the development of algorithms that solve the travelling salesman problem [8], the construction of bypass trajectories when moving with obstacles [9]. Also, the Dubins' car model is used in the pursuit-evasion differential game. Such a game involves the presence of two agents: the pursuer must catch the target, and the escapee must evade the pursuer. An analytical solution to the problem of finding the optimal interception time and synthesis of the optimal trajectory for such a game was obtained in [4]. The problem of synthesis of intercept trajectories for objects moving along a circular trajectory was considered in [10].

The solution of the problems of interception by the Dubins' car can also be obtained with the help of computers. Recently, neural network reinforcement learning methods have been actively used for such tasks, which represent car learning technology without models and are used in cases when there is little or no data for training a neural network at all. Unlike learning with a teacher [11], who needs a set of marked-up data, reinforcement learning is based on the interaction of the agent with the environment [12]. This method is the most effective for finding a solution to the problem of pursuit-evasion.

The Actor-Critic method is used in many relevant studies. For example, in [13], Actor-Critic was used with Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) as a state encoder for racing games. In [14], a fuzzy deterministic policy gradient algorithm was used to obtain a specific physical meaning when teaching politics in the pursuit-evasion game. In [15], the Deep Deterministic Policy Gradient (DDPG) method for interacting with a continuous action space was introduced for the first time. It is this algorithm that will be used in this work for neural network synthesis of the trajectory of interception by the Dubins' car of a target moving at a constant speed along rectilinear and circular trajectories. Thanks to DDPG, it was possible for the first time to obtain a suboptimal trajectory based on a neural network solution.

The relevance of the work is due to both the demand in practice for interception algorithms for one and many moving targets, and the possibility of obtaining some new theoretical results related to the synthesis of interception trajectories. The so-called traveling salesman problem with mobile goals—Moving Target Traveling Salesman Problem (MTTSP) is of particular interest [16]. In this case, the points that need to be bypassed are moving at a given speed. An example of such a scenario is the interception of several evading (or attacking) targets, which are very important for dual-use applications. Obviously, finding the best route to intercept several mobile targets is a particularly difficult task due to the constant change in the position of targets, which significantly increases the computational costs of finding optimal solutions. It is known that a heuristic approach has been proposed in the literature to solve MTTSP.

The authors propose a synthesis of the interception trajectory based on a neural network solution, since analytical results and optimal trajectories for groups of targets are practically absent or unknown. The authors plan to scale this method for similar tasks.

The structure of the work includes 6 sections. Section 2 offers a mathematical formulation of the problem adapted for further application. Section 3 is devoted to the description of the DDPG algorithm, also ready for use in this formulation. Section 4 describes the structure of the neural network, and Section 5 contains the simulation results. In conclusion, the direction of further research is presented.

2. FORMULATION OF THE NEURAL NETWORK INTERCEPTION PROBLEM

The problem of the fastest δ-interception by the Dubins' car (pursuer) of a moving object (target) moving along two given trajectories at a constant speed is considered on the plane. As in [4], the dynamics for the pursuer was selected as

$$\begin{cases} \dot{x_P} = \cos\varphi \\ \dot{y_P} = \sin\varphi \\ \dot{\varphi} = u, \ |u(t)| \leqslant 1. \end{cases} \tag{1}$$

Here $x_P(t)$ and $y_P(t)$ are the coordinates of the Dubins' car on the Cartesian plane, $\varphi(t)$ is the angle between the direction of the pursuer's speed and the abscissa axis, and $u(t)$ is a time-dependent control that shown in Fig. 1. The coordinates and angle of the car are denoted by the vector function $P(t) = (x_P(t), y_P(t), \varphi(t))$.

The initial conditions of the system (1) are fixed:

$$x_P(0) = 0, \quad y_P(0) = 0, \quad \varphi(0) = \frac{\pi}{2}. \tag{2}$$

Continuous vector function $E(t) = (x_E(t), y_E(t))$ defines the trajectory of the target on the Cartesian plane.

The terminal condition of δ-interception for a neural network solution has the following form:

$$(x_P(T) - x_E(T))^2 + (y_P(T) - y_E(T))^2 \leqslant \delta^2, \tag{3}$$

where $T \in \mathbb{R}_0^+$—the time of movement from the starting point to the interception point, and δ—the specified interception radius—the maximum allowable distance between the pursuer and the target at which the interception it can be considered perfect. This parameter is introduced to define the concept of interception specifically for a neural network solution.

Let's set the task of intercepting the target in minimal time as an optimal control problem in the class of piecewise constant functions:

$$J[u] \stackrel{def}{=} \int_0^T dt \to \min_u. \tag{4}$$

Let's start describing the dynamics of the goal. According to the condition of the task, the target moves at a constant speed in a straight line or in a circle. Then the parametrized coordinate

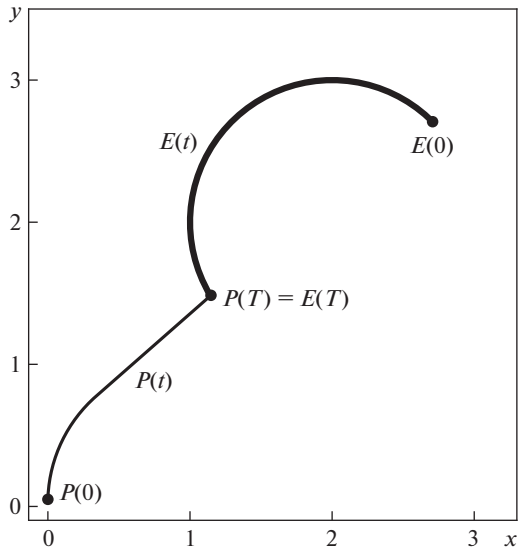

Fig. 1. Mutual location of objects.

equations will have the following form:

$$\begin{cases} x_E(t) = R\cos(\omega t + \phi) + x_0 \\ y_E(t) = R\sin(\omega t + \phi) + y_0; \end{cases} \tag{5}$$

$$\begin{cases} x_E(t) = v_x t + x_0 \\ y_E(t) = v_y t + y_0, \end{cases} \tag{6}$$

where x_0 and y_0 are the initial conditions of the target coordinates and are chosen arbitrarily.

Taking into account the relative position of the pursuer and the target, we introduce a formula for finding the angle between the abscissa axis and the straight line connecting the coordinate points of the target and the pursuer. Let (x_P, y_P) and (x_E, y_E)—the coordinates of the pursuer and the target, respectively, at some point in time t. Then the desired value of the angle is found by the formula

$$\psi = \arctan\left(\frac{y_E - y_P}{x_E - x_P}\right).$$

We will also introduce a formula for calculating the distance L between agents:

$$L = \sqrt{(x_P - x_E)^2 + (y_P - y_E)^2}.$$

Next, to simplify the study of the problem, we will make the transition to the new coordinates. To do this, you need to be able to compare the current state of agents $S = (x_P, y_P, \varphi, x_E, y_E)$ and the state predicted by the neural network $S' = (x'_P, y'_P, \varphi', x'_E, y'_E)$.

We get the values for the functions of the angles ψ and ψ' from the states S and S', respectively, and also calculate the distance L' when the agents are in the state S'. We introduce the angle between the direction of the speed of the pursuer and the line connecting the coordinate points of the agents:

$$\Theta = \varphi' - \psi'.$$

Let's introduce the rotation speed as a quotient of the difference $\psi' - \psi$ and the time interval Δt during which the transition from the state S to the state S' occurred:

$$\omega = \frac{\psi' - \psi}{\Delta t}.$$

The totality of (L', ω, Θ) and there are the desired coordinates in which we will build a neural network solution. At the initial moment of time, when the result of the neural network has not yet been received, the coordinates are $(L'(0), \omega(0), \Theta(0))$ are calculated as follows:

$$\begin{cases} L'(0) = L(S) \\ \omega(0) = 0 \\ \Theta(0) = \varphi(0) - \psi(0), \end{cases} \tag{7}$$

where $\psi(0) = \arctan\left(\frac{y_E(0) - y_P(0)}{x_E(0) - x_P(0)}\right)$.

3. ALGORITHM DEEP DETERMINISTIC POLICY GRADIENT

DDPG—is an Actor-Critic algorithm based on a deterministic policy gradient. The DPG (Deterministic Policy Gradient) algorithm consists of a parameterized function Actor $\mu(s \mid \theta^\mu)$, which sets control at the current time by deterministic matching of states with a specific action. The function Critic $Q(s, a)$ is updated using the Bellman equation in the same way as with Q training.

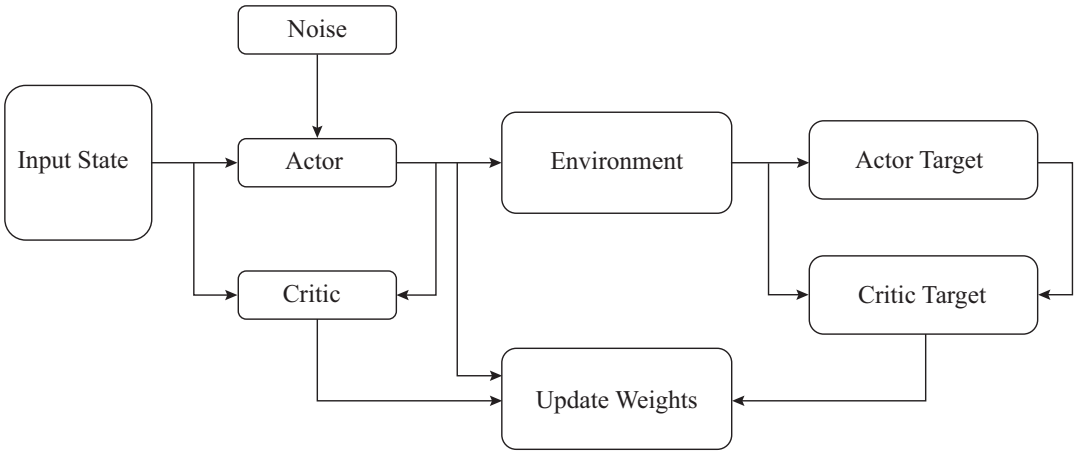

Fig. 2. The general structure of the Deep Deterministic Policy Gradient algorithm.

The Actor is updated by applying a chain rule to the expected reward from the initial distribution of J in relation to the parameters of the Actor:

$$\begin{aligned}\nabla\theta^{\mu}\mathrm{J} &\approx \mathbb{E}_{s_{\mathrm{t}}\sim\rho^{\beta}}\left[\nabla_a Q\left(s,a\mid\theta^{Q}\right)\Big|_{s=s_{\mathrm{t}},a=\mu(s_{\mathrm{t}}|\theta^{\mu})}\right]\\ &= \mathbb{E}_{s_{\mathrm{t}}\sim\rho^{\beta}}\left[\nabla_{\theta^{\mu}} Q\left(s,a\mid\theta^{Q}\right)\Big|_{s=s_{\mathrm{t}},a=\mu(s_{\mathrm{t}})}\nabla_{\theta_{\mu}}\mu\left(s\mid\theta^{\mu}\right)\Big|_{s=s_{\mathrm{t}}}\right].\end{aligned} \tag{8}$$

DDPG combines the advantages of its predecessors, which make it more stable and effective in training. Since different trajectories can be very different from each other, DDPG uses the idea of DQN [17], called a playback buffer. The playback buffer—is a finite-size buffer into which media data is stored at any given time. It is necessary to achieve a uniform distribution of the transition sample and discrete control of neural network training. Actor and Critic are updated by evenly sampling the mini-batch from the playback buffer. Another addition to DDPG was the concept of updating program targets instead of directly copying weights to the target network. Network being updated $Q\left(s,a\mid\theta^{Q}\right)$ is also used to calculate the target value, so updating Q is subject to divergence. This is possible if you make a copy of the Actor and Critic networks, $Q'\left(s,a\mid\theta^{Q'}\right)$ and $\mu'\left(s,a\mid\theta^{\mu'}\right)$. The weights of these networks are as follows: $\theta'\leftarrow\tau\theta+(1-\tau)\theta'$ with $\tau\ll 1$. The research problem is solved by adding the noise received from the noise process N to the control of the actor. In this study, the Ornstein–Uhlenbeck process is selected [18].

The general structure of DDPG is shown in Fig. 2. Since the task requires that the controls are enclosed in a numerical interval, it is necessary to introduce restrictions. To do this, the program used the *clip*() function, which limits the range of action values in the range $[-1;1]$.

Algorithm 1.

Input data: discount coefficient γ, number of episodes M, number of training steps T in each episode, batch size N, training coefficients of neural networks Actor and Critic r_a and r_c, respectively.

1. Arbitrary initialization of the networks Actor $\mu(s|\theta^{\mu})$ and Critic $Q(s,a|\theta^{Q})$
2. Initialization of target networks Q' and μ' with weight parameters $\theta^{Q}=\theta^{Q'}$ and $\theta^{\mu}=\theta^{\mu'}$
3. Initializing the R buffer
4. for episode = 1 to M do
5. Initialization of a random action $a_t=\mu(s_t|\theta^{\mu})+\eta_t$ according to the current control and research noise

6. Getting the initial state of the s_1 environment
7. for $t = 1$ to T do
8. Performing the action a_t, acquiring the reward r_t and obtaining a new state of the environment s_{t+1}
9. Saving the transition (s_t, a_t, r_t, s_{t+1}) in the buffer R
10. Random sampling of N transitions (s_i, a_i, r_i, s_{i+1}) from R
11. Getting $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
12. Updating the weights of the Critic network by minimizing the loss function

$$\widehat{L} = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

13. Updating an Actor Policy with an Effective Policy Gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

14. Updating target networks

$$\theta^{Q'} = \tau\theta^Q + (1-\tau)\theta^Q, \quad \theta^{\mu'} = \tau\theta^\mu + (1-\tau)\theta^\mu$$

15. endfor
16. endfor

Output data: Control $u = \mu(s|\theta^\mu)$

Table 1 shows the differences between the Actor, Critic networks and their target networks. It contains input and output values, as well as formulas for calculating these values.

A detailed description of the DDPG method is given in the Algorithm 1.

Table 1. Differences between Actor, Critic networks and their target networks

Network	Formula	Input data	Output data
Critic target	$Q'\left(s_{t+1}, \mu'\left(s_{t+1}\|\theta^{\mu'}\right)\|\theta^{Q'}\right)$	the next state of the environment; the output of the target network Actor	value Q', which is used to calculate y_i
Critic	$Q\left(s_t, a\|\theta^Q\right)$	current state of the environment; current action	the Q value that is needed to calculate the loss and update the Actor network
Actor target	$\mu'\left(s_{t+1}\|\theta^{\mu'}\right)$	the next state of the environment	the action μ' used as the input value of the target network Critic
Actor	$\mu\left(s_t\|\theta^u\right)$	current state of the environment	the μ action that is used to update the Actor network

4. NEURAL NETWORK

4.1. Network Architecture

To implement the Deep Deterministic Policy Gradient algorithm, two neural networks were written for each method: Critic and Actor. Their architectures are depicted in Figs. 3 and 4.

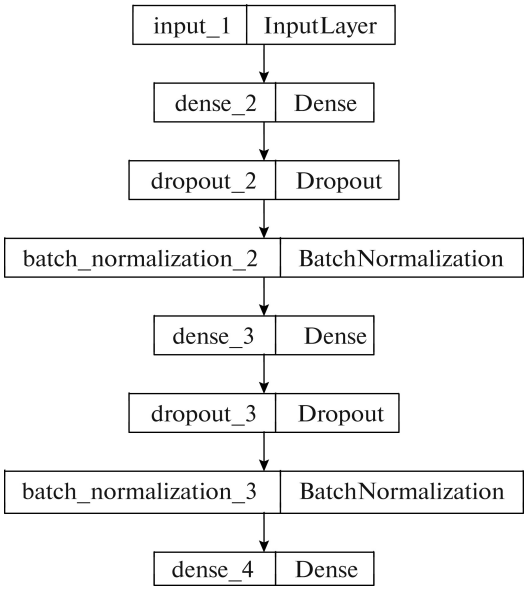

Fig. 3. Actor neural network architecture.

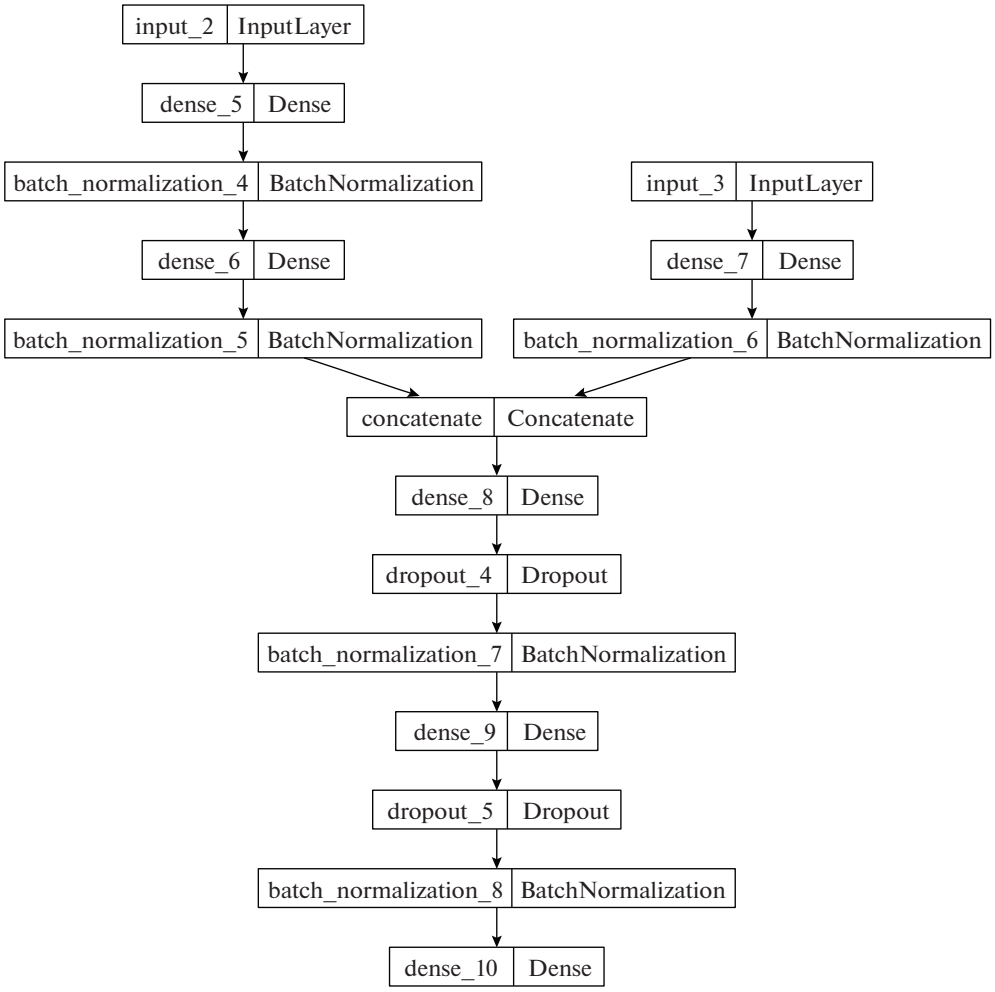

Fig. 4. The architecture of the neural network Critic.

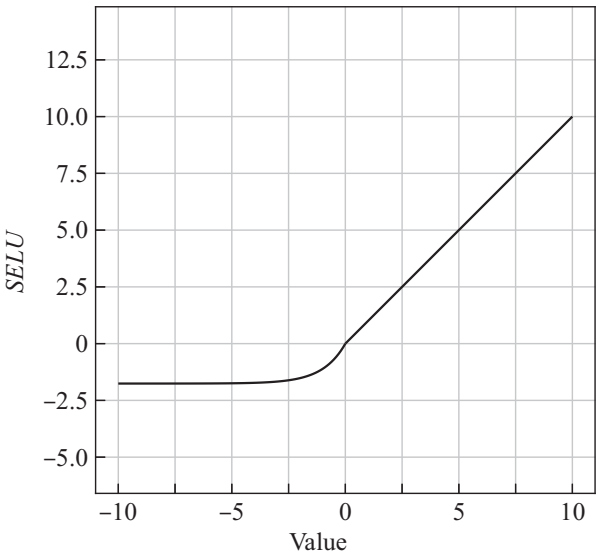

Fig. 5. Graph of the activation function $SELU$.

The Actor network has four fully connected hidden layers with 256 neurons, with $SELU$ activation function. Since the possible actions are in the range $[-1, 1]$, it is convenient to take the activation function for the output layer as *tanh*. The Critic network has five fully connected hidden layers with 16, 32, 32 and two layers with 512 neurons, with an activation function $SELU$.

The Critic and Actor networks are made up of fully connected *Dense* layers, for the output values of which the normalization operation and the *Dropout* [19] method are used, which is effective in combating the problem of retraining neural networks. To calculate the output of the Actor network from the last layer, the hyperbolic tangent activation function is selected.

The Critic network has a complex structure because it takes two input values: the state of the environment and the actions of the pursuer. Next, the layers are connected using the *Concatenate* method and the values pass through the fully connected layers of the network to the output, which is a layer of unit dimension.

4.2. Hyperparameters

The $SELU$ [20] function was chosen as the activation function in the hidden layers of the Critic and Actor neural networks, which is given by the following equation:

$$SELU(x) = \lambda \begin{cases} x, & x > 0 \\ \alpha e^x - \alpha, & x \leqslant 0, \end{cases}$$

where $\lambda \approx 1.0507$, and $\alpha \approx 1.6732$.

The graph of the $SELU$ function is shown in Fig. 5.

The $SELU$ function has the property of self-normalizing input data when using the *LeCun* initialization method, which initializes network parameters as a normal distribution. Therefore, the output values of this function have a zero mean and a single standard deviation.

In the form of a reward function for the pursuer, the following expression was chosen, depending only on the distance L between the agents:

$$r(L) = -\lg(10L) - L^2. \tag{9}$$

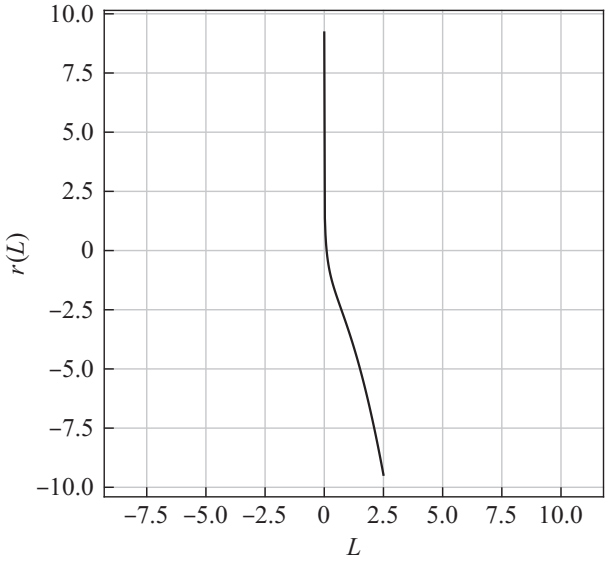

Fig. 6. A graph of the dependence of remuneration on the distance between agents.

The graph of this function is shown in Fig. 6. On it you can see that the value of r grows rapidly with a decrease in L, and when the distance takes a zero value, the agent receives the maximum reward.

The values of hyperparameters of neural networks are given in Table 2. The parameters γ, τ, episode size and time interval were selected as a result of the analysis in accordance with [14]. However, the values of the mini-batch size, buffer volume R, step size and training coefficients of Actor-Critic networks were selected empirically—the network synthesized the trajectories of intercepting the movement of the target, and then their analysis was carried out for compliance with the physical task. For example, if the average reward schedule did not increase during 100–200 training episodes, and the values of the error functions of the Actor-Critic neural networks did not decrease over the same period, then the values of the training coefficients of the networks decreased, and the size of the mini-batch increased.

Table 2. Values of neural network parameters

Parameter	Value	Description
γ	0.98	The discount factor used in the Bellman equation
τ	0.01	Coefficient of soft updating of target networks
Size mini-batch	64	Number of samples to update the weights
R Buffer Size	10 000	The amount of data from which examples are selected for updating
Episode Size	1000	Number of episodes used for training
Step Size	400	The number of training steps in each episode
Time interval	0.1	Time of each step of training
The learning coefficient of the Actor network	5e-5	The learning factor used to update the Actor network
The learning coefficient of the Critic network	1e-4	The learning factor used to update the Critic network

5. SIMULATION RESULTS

5.1. Neural Network Learning Process

The simulation was performed using Python and the TensorFlow framework. The initial parameters of the movement of the target and the pursuer during neural network training are given in Table 3.

Table 3. Initial parameters of target and pursuer movement during network training

Parameter	Value
The initial coordinate of the target movement $x_E(0)$	An arbitrary value in the interval $(-3;3)$
The initial coordinate of the target movement $y_E(0)$	An arbitrary value in the interval $(-3;3)$
Initial coordinates of the pursuer's movement $(x_P(0); y_P(0))$	$(0;0)$
Initial orientation of the pursuer $\varphi(0)$	$\pi/2$
Constant speed of the pursuer v	1
Intercept radius δ	0.2

The initial coordinates of the target movement are randomly selected using the numpy.random.uniform() function in the range $(-3;3)$ so that the network trains on different examples and works effectively after the training process. The target speeds have always had a constant value throughout the learning process $v_x = 0.5$ and $v_y = 0.5$.

Neural network training was carried out on a process with the characteristics specified in Table 4. Due to the complexity of the neural network model, the learning process lasted about four hours.

Table 4. Characteristics of the equipment where the network was trained

Parameter	Value
Processor	Intel(R) Core(TM) i7-8565U
Lithography	14 nm
Number of cores	4
Number of threads	8
Processor base clock frequency	1.80 GHz
Cache memory	8 MB
Computer RAM	16 GB

In Fig. 7 shows a graph of the average remuneration for the entire training period. During the training of the model, there is a sharp increase in the value of the agent's reward in the first 100–150 episodes. Filling of the playback buffer R corresponds to this process. Next, the training

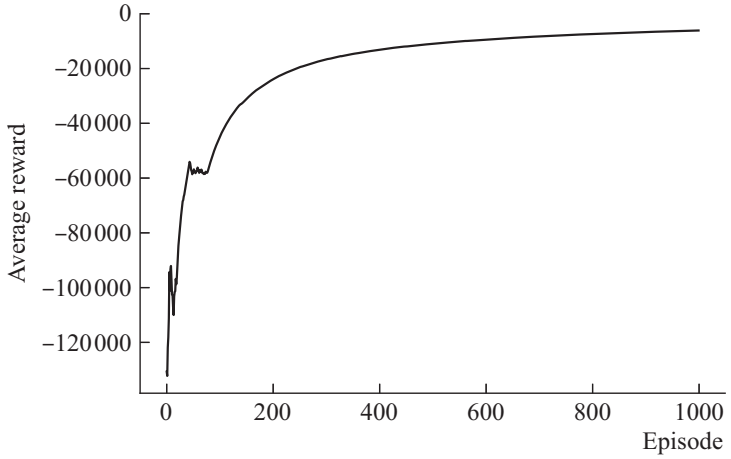

Fig. 7. The dependence of the average reward on the episode number.

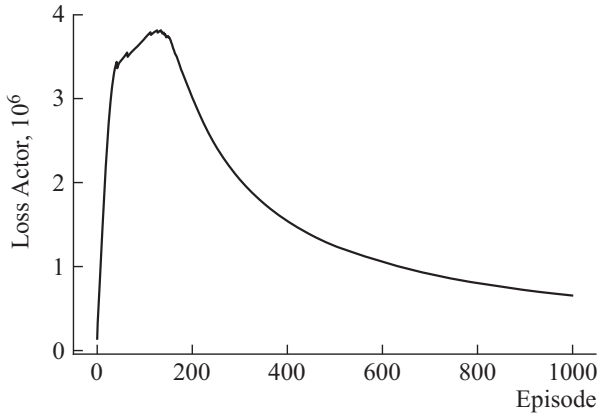

Fig. 8. The dependence of the loss function value on the episode of the Actor network.

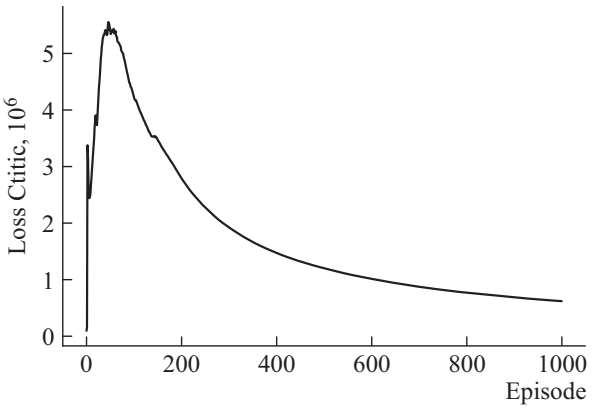

Fig. 9. The dependence of the loss function value on the episode of the Critic network.

examples are randomly taken from R, the network training process takes place and the resulting tuple of states replaces the old data sample in R. At this stage, there is a slow increase in the average remuneration, see Fig. 7.

Graphs of dependencies of the loss function of the Actor and Critic neural networks were also obtained. They are shown in Fig. 8 and 9 respectively.

The graphs show a gradual decrease in the value of the loss function with an increase in training episodes, which indicates the correct choice of training coefficients.

5.2. Learning Result

In Fig. 10 shows the trajectories obtained using a neural network and an analytical solution. The initial parameters of the target and the pursuer in this case had the values specified in Table 5.

Table 5. Initial parameters of the movement of the target and the pursuer

Parameter	Value	Value
Initial coordinates of the target movement	$(0.8; -0.4)$	$(-2.5; -0.25)$
Constant target rate v_x	0.5	0.5
Constant target rate v_y	0.5	0.5
Initial coordinates of the pursuer's movement	$(0; 0)$	$(0; 0)$
Initial orientation of the pursuer $\varphi(0)$	$\pi/2$	$\pi/2$
Constant rate of the pursuer v	1	1
Intercept radius δ	0.2	0.2

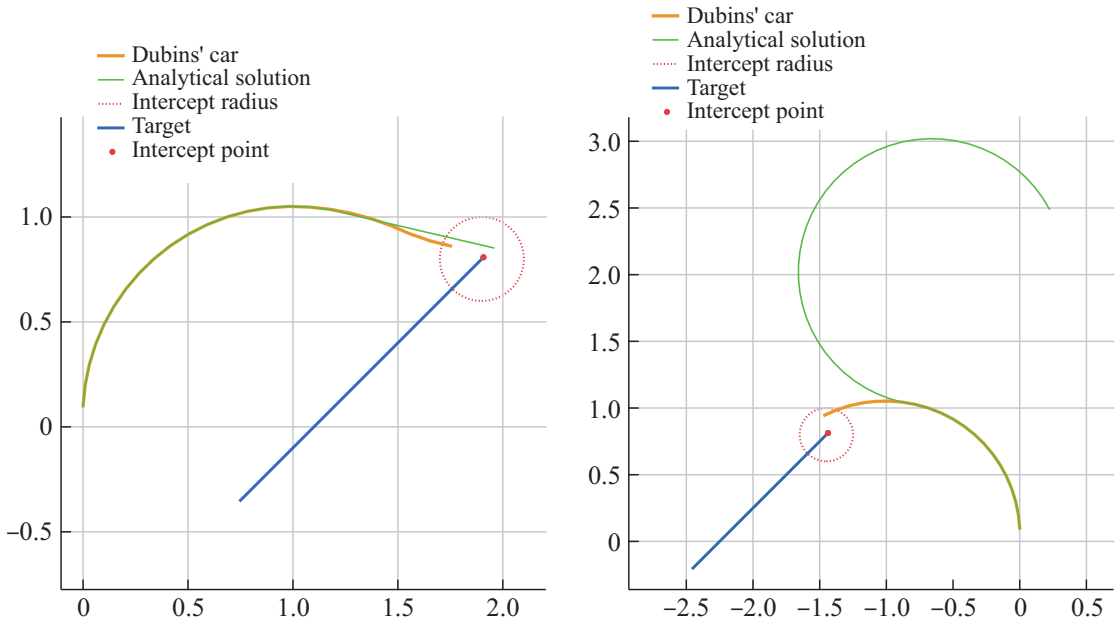

Fig. 10. Comparison of interception trajectories of a rectilinearly moving target with different initial parameters.

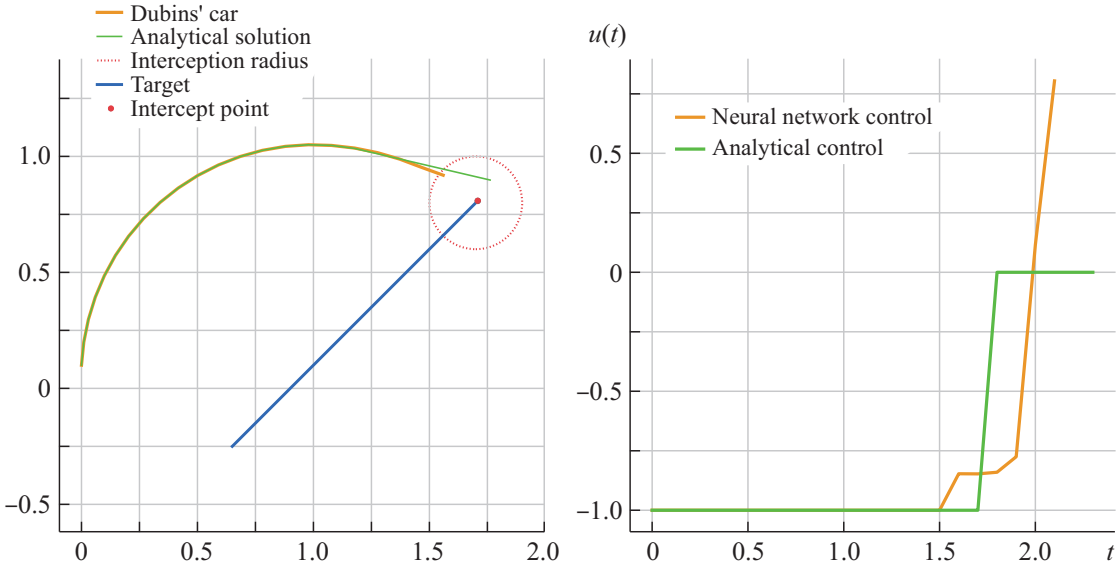

Fig. 11. Comparison of control functions from time.

Along the trajectories shown in Fig. 10, it can be seen that the network was able to build a more efficient trajectory. In this case, the optimal interception time obtained using the analytical solution is $T_{opt} \approx 5.42$ s. And the time for which the network was able to intercept the target is $T_{nn} \approx 2.1$ s. This result is explained by the presence of the intercept radius $\delta = 0.2$.

In Fig. 11 on the right you can see a comparison of neural network control graphs with analytical. As can be seen, the controls differ significantly in the final section of the trajectory due to the fact that the neural network adjusts the terminal interception conditions. Optimal synthesis in a problem with an unfixed intercept angle consists of "Arc-line" or "Arc-arc" sections [4], and in a problem with a fixed intercept angle—in general, from the "Arc-line-arc" section [21]. It is the

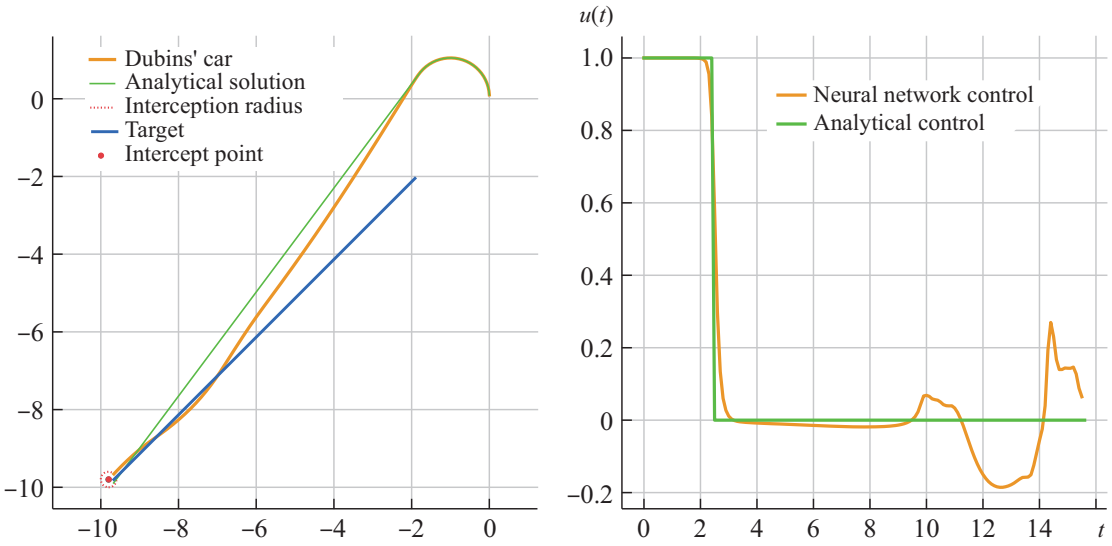

Fig. 12. Comparison of control functions from time.

latter option that synthesizes the neural network. At the same time, as can be seen from Fig. 10, there is a section of the trajectory where the neural network chooses not the optimal, but close to the optimal value of the turning radius. The second reason for the difference is that the neural network optimizes the local reward function, which is different from the performance functional that was used when setting the task.

In Fig. 12 shows the trajectories of intercepting the target and the dependence of the control function on time. On the right graph, it can be observed that the neural network control function has values close to optimal in the area where the analytical solution gives zero control. In addition, the deviations of the neural network control do not exceed the value of the intercept radius δ. The interception times in this case are almost identical: $T_{opt} \approx T_{nn} \approx 21$ s.

5.3. Sensitivity Analysis

Let's analyze how much the neural network solution depends on the input parameters, since in theory the neural network should generalize the resulting solution well to states and parameters that it has not yet "seen" during training.

To intercept a target moving in a circle, we will train the neural network only on targets with a single radius and a single angular velocity and check whether it can successfully catch a target with other parameters. As can be seen in Fig. 13, the network successfully copes with the task, in the left figure the angular velocity of the target is 0.7 of the angular velocity used in training, in the right figure the interception of an ordinary target is depicted. Experiments were conducted for angular velocity values from 0.7 to 1.3, in which the neural network successfully intercepted the target.

In the case of interception of a rectilinearly moving target, the neural network was trained at the target velocity values $v_x = v_y = 0.5$. In Fig. 14 shows the results of network testing with speeds differing by 20%—in the left figure, the target has a speed of $v_x = v_y = 0.4$, and on the right $v_x = v_y = 0.6$.

It follows from the results obtained that the network generalizes the solution well. This can be useful for applied tasks, since in them the parameters are often known with some error.

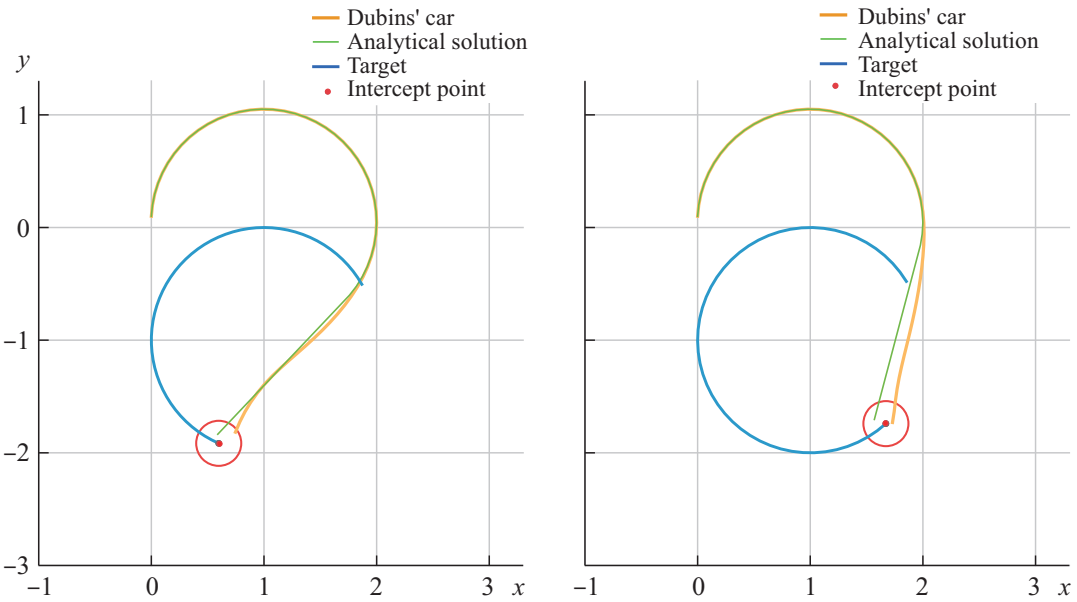

Fig. 13. Comparison of circular intercept trajectories at different initial parameters.

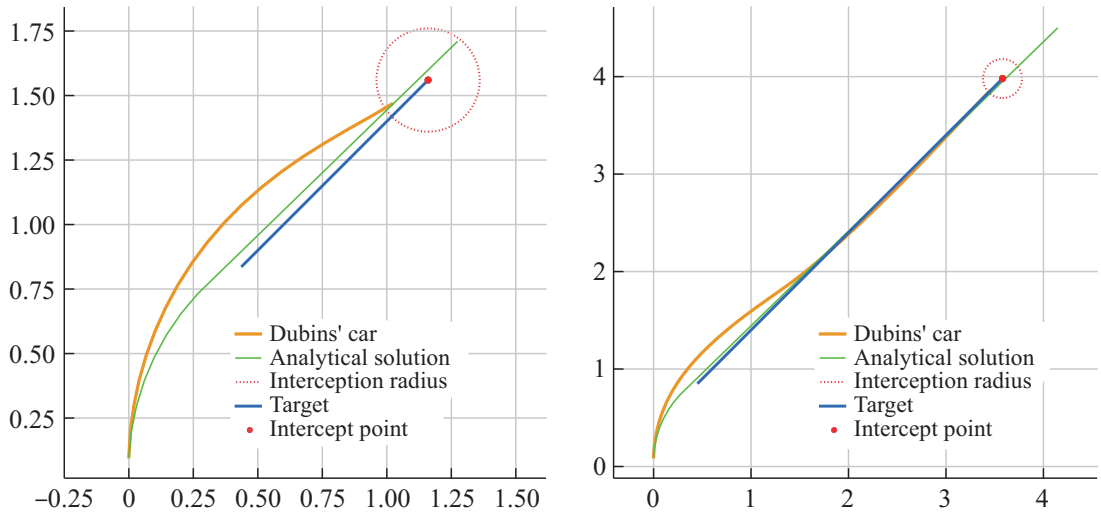

Fig. 14. Comparison of target intercept trajectories at different initial parameters.

6. CONCLUSION

The paper proposed two DDPG-based neural network algorithms for the synthesis of trajectories of interception by the Dubins' car of targets moving along rectilinear and circular trajectories. The features of the proposed algorithms are their ability to work with the space of continuous actions, the guarantee of learning and working with different relative initial positions of goals and the Dubins' car. It is shown that the network successfully generalizes the solution and in some situations offers the fastest solution to the interception problem.

The undoubted advantages of the proposed algorithms can be used, and the algorithms themselves are modified to obtain a barrier surface in the differential game of two cars.

FUNDING

The work was supported by a grant from the ICS RAS Youth Scientific School "Methods of optimization and motion planning of controlled objects". The work of A.A. Galyaev and I.A. Nasonov was partially supported by the Russian Scientific Foundation (project no. 23-19-00134).

REFERENCES

1. Isaacs, R., *Differential Games*, New York: John Wiley and Sons, 1965.
2. Markov, A.A., A Few Examples of Solving Special Problems on the Largest and Smallest Values, *The Communications of the Kharkov Mathematical Society*, 1889, Ser. 2, vol. 1, pp. 250–276.
3. Dubins, L.E., On Curves of Minimal Length with a Constraint on Average Curvature and with Prescribed Initial and Terminal Positions and Tangents, *Amer. J. Math.*, 1957, no. 79, pp. 497–516.
4. Galyaev, A.A. and Buzikov, M.E., Time-Optimal Interception of a Moving Target by a Dubins Car, *Autom. Remote Control*, 2021, vol. 82, pp. 745–758.
5. Glizer, V.Y. and Shinar, J., On the Structure of a Class of Time-Optimal Trajectories, *Optim. Control Appl. Method*, 1993, vol. 14, no. 4, pp. 271–279.
6. Berdyshev, Yu.I., A Problem of the Sequential Approach of a Nonlinear Object to Two Moving Points, *Tr. Inst. Mat. Mekh. Ural. Otd. Ross. Akad. Nauk*, 2005, vol. 11, no. 1, pp. 43–52.
7. Xing, Z., Algorithm for Path Planning of Curvature-constrained UAVs Performing Surveillance of Multiple Ground Targets, *Chin. J. Aeronaut.*, 2014, vol. 27, no. 3, pp. 622–633.
8. Ny, J.L., Feron, E., and Frazzoli, E., On the Dubins Traveling Salesman Problem, *IEEE Transactions on Automatic Control*, 2014, vol. 57, pp. 265–270.
9. Yang, D., Li, D., and Sun, H., 2D Dubins Path in Environments with Obstacle, *Math. Problem. Engineer.*, 2013, vol. 2013, pp. 1–6.
10. Manyam, S.G. et al., Optimal Dubins Paths to Intercept a Moving Target on a Circle, *Proceedings of the American Control Conference*, 2019, vol. 2019-July, pp. 828–834.
11. Caruana, R. and Niculescu-Mizil, A., An Empirical Comparison of Supervised Learning Algorithms, *ICML Proceedings of the 23rd International Conference on Machine Learning*, June 2006, pp. 161–168.
12. Arulkumaran, K., Deisenroth, M.P., Brundage, M., and Bharath, A.A., Deep Reinforcement Learning: A Brief Survey, *IEEE Signal Processing Magazine*, 2017, vol. 34, no. 6, pp. 26–38.
13. Perot, E., Jaritz, M., Toromanoff, M., and de Charette, R., End-to-End Driving in a Realistic Racing Game with Deep Reinforcement Learning, *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 474–475.
14. Al-Talabi, A.A. and Schwartz, H.M., Kalman Fuzzy Actor-Critic Learning Automaton Algorithm for the Pursuit-Evasion Differential Game, *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2016, pp. 1015–1022.
15. Hartmann, G., Shiller, Z., and Azaria, A., Deep Reinforcement Learning for Time Optimal Velocity Control using Prior Knowledge, *IEEE 31st International Conference on Tools with Artificial Intelligence*, 2019, pp. 186–193.
16. Helvig, C.S., Gabriel Robins, and Alex Zelikovsky, The Moving-Target Traveling Salesman Problem, *J. Algorithm*, 2003, vol. 49, no. 1, pp. 153–174.
17. Mnih, V., Kavukcuoglu, K., Silver, D., et al., Human-Level Control Through Deep Reinforcement Learning, *Nature*, 2015, vol. 518, pp. 529–533.
18. Uhlenbeck, G.E. and Ornstein, L.S., On the Theory of the Brownian Motion, *Physic. Rev.*, 1930, vol. 36, pp. 823–841.

19. Hinton, G.E., Srivastava, N., Krizhevsky, A., et al., Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors. arXiv. 2012.

20. Klambauer, G., Unterthiner, T., Mayr, A., et al., Self-Normalizing Neural Networks, *Advances in Neural Information Processing Systems*, 2017, pp. 972–981.

21. Buzikov, M.E. and Galyaev, A.A., Minimum-Time Lateral Interception of a Moving Target by a Dubins Car, *Automatica*, 2022, vol. 135, 109968.

This paper was recommended for publication by O.P. Kuznetsov, a member of the Editorial Board