An Application of Continuous Deep Reinforcement Learning Approach to Pursuit-Evasion Differential Game

Maolin Wang, Lixin Wang, Ting Yue
School of Aeronautic Science and Engineering
Beihang University
Beijing, China
e-mail: wangmaolin@buaa.edu.cn

***Abstract*—Pursuit-evasion differential game is a classic decision-making process in continuous domain. Most recently, the reinforcement learning (RL) technique has greatly advanced the research in decision-making field. In this paper, the dynamic model of the game is described and the optimization problem of the purser in the game is addressed. To learn the control strategy with self-learning, reinforcement learning is considered. An actor-critic based, model-free, end-to-end approach Deep Deterministic Policy Gradient (DDPG) Algorithm is applied to train the pursuer. In the first training phase the pursuer is trained only with a given evader's control strategy. In the second training phase, the pursuer and evader are trained simultaneously without any expert knowledge given in advance. The result shows that the pursuer and the evader can learn the control strategy during the training phase.**

Keywords-Differential Game; Reinforcement Learning; DDPG; Self-Learning

I. INTRODUCTION

Differential games are a group of problems related to the modeling and analysis of conflict in the context of a dynamic system. Pursuit-evasion scenario, as a classic differential game which was firstly introduced by Isaacs[1], has been extensively studied in the field of unmanned aerial vehicle (UAV) and unmanned aerial combat. It is concerned with finding the control strategies of two different characters, the pursuer and the evader. The main object of the pursuer is to track and catch the evader in the minimum time under the constraint of the dynamic equations. Since Isaacs, a large amount of research has been published to solve the pursuit-evasion differential game in various methods[2]. In [3], a control strategy for the pursuer was presented when the evader behaved intelligently. In [4], a reachability-based approach was adopted to deal with the pursuit-evasion differential game. In [5], a resistance model with multi-parameter in the three-dimensional space by studying on the model of pursuit-evasion differential game is constituted. Based on the researching of artificial fish school algorithm, an intelligent optimum algorithm of differential game on the basis of artificial fish school algorithm is proposed. In [6], the differential game was considered as a Markov decision process (MDP) and a novel method CTMDP to address the uncertainties in pursuit-evasion problem.

Recent years, reinforcement Learning (RL) develops rapidly and has advanced the research in decision-making area. As a computational approach to understanding and automating goal-directed learning and decision-making, RL is a model-free machine learning approach and is preferable to be applied in cases without expert knowledge. Different from supervised learning, which learns from samples provided by a knowledgeable external supervisor, reinforcement learning is used for learning from interaction. There are several research applied RL to pursuit-evasion differential game. In [7], RL was applied to train the individual agents in a multi-agent pursuit-evasion game to learn a particular pursuit strategy. In [8], Q(λ)-learning was combined with the fuzzy inference system (FIS) and two techniques were proposed. RL was used to tune the parameters of FISs.

However, the applicability of RL agents has long been previously limited to domains in which useful features can be handcrafted, or to domains with fully observed, low-dimensional state spaces. Combined with advances in training deep neural networks, Deep Reinforcement Learning (DRL) has developed rapidly in recent years[7]-[11]. Deep Q-Learning Network (DQN) was developed and achieved success in the field of video game and Go game. In [12], an actor-critic based, model-free, end-to-end algorithm named Deep Deterministic Policy Gradient (DDPG) was proposed to solve the continuous reinforcement learning problem. DDPG was based on the Deterministic Policy Gradient (DPG) algorithm proposed in [13].

In this paper, it is assumed that the pursuer and evader does not know its control strategy during a pursuit-evasion differential game. The learning goal is to make the agent able to self-learn the optimal control strategy without any knowledge given in advance. To solve this continuous optimization problem, DDPG algorithm is applied which enables the agent to learn the strategy by interacting with the environment.

The paper is organized as follows: the model of pursuit-evasion differential game is built in section II. Some basic

978-1-5386-6243-4/19/$31.00 ©2019 IEEE

knowledge of reinforcement learning is briefly introduced in section III. Then the continuous deep reinforcement learning approach is applied to train the pursuer in one-verse-one pursuit-evasion differential game in section IV. The simulation result is shown in section V which contains two phases. In phase I, the evader obtains the optimal strategy proposed in [3]. In phase II, the pursuer and the evader are trained simultaneously with two individual networks. The conclusion and the future work are discussed in section VI.

II. GAME FORMULATION

The pursuit-evasion game is a classical application of differential game as shown in Fig. 1. The goal of the pursuer is to catch the evader in the shortest possible time. The goal of the evader is to avoid the meeting of the players' trajectories on a whole semi-infinite interval of time or if it is possible to maximally postpone the moment of meeting[14]. In equation type, it can be described as:

$$\min T_{capture} = f(V_p, \delta_{up}, V_e, \delta_{ue}, L) \tag{1}$$

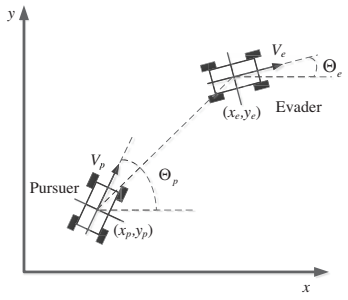

Figure 1. The model of pursuit-evader differential game

where the velocity of pursuer and evader is denoted as V_p and V_e, the steer angle of pursuer and evader is denoted as δ_{up} and δ_{ue} and L is the distance between the pursuer and evader.

In Fig.1, the (x,y) pairs is the position of the pursuer and evader. The angle Θ indicates the orientation angle of two cars. The velocity of the two cars are denoted as V in Fig. 1 which are considered as constant in this game. The dynamic equations of the game are demonstrated below[8]:

$$\begin{cases} V_i = const \\ \dot{x}_i = V_i \sin(\Theta_i) \\ \dot{y}_i = V_i \cos(\Theta_i) \quad i = p,e; \\ \dot{\Theta}_i = \dfrac{V_i}{L_i}\tan(\delta_{ui}) \end{cases} \tag{2}$$

where L is the distance between the front and rear axle and u is the steering angle which satisfy the range that $\delta_{ui} \in [-\delta_{u\max}, \delta_{u\max}]$. From Equation (2) it can be seen that for each car a minimum turning radius exists which can be calculated as:

$$R_{\min} = \frac{L_i}{\tan(\delta_{u\max})} \tag{3}$$

To make the evader more maneuverable than the pursuer, the value of max steering angle satisfied that $u_{e\max} > u_{p\max}$ while the value of max velocity $V_p > V_e$. The optimal strategy of pursuit-evasion model was proposed in [3] that:

$$\delta_{ui} = \begin{cases} -\delta_{ui\max} & \delta_i < -\delta_{ui\max} \\ \delta_i & -\delta_{ui\max} \le \delta_i \le \delta_{ui\max} \\ \delta_{ui\max} & \delta_i > \delta_{ui\max} \end{cases} \quad i = p,e \tag{4}$$

where $\delta_i = \arctan((y_e-y_p)/(x_e-x_p)) - \theta_i$. The capture is considered to occur when the distance between the pursuer and evader l satisfies:

$$l = \sqrt{(x_e - x_p)^2 + (y_e - y_p)^2} \le l_{capture} \tag{5}$$

where $l_{capture}$ is a certain distance. In this simulation, a value of 1.0m is found to be suitable for l_{capture}. Given the optimal analytical solution as a reference, the simulation result then can be evaluated.

III. ALGORTITHM

A. Reinforcement Learning

Reinforcement learning is a primary approach to learn control strategies by considering what actions autonomous agents should take to maximize a numerical reward signal[15]. Agent-environment interaction in RL is shown in Fig.2

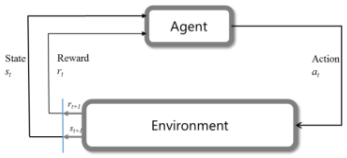

Figure 2. Agent-environment interaction in RL

It is assumed that future rewards are discounted by a factor of γ per time-step, and the future discounted return at time t is defined as:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \tag{6}$$

where $0 < \gamma \le 1$ and T is the time-step at which the episode terminates. The action-value function $Q(s,a)$ is the expected return achieved by taking an action, a, in a states, s, and is defined as:

$$Q(s,a) = E_\pi \left(R_t \mid s_t = s, a_t = a, \pi \right) \tag{7}$$

where π is a policy mapping sequences to actions. The optimal action-value function $Q^*(s,a)$ is defined as the maximum expected value:

$$Q^*(s,a) = \max_\pi Q \tag{8}$$

and the optimal action a^* in state s is defined as:

$$a^* = \arg\max_{a'} Q(s,a') \tag{9}$$

The optimal action-value function obeys an important iteration equation named Bellman equation[16]. The optimal strategy is to select the action maximizing the expect value of $r + \gamma Q^*(s',a')$:

$$Q^*(s,a) = E\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a \right] \tag{10}$$

where a' is the successor action of the successor sequence s'.

The basic idea of RL is to use Bellman equation as an iterate update to estimate the action-value function. Such value iteration algorithm will converge to the optimal action-value function. The representative Q-learning algorithm has been widely applied to low-dimensional discrete RL problems.

However, for high-dimensional complex problem, this basic approach is impractical because it may lead to 'curse of dimensionality'. Instead, the function approximation is applied to estimate the action-value function. In DRL, a deep neural network with weights θ is used as a non-linear function approximator[9]. The loss function $L_i(\theta_i)$ is defined as:

$$L_i(\theta_i) = E_{s,a\sim\rho}\left[(y_i - Q(s,a;\theta_i)^2 \right] \tag{11}$$

where $y_i = E\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) \mid s,a \right]$ is the target for iteration i and $\rho(s,a)$ is a probability distribution over sequences s and action a, which is referred to as the behavior distribution.

B. Deep Deterministic Policy Gradient Algorithm

Actor-critic methods are Temporal-Difference methods that have a separate memory structure to explicitly represent the policy independent of the value function[15]. The structure used to select actions is named actor while critic is applied to estimate the value function. The architecture of the actor-critic is shown in Fig. 3.

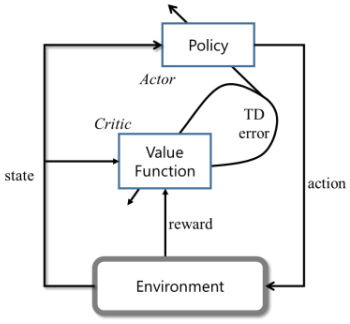

Figure 3. The actor-critic architecture

It is not possible to apply Q-learning to continuous action spaces, therefore, to solve the complex RL problem with continuous actions, the deterministic gradient policy algorithm was proposed[13]. It is demonstrated that deterministic policy gradient algorithms (DPG) can significantly outperform their stochastic counterparts in high-dimensional action spaces. Under the success of the Deep Q-Learning and DPG, Deep Deterministic Policy Gradient algorithm (DDPG) was proposed in [12]. An actor-critic, model-free algorithm based on the deterministic policy gradient that can operate over continuous action spaces was presented. For many tasks the algorithm can learn policies end-to-end directly.

The agent's goal is to obtain a policy which maximizes the cumulative discounted reward from the start state, denoted by the performance objective as an expectation:

$$J_\pi = \mathrm{E}\left[r_1^\gamma \mid \pi \right] \tag{12}$$

It is proved in [13] that the deterministic policy gradient exists:

$$\begin{aligned} \nabla_\theta J(\mu_\theta) &= \int_s \rho^\mu(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a) \mid_{a=\mu_\theta(s)} \mathrm{d}s \\ &= \mathrm{E}_{s\sim\rho^\mu}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a) \right] \end{aligned} \tag{13}$$

in the condition that $\nabla_\theta \mu_\theta(s)$ and $\nabla_a Q^\mu(s,a)$ exists where θ^Q and θ^μ are parameters of critic and actor, respectively.

IV. Solving Pursuit-Evasion Problem Via DDPG

In this section, the pursuit-evasion differential problem is considered as a Markov Decision Process(MDP) and DDPG algorithm is applied to solve the control problem of the pursuer.

A. MDP Modeling

MDP describes such a process where an agent at some state takes an action and transits to the next state with an one-step cost. It consists four components: a state space $\mathbf{S}$, an action space $\mathbf{A}$, an one step cost function $r(\boldsymbol{s},\boldsymbol{a})$:$\mathbf{S}\times\mathbf{A}\rightarrow\mathbf{R}$ and a stationary one-step transition probability $p(s_t|s_1,a_1,\ldots,s_{t-1},a_{t-1})$. The Markov property means that current state is only dependent on the last state and action:

$$p(s_t \mid s_1,a_1,\ldots,s_{t-1},a_{t-1}) = p(s_t \mid s_{t-1},a_{t-1}) \tag{14}$$

In pursuit-evasion problem, the state s_t can be expressed as:

$$s_t = \left[x_p, y_p, \Theta_p, x_e, y_e, \Theta_e \right]^T \tag{15}$$

where x, y and Θ denotes to the position and heading angle defined in Equation(2). The action a_t can be expressed as:

$$a_t = \delta_u \tag{16}$$

where δ_u denotes to the steering angle. The reward function r_{t+1} is expressed as:

$$\begin{aligned} r_{t+1} &= \frac{D(t) - D(t+1)}{(V_p + V_e)T} \\ D(t) &= \sqrt{\left(x_e(t) - x_p(t) \right)^2 + \left(x_e(t) - x_p(t) \right)^2} \end{aligned} \tag{17}$$

where T is the sampling time. The reward function r_{t+1} indicates whether the pursuer is moving towards the evader.

B. DDPG Structure

The Structure of DDPG is based on the description in Section III.B, and the overall roadmap in each step is shown in Fig. 5. Especially, for the problem in which the value of output has an up/down limit, a constraint is required. Inspired by the approach in neural network, a hyperbolic tangent function tanh is applied as an activation function in the output layer to limit the value of action in the range of [-1, 1].

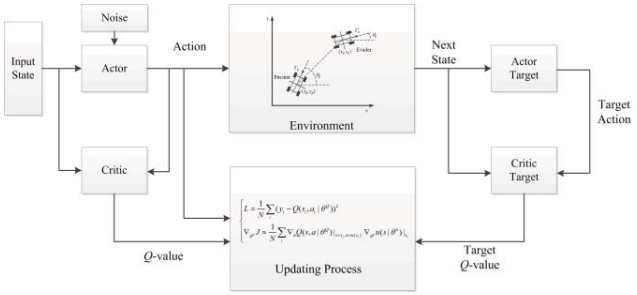

Figure 4. Overall Structure of DDPG

Reinforcement learning is known to be unstable or even to diverge when a nonlinear function approximator is used to represent the action-value function[17]. In [9], the use of target network and replay buffer is proposed. The differences between target networks and actor-critic networks is shown in Table 1.The overall view of DDPG algorithm applied to the pursuit-evasion differential game is shown in Algorithm 1.

Algorithm 1 DDPG Applied to the Pursuit-evasion Differential Game

Input:

Discount factor γ, Number of episodes M, number of steps for each episode T, batch size N, learning rate for actor networks and critic networks r_a and r_c.

1: Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor network $u(s|\theta^\mu)$.

2: Initialize target network Q' and μ' with parameters $\theta^Q=\theta^{Q'}$, $\theta^\mu=\theta^{\mu'}$.

3: Initialize replay buffer R.

4: Initialize the best policy θ^{μ^*}.

4: for episode = 1 to M do

5: Initialize a random $a_t=u(s_t|\theta^\mu)+N_t$ according to the current policy and exploration noise.

6: Receive initial observation state s_1.

7: for t=1 to T do

8: Execute action a_t and observe reward r_t and observe new state s_{t+1}.

9: Store transition (s_t,a_t,r_t,s_{t+1}) to R.

10: Sample a random minibatch of N transitions (s_t,a_t,r_t,s_{t+1}) from R.

11: Set $y_i=r_i+\gamma Q'\left(s_{i+1},u'(s_{i+1}|\theta^{u'})|\theta^{Q'}\right)$

12: Update critic by minimizing the loss: $L=\frac{1}{N}\sum_i\left(y_i-Q(s_i,a_i|\theta^Q)\right)^2$.

13: Update the actor policy using the sampled policy gradient:

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

14: Update the target networks:

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

15: end for

16: Evaluate the policy and update the best policy θ^{μ^*}.

17: end for

Output:

Control strategy for pursuer $\delta_{up}=\mu(s|\theta^\mu)$.

TABLE I. DIFFERENCES BETWEEN TARGET NETWORKS AND ACTOR-CRITIC NETWORKS

Network	Formula	Input	Output
critic target	$Q'\left(s_{t+1},u'(s_{t+1}|\theta^{u'})|\theta^{Q'}\right)$	next state; output of actor target network	Q' value used to calculate y_i
critic	$Q(s_t,a|\theta^Q)$	current state; current action	Q value used to calculate TD-error and update the actor network
actor target	$u'(s_{t+1}|\theta^{u'})$	next state	action u' used as input of target critic network
actor	$u(s_t|\theta^u)$	current state	action u used to update the actor network

The actor policy is updated every step in each episode and an evaluation is required after each episode so that the best policy can be stored. Considering the convergence of the algorithm, the best policy may not be the newest policy. To get the best policy of the entire finished episode, a random performance comparison is applied. Algorithm 2 shows the procedure used to determine the best policy.

The performance index J in Algorithm 2 is designed based on the game. For pursuit-evader differential game, it is set to be the rest time, that is, the total simulation time minus the capture time. The best policy updating procedure guarantees that the output actor policy of the DDPG will always be the one which get the best performance among all the episodes.

Algorithm 2 Policy Evaluation

1: Input:

current_policy, best_policy

2: for n = 1: n_games do:

3: Initial a random state *s_initial.*

4: Finish the game independently using current_policy and best_policy, respectively.

5: end for

6: Observe the performance index of two policies, *J_current* and *J_best.*

7: if *J_current*> *J_best* THEN

8: best_policy = current_policy

9: end if

10: Output: best_policy θ^{μ^*}.

V. SIMULATION RESULTS

A. Simulation Environment

The training is conducted on Pytorch platform using CPU Intel Xeon E5-2620 with a 2.10GHz clock frequency and 32 Gigabytes of RAM. Adaptive moment estimation (ADAM) algorithm is applied in gradient decent to accelerate the convergence[18]. Actor networks have two fully-connected hidden layers with 64 and 128 units which are activated with Relu function. In order to bound the actions, the final output layer of the actor is a tanh layer. The critic networks have three fully-connected hidden layers with 64, 128 and 64 units activated with Relu function. The actions are not included until the second hidden layer. And L_2 weight decay of 10^{-3} is

applied to the critic networks. The other hyperparameters involved in this simulation is list in Table II.

TABLE II. VALUES OF SIMULATION HYPERPARAMETERS

Parameters	Value	Description
γ	0.9	Discount factor used in the reinforcement update.
τ	0.01	Soft update factor used in target Network update.
Minibatch size	32	Number of training cases over which each ADAM update is computed.
Replay memory size	30000	ADAM updates are sampled from this number of most recent steps.
Episode size	1000	Number of Episodes use in FDPG.
Step size	200	Number of steps used in each episode.
Sampling time	0.1	Size of T in each step.
Learning rate of critic	0.001	The learning rate use in ADAM to update the critic.
Learning rate of actor	0.001	The learning rate use in ADAM to update the actor.
β_1	0.9	Exponential decay hyperparameter used in ADAM.
β_2	0.999	Exponential decay hyperparameter used in ADAM.
ε	1e-8	Stability constant used in ADAM.

The initial condition of the training are listed in Table III An episode is terminal on the condition that the pursuer catches the evader or the sampling steps reached 200 and the training starts when the replay buffer is full.

TABLE III. INITIAL CONDITION OF THE TRAINING

Parameters	Value
Initial position of pursuer	(0,0)
Initial orientation of pursuer θ_p	0
Constant velocity of pursuer V_P	1m/s
distance between the front and rear axle L_P	0.3m
The steering angle of pursuer u_p	[-0.5,0.5]
Initial position of evader	a random position
Initial orientation of evader θ_e	0
Constant velocity of evader V_e	0.5m/s
distance between the front and rear axle L_e	0.3m
The steering angle of evader u_e	[-1,1]
Capture radius	0.10m

B. *Training Phase I*

In the training phase I, the control strategy of the evader obeys the optimal equation defined in Equation(4) and the DDPG applies to the pursuer only. Figure 5 shows the total reward during the training process. Training starts at Episode 150 when replay buffer reaches full. The total reward increases to positive and maintains stable after Episode 300. Figure 6 shows the paths of pursuer and evader after 300 episodes. The purser starts at point (0,0) and the evader starts at point (7,-2). It can be seen that the pursuer fails to catch the evader and the path is unsmooth. Figure 7 shows the paths of pursuer and evader with the best policy after 1000 episodes. The initial positions of the pursuer and the evader are the same as that in Figure 6. The path is smooth enough and the pursuer is able to catch the evader directly. The changes in the paths shows that the pursuer learns the control policy with training episodes increasing.

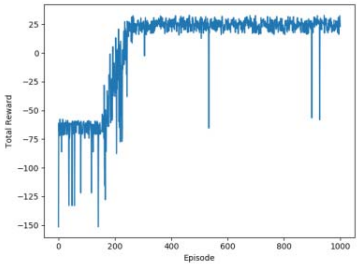

Figure 5. Total Reward Function of the Pursuer in Training

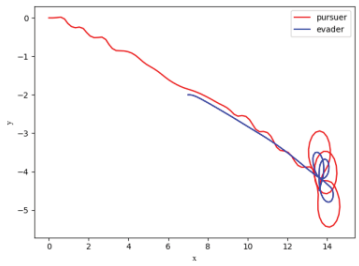

Figure 6. Paths of Pursuer and Evader after 300 Episodes

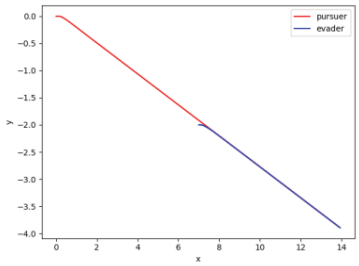

Figure 7. Paths of Pursuer and Evader with the Best Policy

C. *Traning Phase II*

In the training phase II, the pursuer and evader are trained simultaneously with two individual networks. Pursuit-evasion differential game is a zero-sum game so the reward of the purser and evader is opposite.

Figure 8 shows the total reward of the pursuer and evader during the training process. It can be seen that after about 200 episodes, the total reward of the both agents converges. The pursuer has an advantage over the evader. Figure 9 shows the paths of the pursuer and evader in a single game with a time period of 14.7s. The pursuer starts at point (0,0) and the evader starts at point(6,-5). When game starts, the evader tries to flee from the pursuer but the pursuer moves directly towards the evader. As the distance decreases, the evader tries to turn a circle in order to take advantage of its better maneuverability. The pursuer finally captures the

evader with a superiority in velocity. It can be seen that the pursuer has learned how to catch the evader and evader has learned how to escape from the capture, simultaneously.

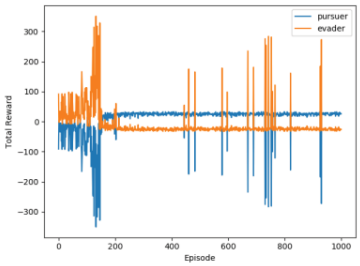

Figure 8. Total Reward Function of the Pursuer and Evader in Training

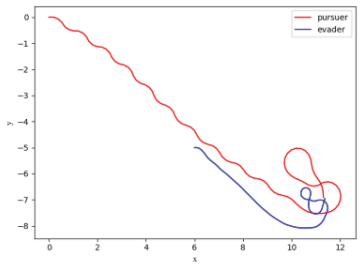

Figure 9. Paths of Pursuer and Evader

VI. CONCULUSION AND FUTURE WORK

In this paper, the continuous deep reinforcement learning algorithm DDPG is applied to a 1-vs-1 pursuit-evasion differential game. The training has two phases, phase I trains the pursuer only and phase II trains the pursuer and evader the simultaneously without any expert knowledge given in advance. From the training phase I, it can be concluded that the pursuer is able to obtain the control strategy in self-learning process. The result of training phase II shows that when applying DDPG algorithm to a zero-sum game, the both agents can learn a control strategy during interacting with the dynamic environment.

In future work, the complexity of the dynamic model can be updated to solve a more practical decision-making problem such as three-dimensional air combat problem.

REFERENCES

[1] Rufus Isaacs, *Differential Games*, Dover, 1999.

[2] O.Hajek, *Mathematics in Science and Engieering Volume 120: Pursuit Games*, Academic Press, Inc, New York, 1975.

[3] Shen Hin Lim, Furukawa, T., Dissanayake, G., Durrant-Whyte, H.F, "A Time-Optimal Control Strategy for Pursuit-Evasion Games Problems," Processing of the 2004 IEEE International Conference on Robotics and Automation, Vol.4, IEEE, New Orleans, LA, 2004, pp. 3962-3967.

[4] W. Sun, P. Tsiotras, T. Lolla, D. N. Subramani and P. F. J. Lermusiaux, "Pursuit-evasion games in dynamic flow fields via reachability set analysis," 2017 American Control Conference (ACC), Seattle, WA, 2017, pp. 4595-4600.

[5] Shi Hong-yan, Shang Zhi-qiang, "Study on a solution of pursuit-evasion differential game based on artificial fish school algorithm," 2010 Chinese Control and Decision Conference, Xuzhou, 2010, pp. 2092-2096.

[6] S. Jia, X. Wang and L. Shen, "A Continuous-Time Markov Decision Process-Based Method With Application in a Pursuit-Evasion Example," in IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 46, no. 9, pp. 1215-1225, Sept. 2016.

[7] Y. Ishiwaka, T. Satob, Y. Kakazu, "An approach to the pursuit problem on a heterogeneous multiagent system using reinforcenment learning," Robotics and Autonomous System, Vol 59, No. 1, 2010, pp. 3-30.

[8] Desouky S F, Schwartz H M, "Self-learning Fuzzy Logic Controllers for Pursuit–Evasion Differential Games," Robotics and Autonomous Systems, Vol.59, No. 1, 2011, pp. 22-33.

[9] Mnih V, Kavukcuoglu K, Silver D, et al, "Human-level Control Through Deep Reinforcement Learning," Nature, Vol. 518, 26 Feb. 2015, pp. 529-533.

[10] Silver, David, et al, "Mastering the Game of Go with Deep Neural Networks and Tree Search," Nature, Vol. 529, 28 Jan. 2016, pp. 484-489.

[11] Silver D, Schrittwieser J, Simonyan K, et al, "Mastering the Game of Go without Human Knowledge," Nature, Vol. 550, 19 Oct. 2017, pp. 354-359.

[12] Lillicrap T P, Hunt J J, Pritzel A, et al, "Continuous Control with Deep Reinforcement Learning," arXiv preprint, arXiv: 1509.02971, 2015.

[13] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, et al, "Deterministic Policy Gradient Algorithms," ICML, Jun 2014, Beijing, China. <hal-00938992>

[14] Liubarshchuk. Ievgen, Althöfer. Ingo, "The Problem of Approach in Differential-Difference Games," Int J Game Theory, Vol.45, No.3, 2016, pp. 511-523.

[15] R.S. Sutton, A.G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998.

[16] Bellman, R., "Dynamic Programming and Lagrange Multipliers," Proceedings of the National Academy of Sciences, Vol.42, No. 10, 1956, pp. 767–769.

[17] Tsitsiklis, J. & Roy, B. V, "An Analysis of Temporal-Difference Learning with Function Approximation," IEEE Trans. Automat. Contr, Vol. 42, No.5, 1997, pp. 674–690.

[18] Shen Hin Lim, Furukawa, T., Dissanayake, G., Durrant-Whyte, H.F, "A Time-Optimal Control Strategy for Pursuit-Evasion Games Problems," Processing of the 2004 IEEE International Conference on Robotics and Automation, Vol.4, IEEE, New Orleans, LA, 2004, pp. 3962-3967.