Proceedings of The 2020 IEEE International
Conference on Real-time Computing and Robotics
September 28-29, 2020

A Deep Reinforcement Learning Approach for the Pursuit Evasion Game in the Presence of Obstacles

Qi Qi, Xuebo Zhang and Xian Guo

Abstract— **In a pursuit-evasion game, the pursuer tries to capture the evader, while the evader actively avoids being captured. Traditional approaches usually ignore or simplify kinematic constraints by using a grip world discrete model and they assume that the game is played in free space without obstacles. In this paper, a curriculum deep reinforcement learning approach is proposed for the pursuit-evasion game, which considers the kinematics of mobile robot in practical applications and the influence of static obstacles in the environment. To improve the system performance, we use the mechanism of self-play to train the pursuer and the evader at the same time. In addition, the method of curriculum learning is used, making the agent learn simpler tasks before learning more complicated ones. Comparative simulation results show that the proposed approach presents superior performance for both pursuer and evader when playing against intelligent opponents.**

I. INTRODUCTION

The pursuit-evasion game is a type of differential games which was first proposed in [1]. In a pursuit-evasion game, the players have conflicting objectives: the pursuer attempts to capture the evader in minimal time, in contrast, the evader tries not to be caught for as long as possible. Because of its extensive applications in the real world, such as search and rescue missions and computer games, many research efforts have been focused on pursuit-evasion game. The works [2], [3] reviewed recent results in the area of pursuit-evasion and probabilistic search. In this paper, we consider the pursuit-evasion game in the environment with static obstacles. At the same time, in order to be applied to real-world systems, the players in the game are mobile robots whose kinematics need to meet the non-holonomic constraints, velocity and acceleration constraints [4]–[6]. Such realistic considerations for both kinematics and environments make the pursuit-evasion problem more challenging and further exploration is needed.

In recent years, the combination of reinforcement learning and neural networks has achieved impressive results in virtual games [7]–[11]. In reinforcement learning, the agent can learn complex behaviors through interactions with an unknown environment without any human knowledge. The agent receives rewards from the environment, and uses the existing knowledge to improve its future performance, thus learning the optimal or suboptimal strategy. Therefore, reinforcement learning presents several remarkable advantages including adaptability to different environments and different strategies of opponents.

A number of methods based on reinforcement learning have been proposed to study the pursuit-evasion game [12]–[15]. In [12], a fuzzy logic controller combined with a reinforcement learning algorithm is designed in a "guarding a territory" game. Multi-agent deep Q-network (MADQN) is proposed in [13] to train the pursuer, while the evader follows a heuristic strategy, that is, moving towards the direction furthest from the pursuer. In [13], the global state of the system was represented as a image-like tensor, and the action space was discrete. [14] proposed an asymmetric dueling mechanism for visual active tracking (AD-VAT). In AD-VAT, the tracker and target were regarded as pursuer and evader respectively, and were improved synchronously during the training process.

To the best of our knowledge, most of the research on pursuit-evasion game based on reinforcement learning assumes that players move at a constant speed or on a grid map (i.e. move up, down, left and right). In this paper, we consider the kinematics of the robot in practical application, where the linear velocity of the robot is continuous. In addition, the influence of obstacles on the robot's motion is also considered. During the training process, the self-play mechanism, which has been used successfully in AlphaGo [16], Dota 2 [8], [17], is applied in the pursuit-evasion scenario in this paper. In addition, we combine a curriculum learning process to provide a pursuer with a certain level of intelligence. Comparative simulation results show that the proposed approach presents superior performance for both pursuer and evader when playing against intelligent opponents.

The main contributions of this paper are: 1) We consider the impact of obstacles on robot's motion, as well as the non-holonomic constraints of the robot in practical applications. 2) A self-play mechanism is applied to train the both players in pursuit-evasion game simultaneously. 3) Curriculum learning is used to train the agent for simple tasks before learning more difficult ones.

This paper is organized as follows. Section II introduces the pursuit-evasion scenario and the constitution of DQN, and gives an overview of the proposed approach. Section III describes the learning setup. Training process is introduced in Section IV. Simulation results are carried out and comparison results are shown in Section V. Section VI gives some

*This paper is supported in part by the National Natural Science Foundation of China under grant U1613210, Tianjin Science Fund for Distinguished Young Scholars under Grant 19JCJQJC62100, the Fundamental Research Funds for the Central Universities, and Natural Science Foundation of Tianjin under Grant 17JCYBJC40600 and 19JCQNJC03200.

Q. Qi, X. Zhang (corresponding author) and X. Guo are with the Institute of Robotics and Automatic Information System (IRAIS), Tianjin Key Laboratory of Intelligent Robotics (TJKLIR), Nankai University, Tianjin 300071, China. nkqiqi@mail.nankai.edu.cn, zhangxuebo@nankai.edu.cn, guoxian@nankai.edu.cn

978-1-7281-7293-4/20/$31.00 © 2020 IEEE

concluding remarks.

II. Formulation and Overall Framework

In this section, the scenario of the pursuit-evasion game is described first. Then the constitution of DQN is introduced, and the overall framework of the proposed algorithm is explained in details.

A. Problem Definition

The pursuit-evasion game scenario is described as follows. There is a pursuer and an evader in the environment, which have opposite objectives. The evader needs to avoid being captured, and the pursuer wants to capture the evader. There are also static circular obstacles in the environment. Each agent must avoid the obstacles while completing its main task. The pursuit-evasion game scenario is shown in Fig. 1.

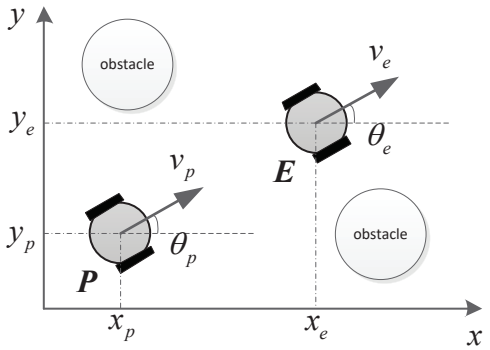


Fig. 1. Pursuit-evasion game scenario. P is the pursuer, E is the evader. (x_p, y_p) is the position of the pursuer, v_p is the linear velocity of the pursuer, θ_p refers to the direction angle of the pursuer. (x_e, y_e) is the position of the evader, v_e is the linear velocity of the evader, θ_e refers to the direction angle of the evader.

Both players in this paper are mobile robots with non-holonomic constraints [19]. The kinematics model of the robot is described as follows.

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{v} &= u_1 \\ \dot{\theta} &= u_2 \end{aligned} \tag{1}$$

where x, y represents the position of the robot, and $v\epsilon[v_{min}, v_{max}]$ is the linear velocity. u_1 and u_2 are the control input of the system, representing linear acceleration and angular velocity respectively, $u_1\epsilon[-u_{1_{max}}, u_{1_{max}}]$, $u_1\epsilon[-u_{2_{max}}, u_{2_{max}}]$. The performance parameters of the pursuer and the evader are the same, which is fair to both sides.

In addition, the two robots cannot exceed the boundary during the movement, which defines the following constraints.

$$\begin{aligned} x_l \le x_p \le x_u \\ y_l \le y_p \le y_u \\ x_l \le x_e \le x_u \\ y_l \le y_e \le y_u \end{aligned} \tag{2}$$

$[x_l, y_l, x_u, y_u]$ is the boundary of the environment. Exceeding the boundary is considered as failure as well as hitting an obstacle. When the distance between the two robots is less than a threshold, it is considered that the pursuer successfully catches the evader.

B. Deep Q-network

Deep Q-network (DQN) is one of the classic algorithms in deep reinforcement learning [18]. It uses a neural network in deep learning as a generalization model of the action-value function. At the same time, the classic Q-learning algorithm in reinforcement learning is used to update the parameters of model.

Experience replay is a key technique used in DQN, which breaks the correlations between training samples, and ensures the convergence of value function. The agent interacts with an environment and get a tuple of transition (s_t, a_t, r_t, s_{t+1}). In DQN, the transitions are stored in an experience pool. During each training, the data is randomly sampled from the experience pool. DQN sets a neural network to estimate the current action-value Q, while setting a target network with the same structure and different parameters to estimate the target action-value $\hat{Q}$. The loss function at iteration i is defined as:

$$L(\theta_i) = E[(r + \gamma \max_{a'} \hat{Q}(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2] \tag{3}$$

in which θ_i represents the parameters of the Q-network at iteration i and θ_i^- are the parameters of the target Q-network. Then the gradient descent method is used to continuously update the weights. θ_i are updated every iteration, while the target Q-network parameters θ_i^- are updated every fixed number of iterations with θ_i.

C. Overall framework

In this paper, curriculum learning [21] is integrated with deep Q-network, which is inspired by the learning mechanism of humans, that is, learning simple problems before learning more complex ones. A properly ordered training process could improve training efficiency and achieve better performance.

According to the difficulty of the task, the training process is divided into two steps. In the first step, only the pursuer is trained to reach a fixed point. Since the opponent is stationary, the noise of the samples is smaller, which relatively reduce the difficulty of training.

Then, the pursuer with a certain level of intelligence is used in the next step, where the pursuer and the evader are trained simultaneously during competitive self-play. In the process of competition, the agents always play with the opponent with an appropriate level. When exploring diverse trajectories, both players can reinforce each other, which makes the agents more robust. Fig. 2 shows the overall framework of the proposed approach.

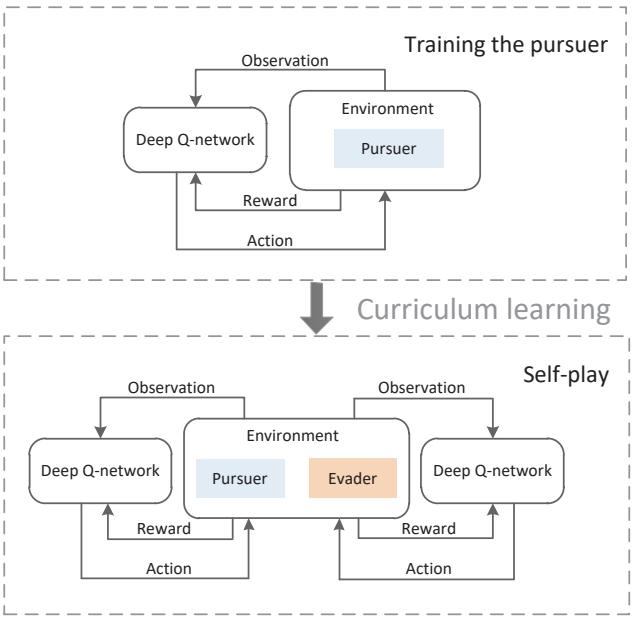


Fig. 2. Overall framework for Deep Q-network.

III. Learning Setup

In this section, the key ingredients of reinforcement learning is introduced, including the action space, network input, reward function and network architecture.

A. Action space

According to the definition of the control input of the kinematics model of the mobile robot, the action space A is defined as follows.

$$A = [u_1, u_2] \tag{4}$$

where the first element u_1 controls the acceleration of the robot, and the second element u_2 controls the robot to turn left or right. Since the action space in DQN is discrete, each dimension is discretized into three gears as follows.

$$\begin{aligned} u_1 &= \{-u_{1_{max}}, 0, u_{1_{max}}\} \\ u_2 &= \{-u_{2_{max}}, 0, u_{2_{max}}\} \end{aligned} \tag{5}$$

where $u_{1_{max}}$ and $u_{2_{max}}$ denote the maximum linear acceleration and angular speed, respectively. Thus there are 9 combinations of actions in total. It is important to emphasize that, since the input is acceleration, the linear velocity of the robot is continuous.

B. Network input

The state of each agent could be expressed as:

$$S_i = [x_i, y_i, v_i, \theta_i] \tag{6}$$

We assume that the states of both sides are observable during competition. S_{self} represents the state information of the agent itself, and S_{opp} represents the state of the opponent. Therefore, when training each agent, the input of the network could be defined as

$$S = [S_{self}, S_{opp}] \tag{7}$$

which is an 8-dimensional vector. Since obstacles and boundaries of the environment need to be considered, the absolute state of the agent is chosen as the network input instead of the relative state.

C. Reward function

The reward function indicates how well an action is taken in a particular state. Since the pursuer and the evader need to be trained simultaneously in this paper, the reward function is designed separately for the two agents. Firstly, the reward function of the pursuer is described below.

$$r_{pursuer} = \begin{cases} 10 & \text{if } d_t < D_{catch} \\ -10 & \text{if collide with an obstacle} \\ -10 & \text{if out of range} \\ c(d_{t-1} - d_t) & \text{otherwise} \end{cases} \tag{8}$$

where d_t indicates the distance between two agents at time t. D_{catch} is the threshold for the pursuer to catch the evader. c is a hyper-parameter.

If d_t is smaller than D_{catch}, a positive reward will be returned to the pursuer. If the pursuer collides with an obstacle or exceeds the boundary, a negative reward would be assigned, then the episode terminates immediately. Otherwise, an immediate reward is defined as the change in distance from the target, $c(d_{t-1} - d_t)$, which makes the pursuer continuously move towards the target.

As for the evader, a negative reward would be returned to punish it for being captured. When hitting an obstacle or moving out of range, the reward setting is the same as that of the pursuer. And the immediate reward is defined as $c(d_t - d_{t-1})$, in order to encourage the evader to run far away from the pursuer. In summary, the reward function of the evader is defined as follows.

$$r_{evader} = \begin{cases} -10 & \text{if } d_t < D_{catch} \\ -10 & \text{if collide with an obstacle} \\ -10 & \text{if out of range} \\ c(d_t - d_{t-1}) & \text{otherwise} \end{cases} \tag{9}$$

D. Network architecture

As presented in Fig. 3, the neural network has four fully-connected hidden layers with 256 units. Each hidden layer uses ReLU as an activation function. The output layer is a fully-connected layer with 9 units, representing state-action value of the corresponding action.

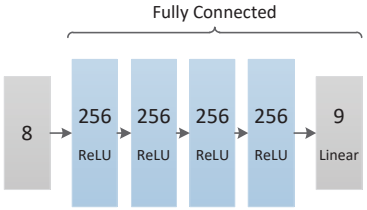


Fig. 3. Network architecture. Each layer is represented by its dimension and activation mode.

IV. Training Process

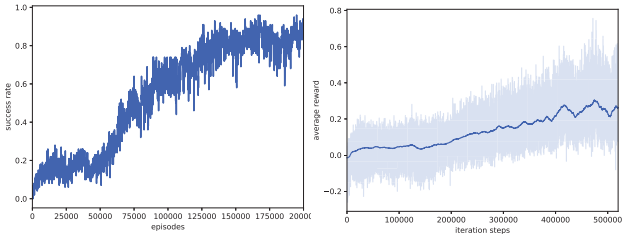
We trained the model with an Adam [20] optimizer on an Nvidia GeForce GTX 1050 Ti GPU. The learning rate α is set to 0.0001, and the reward discount factor γ is 0.99. The environment configuration is introduced in Section II-B, and all simulation parameters are listed in TABLE I.

TABLE I

SIMULATION PARAMETERS

parameter	value	description
x_l	$0m$	lower bound of x coordinate
x_u	$10m$	upper bound of x coordinate
y_l	$0m$	lower bound of y coordinate
y_u	$10m$	upper bound of y coordinate
v_{min}	$0m/s$	minimum linear velocity
v_{max}	$1.2m/s$	maximum linear velocity
$u_{1_{max}}$	$0.3m/s^2$	maximum linear acceleration
$u_{2_{max}}$	$0.8rad/s$	maximum angular velocity
D_{catch}	$0.3m$	capture distance

The training process is divided into two steps. In step 1, the evader is stationary, only the pursuer is trained. In this step, the pursuer is first trained to reach a fixed position. At the beginning of each episode, the states of the pursuer and the evader are randomly initialized. The pursuer then chooses actions based on the epsilon greedy strategy, while the evader stays still. A successful capture is defined as reaching the evader's location before hitting an obstacle or moving out of range. Reward setting of the pursuer is introduced in Section III-C. After 200,000 episodes, the success rate is up to 96%. Fig. 4 shows the success rate and average reward in step 1.



(a) Success rate (b) Average reward

Fig. 4. Success rate and average reward in step 1.

In step 2, pursuer and evader are trained simultaneously during competitive self-play. The model with the highest success rate in the previous step is loaded, and the pursuer is trained on the basis of this model. The evader is trained from scratch. When exploring the escape strategy, the evader will generate various trajectories to train the pursuer, making the pursuer stronger. In turn, the pursuer will spawn a more powerful evader. Through competitive self-play, weaknesses of the agents are continuously discovered and overcome, and the performance of both sides are gradually improved. The reward settings of pursuer and evader are described in Section III-C. A total of 300,000 episodes were trained at this step. Fig. 5 shows the trajectories at different training stages. With the development of training, the capabilities of the agents gradually increase. Algorithms implemented in step 1 and step 2 are shown in Algorithm 1 and Algorithm 2.

Algorithm 1 Training the Pursuer Based on DQN

Initialize replay buffer D_p
Initialize action-value function Q_p of the pursuer with random weights θ_p
Initialize target action-value function $\hat{Q}_p$ with weights $\theta_p^- = \theta_p$
for episode = 1 to M **do**
  Initialize state s_{p_1} and s_{e_1}
  for t = 1 to T **do**
    The pursuer execute action a_{p_t} and observe reward r_{p_t} and next state $s_{p_{t+1}}$
    Store transition $(s_{p_t}, a_{p_t}, r_{p_t}, s_{p_{t+1}})$ in D_p
    Sample a random mini-batch of transitions from D_p
    Update θ_p according to Eq. 3
    Reset $\hat{Q}_p = Q_p$ every C steps
  end for
end for
return Q_p

Algorithm 2 Self-play Based on DQN

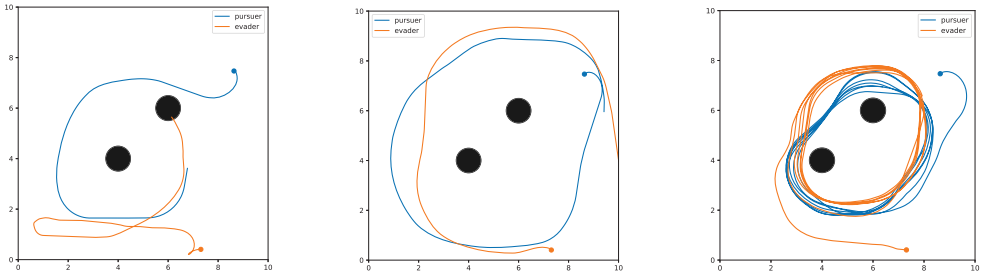
Initialize replay buffer D_p and D_e
Load model Q_p, $\hat{Q}_p$
Initialize action-value function Q_e of the evader with random weights θ_e
Initialize $\hat{Q}_e$ with weights $\theta_e^- = \theta_e$
for episode = 1 to M **do**
  Initialize state s_{p_1} and s_{e_1}
  for t = 1 to T **do**
    The pursuer execute action a_{p_t} and observe reward r_{p_t} and next state $s_{p_{t+1}}$
    The evader execute action a_{e_t} and observe reward r_{e_t} and next state $s_{e_{t+1}}$
    Store transition $(s_{p_t}, a_{p_t}, r_{p_t}, s_{p_{t+1}})$ in D_p, $(s_{e_t}, a_{e_t}, r_{e_t}, s_{e_{t+1}})$ in D_e
    Sample a random mini-batch of transitions from D_p and D_e respectively
    Update θ_p and θ_e according to Eq. 3
    Reset $\hat{Q}_p = Q_p$ and $\hat{Q}_e = Q_e$ every C steps
  end for
end for
return Q_p, Q_e

V. Simulations

In this section, we first verify the effectiveness of curriculum learning, comparing our model with the model trained by self-play from scratch. In addition, we evaluate the pursuer in this paper with the one trained with a random walking opponent from different aspects.

A. Comparison with self-play from scratch

Agents obtained by curriculum learning are denoted as $P_{curriculum}$ and $E_{curriculum}$. In order to evaluate the effectiveness of the curriculum learning applied in this paper, another pair of pursuer and evader were trained by self-play from scratch, which could be named as $P_{scratch}$ and



(a) Trajectory of the 70,000th episode (b) Trajectory of the 240,000th episode (c) Trajectory of the 300,000th episode

Fig. 5. Trajectories with the same initial position at different training stages in step 2. (a) Trajectory after training 70,000 episodes. The evader eventually hit an obstacle. (b) After 240,000 episodes, the total number of steps has increased, but the evader finally exceeds boundary. (c) After 300,000 episodes, pursuer and evader can circle around obstacles for 1000 steps.

$E_{scratch}$. After 300,000 episodes of training, the final model was used for comparison. The statistical result is shown in Fig. 6. We first made $P_{curriculum}$ play against $E_{scratch}$ and recorded the results of 500 rounds. As shown by the yellow columns, the pursuer obtained through curriculum learning could capture the evader which is trained by self-play from scratch at a high ratio of 70%. In contrast, the green columns illustrate the result of $P_{scratch}$ versus $E_{curriculum}$. The probability of $E_{curriculum}$ remaining undefeated is also more than 70%. This indicates that both the pursuer and the evader obtained from the curriculum learning have better performance.

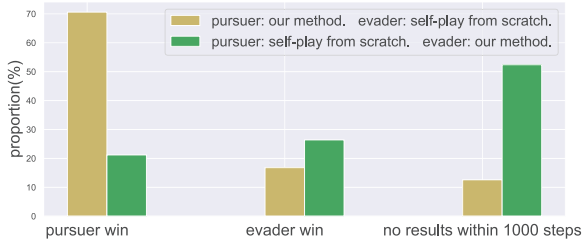


Fig. 6. Results of playing against agents trained by self-play from scratch.

B. Comparison with training on only one side

In this section, we compare the model obtained by our approach with a model trained on only one side. The evader uses a random policy (randomly selects an action from the action space at each step), and the pursuer P_{random} is trained for 500,000 episodes. We firstly compare the training efficiency, which is reflected by the average reward. As shown in Fig. 7, the approach in this paper improves the sampling efficiency and finally reaches a higher average reward.

In addition, two experiments are performed to compare the agents trained by different methods.

1) Play against random evader: Two groups of tests are carried out, in each test, P_{random} or $P_{curriculum}$ is used to chase a random evader. Each test consists of 500 runs, and the positions of the players are randomly initialized. The

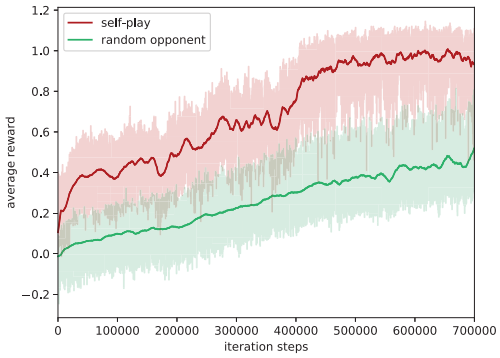


Fig. 7. Comparison of average reward

results are listed in TABLE II. As can be seen from the results, P_{random} which is trained directly against a random evader performs better, and can capture the evader directly. However, $P_{curriculum}$ is more often waiting for the opponent to get out of boundaries or hit an obstacle. Although the win rate is 93%, which is slightly higher than P_{random}'s 91% , the strategy it uses is not aggressive enough.

TABLE II
RESULTS OF PALYING AGAINST RANDOM EVADER

	P_{random}	$P_{curriculum}$
successful capture	336	62
pursuer hit an obstacle	28	15
evader hit an obstacle	26	52
pursuer out of range	17	19
evader out of range	93	351
no results within 1000 steps	0	1

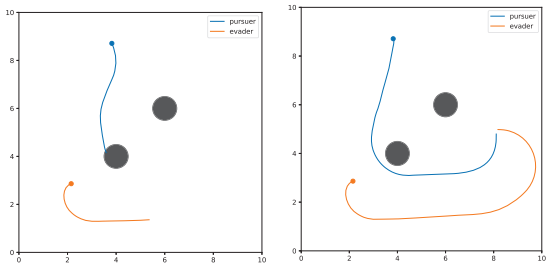
2) Play against intelligent evader: We also compared the performance of $P_{curriculum}$ and P_{random} against an intelligent evader. The evader is $E_{scratch}$ introduced in Section A, which is obtained by self-play from scratch. Similarly, we conduct 500 runs in each test, and the results are shown in TABLE III. When $P_{curriculum}$ acts as the pursuer, the win rate reaches 70.6%. While P_{random}'s win rate as a pursuer is 43.4%. Comparing the two sets of data, it can be seen that

the ratio of $P_{curriculum}$ capturing the target and forcing the target to hit the obstacle is higher. Fig. 6 shows representative trajectories during the game. Based on the results in the two experiments, we infer that our method is more suitable to play against an opponent with certain level of intelligence.

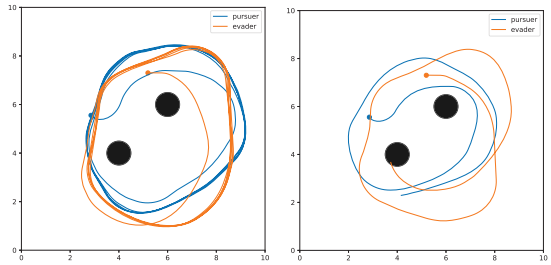
TABLE III

RESULTS OF PALYING AGAINST INTELLIGENT EVADER

	P_{random}	$P_{curriculum}$
successful capture	115	188
pursuer hit an obstacle	69	62
evader hit an obstacle	66	135
pursuer out of range	13	22
evader out of range	36	30
no results within 1000 steps	201	63



(a) Left: P_{random} Right: $P_{curriculum}$



(b) Left: P_{random} Right: $P_{curriculum}$

Fig. 8. Trajectories during the game. On the left is the trajectory of P_{random} against an intelligent evader, and that of $P_{curriculum}$ with same initial positions is on the right. (a) P_{random} hits an obstacle, while $P_{curriculum}$ can avoid the obstacle and successfully catch the target. (b) P_{random} does not win or lose within 1000 steps, $P_{curriculum}$ forces the opponent to hit an obstacle at step 285.

VI. Conclusion

In this paper, a deep reinforcement learning method is applied to train both players in a pursuit-evasion game. In order to be applied to the real-world systems, the kinematics of the agents need to meet the non-holonomic constraints of mobile robot. After that, the effects of obstacles in the environment are also considered. Each robot must perform its main task while avoiding collisions with nearby obstacles. DQN is used in the training of pursuer and evader. The training process is divided into two steps. In step 1, the pursuer is trained to reach a stationary target to provide a purser with a certain level of intelligence. Then in step 2, pursuer and evader are trained simultaneously during competitive self-play. We demonstrate trajectories during the game and conduct several experiments to evaluate our approach. Through comparison and analysis, we infer that agent obtained by our method is more suitable to play against an opponent with certain intelligence. In future work, research efforts will be focused on extending into a continuous action space in order to make the motion of the robot smoother.

References

[1] R. Isaacs, Differential games: a mathematical theory with applications to warfare and pursuit, control and optimization, New York: John Wiley and Sons, 1965

[2] S. J. Benkoski, M. G. Monticino and J. R. Weisinger, "A survey of the search theory literature," Naval Research Logistics, vol. 38, no. 4, pp. 469–494, 1991.

[3] T. Chung, G. Hollinger and V. Isler, "Search and pursuit evasion in mobile robotics," Autonomous Robots, vol. 31, no. 4, pp. 299–316, 2011.

[4] X. Zhang, J. Wang, Y. Fang, H. Gao and J. Yuan, "Multilevel humanlike motion planning for mobile robots in complex indoor environments," IEEE Transactions on Automation Science and Engineering, vol. 16, no. 3, pp. 1244-1258, 2019.

[5] P. Shen, X. Zhang and Y. Fang, "Complete and time-optimal path-constrained trajectory planning with torque and velocity constraints: theory and applications," IEEE/ASME Transactions on Mechatronics, vol. 23, no. 2, pp. 735-746, 2018.

[6] P. Shen, X. Zhang, Y. Fang, and M. Yuan, "Real-time acceleration-continuous path-constrained trajectory planning with built-in tradeoff between cruise and time-optimal motions," IEEE Transactions on Automation Science and Engineering, doi: 10.1109/TASE.2020.2980423.

[7] D. Silver, J. Schrittwieser, K. Simonyan, et al. "Mastering the game of Go without human knowledge," Nature, vol. 550, no. 7676, pp. 354-359, 2017.

[8] OpenAI, "OpenAI Dota 2 1v1 bot," https://openai.com/the-international, 2017.

[9] O. Vinyals, I. Babuschkin, W. M. Czarnecki, et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning," Nature, vol. 575, no. 7782, pp. 350-354, 2019.

[10] J. Schrittwieser, I. Antonoglou, T. Hubert, et al. "Mastering atari, go, chess and shogi by planning with a learnedmodel," arXiv preprint arXiv:1911.08265, 2019.

[11] B. Baker, I. Kanitscheider, T. Markov, et al. "Emergent tool use from multi-agent autocurricula," arXiv preprint arXiv:1909.07528, 2019.

[12] H. Raslan, H. Schwartz and S. Givigi, "A learning invader for the guarding a territory game," Journal of Intelligent and Robotic Systems, vol. 83, no. 1, pp. 55–70, 2016.

[13] M. Egorov, "Multi-agent deep reinforcement learning," http://cs231n.stanford.edu/reports/2016/pdfs/122_Report.pdf, 2016.

[14] F. Zhong, P. Sun, W. Luo, T. Yan and Y. Wang, "AD-VAT: An asymmetric dueling mechanism for learning visual active tracking," in International Conference on Learning Representations, 2019.

[15] S. N. Givigi, H. M. Schwartz and X. Lu, "A reinforcement learning adaptive fuzzy controller for differential games," Journal of Intelligent and Robotic Systems, vol. 59, no. 1, pp. 3-30, 2010.

[16] D. Silver, A. Huang, C. J. Maddison, et al. "Mastering the game of Go with deep neural networks and tree search," Nature, vol. 529, no. 7587, pp. 484-489, 2016.

[17] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever and I. Mordatch, "Emergent complexity via multi-agent competition," arXiv preprint arXiv:1710.03748, 2017.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, et al. "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529-533, 2015.

[19] S. Jagannathan, S. Q. Zhu and F. L. Lewis, "Path planning and control of a mobile base with nonholonomic constraints," Robotica, vol. 12, no. 6, pp. 529-539, 1994.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980 , 2014.

[21] Y. Bengio, J. Louradour, R. Collobert, J. Weston, et al. "Curriculum learning," in International Conference on Machine Learning, pp. 41–48, 2009.