Pursuit-Evasion Game in a bounded game area using deep reinforcement learning and self-play

Mohamed Nadhir Daoud, Hassene Seddik, Ahmad Hably, Chiraz Ben Jabeur

Pursuit-Evasion Game in a bounded game area using deep reinforcement learning and self-play

Mohamed Nadhir DAOUD[1], Hassene SEDDIK[1], Ahmad HABLY[2] and Chiraz BEN JABEUR[1]

***Abstract*— Pursuit-evasion game (PEG) problems are a type of dynamic differential games that received a lot of attention thanks to the ability of this framework to articulate many real-life applications such as in military, aerospace and mobile robotics. Several techniques are used to solve such games, but recently techniques relying on deep reinforcement learning (DRL) gained traction, in particular DRL techniques adapted for problems with continuous action spaces such as Deep Deterministic policy gradients (DDPG). This paper explores the case of a one versus one pursuit-evasion game in a constrained game area, using two twin delayed DDPG (TD3) agents that are trained simultaneously from scratch via self-play only. The simulation results show that agents were performing better than other conventional methods such as Non-linear Model Predictive Control (NMPC).**

I. INTRODUCTION

Mobile robotics is a multi-disciplinary field where robots with a mobility ability autonomously execute specific tasks. One of the core disciplines of mobile robotics is motion planning and control where intelligent techniques are used to guide the robot navigation. Pursuit-evasion game is an example of a dynamic competitive game where two or more robots are playing a zero-sum game. In its simplest form, a pursuer p chases an evader e with the goal of capturing it, while the evader tries to avoid being captured. This simple framework is adapted to many research fields such as in military, surveillance, aerospace and mobile robotics.

The first pursuit-evasion game was introduced by Isaacs under the label "Homicidal chauffeur" [1]. Isaac proposed a game where a fast driver with low maneuverability attacks a slower but agile pedestrian. [2] studied a stochastic version of the homicidal chauffeur game by considering the effect of noise on the game. In more recent work, [3] studied the case of a nonlinear stochastic pursuit-evasion game where the agents were subject to noisy measurements. [4] treated the case of a game with a constrained environment that included moving obstacle. [5] applied real-time nonlinear model predictive control to a pursuit-evasion game of autonomous aircraft. Similarly [6] non-linear model predictive control to a game between unmanned aerial and an unmanned ground vehicles. [7] applied limited information NMPC in a constrained environment to a one versus one pursuit-evasion game. As for the use of reinforcement learning in pursuit-evasion games, [8] used Q-learning, a reinforcement learning algorithm to directly train the individual agents in a 4 vs 1 game. [9] used also Q-learning, but to fine tune the parameters of the fuzzy logic controller for agents operating in a continuous action space. [10] studied a similar set-up but using multi-agent twin delayed deep deterministic policy gradient (TD3), a more advanced DRL technique.

In the case of deep learning for one vs one game, [11] considered the continuous aspect of action spaces in robotics, by using DDPG agents to train both evader and pursuer. However, game environment had neither obstacles nor boundaries, while the agents were trained over 2 separate phases: phase 1 where only the pursuer learns and phase 2 where both agents learn by playing against each other. [12] used Deep Q-network (DQN) to train both the pursuer and the evader in a constrained environment, while comparing two training approaches, namely curriculum learning and self-play. However, due to the use of DQN, the authors were forced to discretize and limit the allowed actions, so to avoid managing a continuous action space.

The goal of this work is to use TD3 for a one vs one pursuit-evasion game, in a constrained environment, to train in single phase simultaneously the pursuer and evader agents, through the use of self-play. Some reinforcement learning basics and recent techniques will be introduced in section I. In section II, a game formulation adapted to the use of reinforcement learning is introduced. Section III provides insight on the reward function used for training. Training algorithm, parameters and evolution are presented in section IV. Section V provides the simulation results and a comparison with a benchmark.

II. REINFORCEMENT LEARNING

A. Basics

Reinforcement learning is a type of machine learning where an agent learns, through trial and error and feedback from the environment, which actions to take in a given situation. Given the environment state s, the agent chooses the action a to perform. Once the action is executed the state of the environment changes accordingly giving a new state s', and the agent receives a corresponding reward r. Reinforcement learning uses Markov Decision Process (MDP) framework to define the interaction between a learning agent and the environment [13], meaning that the probability of reaching the state s' depends only on the immediately preceding state s and action a.

The learning process is generally carried through training episodes composed of a sequence of finite number of transitions or steps, where the final state of an episode is called a terminal state. After each step k the agent is given a reward r_k. The policy π of an agent, described in (1), is the probability of choosing the action a given the state s:

1 with École Nationale d'Ingénieurs de Tunis, hassene.seddik@uvt.tn
2 with Univ. Grenoble Alpes, CNRS, Grenoble INP*, GIPSA-lab, Grenoble 38000, France, ahmad.hably@grenoble-inp.fr

$\pi(s,a) = P(a|s)$ ()

The state value V_π, expressed in (2), is the expected return starting from state s and following the policy π thereafter, until the end of the episode.

$V_\pi(s) = E(\sum_k \gamma^k r_k | s)$ (1)

where $\gamma \in [0,1]$ is a discount factor used to discount future rewards.

Similarly, the action value Q_π, expressed in (3), is the expected return after performing action A starting from state S, and following the policy π thereafter:

$Q_\pi(S,A) = E(\sum_k \gamma^k r_k | s,a)$ (2)

As stated in (4), the optimal policy is the action that maximizes the expected return or the action value of the agent:

$\pi_{opt}(s,a) = \text{argmax}_A Q(s,a)$ (3)

B.Deep Reinforcement Learning

[14] introduced and used deep Q-network (DQN) to play classic Atari games. DQN is known to perform poorly in games with continuous action space, such as control strategy in robotics. Many approaches were introduced to handle continuous action spaces such as Deterministic Policy Gradient (DPG) [15] and actor & critic approaches such as [16]. [17] introduced deep deterministic policy gradient (DDPG) for continuous control, which combines the benefits of DQN, DPG and actor-critic framework . Later, [18] Twin Delayed DDPG or TD3, improving over DDPG shortcomings .

C.TD3 agent

Twin delayed deep deterministic network TD3 is a state-of-the-art policy gradient deep learning algorithm tailored for problems with a continuous action space.TD3 has a main actor-critic network to be trained, and a target actor-critic network, that is softly updated from the main actor-critic network. In the main actor-critic network, the actor π_ϕ is fed with the current state s as input, so it outputs an action a to be executed by the agent. The critic networks Q_{θ_1} and Q_{θ_2} take the current state s and the action a as inputs, to output two estimations Q_1 and Q_2 of the action value.

Similarly, in the target actor-critic network, the target actor $\pi'_{\phi'}$ is given the current state s' as input, so it outputs an action a'. The critic networks $Q'_{\theta'_1}$ and $Q'_{\theta'_2}$ take the new state s' and $\tilde{a}'$, which is a' altered by a clipped random noise ϵ, as inputs to output two estimations of the action value Q'_1 et Q'_2.

ϕ, θ_1, θ_2 are respectively the weights of the actor network and the two critic networks, learned through training. ϕ', θ'_1, θ'_2 are respectively the weights of the target actor network and the two target critic networks, updated periodically using the smooth update factor τ in (6).

$w' \leftarrow \tau w + (1-\tau) w$ ()

Where w is either ϕ, θ_1, θ_2 and $\tau \in [0,1]$ is the smooth update factor. As shown in algorithm 1, similar to other deep reinforcement learning strategy, the training requires the usage of an experience replay buffer.

Algorithm 1: TD3

Initialize pursuer networks π_ϕ, Q_{θ_1} and Q_2 with random parameters ϕ, θ_1, θ_2
Initialize pursuer target networks $\phi' \leftarrow \phi$, $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$
Initialize pursuer Replay Buffer B
for t =1 **to** T **do**
  Select action with exploration noise $a \sim \pi_{\phi_p} + \epsilon$,
  $\epsilon \sim \mathcal{N}(0,\sigma)$ and observe reward r and new state s'
  Execute actions and observe rewards r and new state s'
  Store transition tuple (s,a,r,s') in B
  Sample mini batch of N transitions (s,a,r,s') from B
  $\tilde{a} \leftarrow \pi'_{\phi'}(s') + \epsilon_p$, where $\epsilon \sim clip(\mathcal{N}(0,\tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \, min_{i=1,2} \, Q'_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow argmin_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s,a))^2$
  if t mod d **then**
    Update ϕ by deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s,a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target Networks
    $\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
  end if
end for

While not indicated in algorithm 1, the agent is not used during the first W steps of learning and the actions are rather generated randomly and stored into replay buffer. Depending on the game, the authors used between 1000 and 10000 of these warmup steps.

III.GAME FORMULATION AND NOTATIONS

In this work, a pursuit-evasion game between two non-holonomic robots is studied in a bounded game area, where the pursuer *p* aims to capture the evader *e* as soon as possible, while the evader *e* aims to avoid or delay being captured by *p*. The robots start from the same initial conditions and the positions and orientations of both robots are perfectly known at each step of the game. Every training episode has a maximum number of steps, that if reached, the evader wins the game. The pursuer wins only when it catches the evader before the end of the episode.

The position of the robot is given by the cartesian coordinates of its center (x_i, y_i), while its orientation θ_i is the angle of the robot relative to the x-axis, where *i* is either *p* for the pursuer or *e* for the evader.

At each step k we measure the Euclidian distance D between the centers of the two robots as presented in (7). The pursuer captures the evader when the distance between the robots D is lower than $D_{capture}$.

$$D^{(k)} = \sqrt{(x_p^{(k)} - x_e^{(k)})^2 + (y_p^{(k)} - y_e^{(k)})^2} \quad (4)$$

The robots play in a squared game area of side 2L. The game is stopped if the pursuer or the evader reach the frontiers of the game area. In this case, neither of the robots wins nor loses, but the robot that has reached the boundary has failed.

As in [7], both robots operate at a constant linear velocity v_i, but the pursuer will have a slightly higher speed than the evader. On the other hand, both agents are trained to produce the appropriate angular velocity for the next step $\omega_i^{(k)}$, as long as $|\omega_i^{(k)}| < \omega_{max}$. However, both agents

have the same maximum steering velocity ω_{max}, meaning that no robot have an agility advantage. To be able to train the TD3 agents using reinforcement learning, the observations, the actions the transition function as well as the reward function must be designed. The reward function will be discussed in section IV.

A.Presenting the game observation

An observation O is the observed state of the game at a given time step. It is represented by the positions and orientation of both robots in (8):

$$O^{(k)} = (x_p^{(k)}, y_p^{(k)}, \theta_p^{(k)}, x_e^{(k)}, y_e^{(k)}, \theta_e^{(k)}) \quad (5)$$

When using neural networks for learning, it is common practice to normalize the inputs. The raw observations are scaled down to values in [0,1]. Therefore, s in (9) is the normalized observation that serves as input to the neural net and saved in the replay buffer:

$$s^{(k)} = (\frac{x_p^{(k)}}{L}, \frac{y_p^{(k)}}{L}, \frac{\theta_p^{(k)} mod\ 2\pi}{2\pi}, \frac{x_e^{(k)}}{L}, \frac{y_e^{(k)}}{L}, \frac{\theta_e^{(k)} mod\ 2\pi}{2\pi}) \quad (6)$$

B.Presenting the agent action

As the linear velocities v_i remain constant through the game, there is no action to be produced for the speed. The only action to be produced is the steering velocity $\omega_i^{(k)}$. Therefore, the size of the action domain is 1.

The actor network produces a value in [-1, 1], which is multiplied by the maximum steering velocity ω_{max} for the transition.

C.Presenting the transition function

To make a transition from a state s to the next state s', the kinematic model of the robot must be introduced. In fact, at each step, a non-holonomic mobile robot is represented by:

- the coordinates *(x,y)* of the center of the robot relative to the center of the game area.
- θ which is the orientation its angle relative to the x-axis.
- v its linear velocity
- ω its angular velocity.

The kinematic model of such robot is given by (10) [19].

$$\begin{cases} \dot{x}_i = v_i\, cos\theta_i \\ \dot{y}_i = v_i\, sin\theta_i \\ \dot{\theta}_i = \omega_i \end{cases} \quad (7)$$

The learning is conducted through discrete steps, therefore, the discrete version of (10), given by (11), is used.

$$\begin{cases} x_i^{(k+1)} = x_i^{(k)} + T_s.\, v_i^{(k)}.\, cos\theta_i^{(k)} \\ y_i^{(k+1)} = y_i^{(k)} + T_s.\, v_i^{(k)}.\, sin\theta_i^{(k)} \\ \theta_i^{(k+1)} = \theta_i^{(k)} + T_s.\, \omega_i^{(k)} \end{cases} \quad (8)$$

Where the subscript k denotes the step and T_s the sampling time and *i* is either p for the pursuer or e for the evader.

Using (7), the distance D between the two robots is computed after each transition and fed to the reward function. After the reward is generated, every robot stores its transition $(s, a_i,\ r_i, s')$ in the related memory buffer B_i .

IV.PRESENTING THE REWARD FUNCTION

A.Step reward

Identifying the appropriate reward is critical for a successful training. As the positions of the robots evolve during the game, the criteria that changes accordingly is the distance D between the robots. We tested using - D as a reward for p and D as a reward for e. However, while the pursuer succeeds to learn a good strategy, the evader keeps moving in circles and fails to learn any appropriate strategy.

The alternative is to use the gradient of distance from one step to the next:

$$r_1 = \Delta D * g = (D_k - D_{k+1}) * g \quad (9)$$

where $g = 1000$ is a reward scaling coefficient. The pursuer receives the reward r_1, which, when positive, means that the pursuer is closing the gap. On the other hand, the evader receives a reward of -r_1, which, when positive, means that the evader is distancing the pursuer. As pursuer gets closer to the evader, the r_1 will approach 0, therefore the scaling factor is used to keep giving meaningful feedback to the agent.

B.Outcome reward

Depending on the outcome of the game the pursuer receives the reward r_2 described in (13). The evader receives -r_2.

$$r_2 = \begin{cases} +1000, & if\ \ D \le D_{capture} \\ -1000, & if\ \ step = 1000 \\ 0, & \text{otherwise} \end{cases} \quad (10)$$

C.Time penalty

In order to encourage the pursuer to catch the evader as soon as possible, it is given a small penalty with every new step. The opposite applies also to the evader, a small reward is given to the evader for every step, $r_3 = -10$.

D.Failure penalty

To account for the playground frontiers, the robot that reaches the limits of the play area receives a large penalty r_4. Two variants of this penalty are used. The simplest form of the failure penalty is described in (14).

$$r_4 = \begin{cases} -2000, \text{ if } |x_i| \ge L - 2R_{rob} \text{ or } |y_i| \ge L - 2R_{rob} \\ 0 \quad , \text{ if } |x_i| < L - 2R_{rob} \text{ and } |y_i| < L - 2R_{rob} \end{cases} \quad (11)$$

Where L is the length of the game area and R_{rob} the radius of the robot.

Another form of r_4 is given by (15), that gives a small penalty when the robot enters a buffer zone of width b, prior to reaching the limit of the play area.

$$r_4 = \begin{cases} -2000, & if\ \ |x_i| \ge L - 2R_{rob}\ or\ |y_i| \ge L - 2R_{rob} \\ -10, & if\ \ |x_i| \ge L - 2R_{rob} - b\ \ or\ \ |y_i| \ge L - 2R_{rob} - b \\ 0, & if\ |x_i| < L - 2R_{rob} - b\ and\ |y_i| < L - 2R_{rob} - b \end{cases} \quad (12)$$

A pursuer may face the situation of capturing the evader at exactly the frontier of the game area. r_2 and r_4 were designed such as $|r_4| \gg |r_2|$ in order to prevent the pursuer from destroying itself for a capture.

E.Final reward

The final rewards are given by (16) and (17):

$$r_p = r_1 + r_2 + r_3 + r_{4_p} \quad (13)$$

$$r_e = -r_1 - r_2 - r_3 + r_{4_e} \quad (14)$$

Note that even though the robots are playing a zero-sum-game, r_p and r_e are not always symmetric, specifically when one robot reaches the frontiers of the game area, in this case only the failing robot receives a penalty. It was designed so, in order to avoid rewarding arbitrary actions of an agent, while it had nothing with the failure of the other player.

V.TRAINING

Algorithm 2 details the training steps for two TD3 agents:

Algorithm 2: Training Algorithm

Initialize pursuer networks π_{ϕ_p}, $\mathcal{Q}_{\theta_{1p}}$ and $\mathcal{Q}_{\theta_{2p}}$ with random parameters ϕ_p, θ_{1p}, θ_{2p}
Initialize pursuer target networks $\phi'_p \leftarrow \phi_p$, $\theta'_{1p} \leftarrow \theta_{1p}$, $\theta'_{2p} \leftarrow \theta_{2p}$
Initialize evader networks π_{ϕ_e}, $\mathcal{Q}_{\theta_{1e}}$ and $\mathcal{Q}_{\theta_{1e}}$ with random parameters ϕ_e, θ_{1e}, θ_{2e}
Initialize evader target networks $\phi'_e \leftarrow \phi_e$, $\theta'_{1e} \leftarrow \theta_{1e}$, $\theta'_{2e} \leftarrow \theta_{2e}$
Initialize pursuer Replay Buffer B_p
Initialize evader Replay Buffer B_e
for ep =1 **to** M **do**
  steps ß steps + 1
  for k =1 **to** T **do**
    if steps < *W* **then**
      select random actions for *p* and *e*
    else
      $a_p = \pi_{\phi_p}(s)$
      $a_e = \pi_{\phi_e}(s)$
      Execute actions and observe rewards r_p, r_e and new state s'
      Store pursuer transition (s, a_p, r_p, s') in B_p
      Store evader transition (s, a_e, r_e, s') in B_e
      Sample mini batch of *N* transitions (s, a_p, r_p, s') from B_p
      $\tilde{a}'_p \leftarrow \pi'_{\phi'_p}(s') + \epsilon_p$, where $\epsilon_p \sim clip(\mathcal{N}(0,\sigma), -c, c)$
      Compute $y_p \leftarrow r + \gamma\, min_{j=1,2}\, \mathcal{Q}'_{\theta'_{jp}}(s', \tilde{a}'_p)$
      Update critics $\theta_{jp} \leftarrow argmin_{\theta_{jp}}(\frac{1}{N}\sum(y_p - \mathcal{Q}_{\theta_{jp}}(s,a))^2$
      Sample mini batch of *N* transitions (s, a_e, r_e, s') from B_e
      $\tilde{a}'_e \leftarrow \pi'_{\phi'_e}(s') + \epsilon_e$, where $\epsilon_e \sim clip(\mathcal{N}(0,\sigma), -c, c)$
      Compute $y_e \leftarrow r + \gamma\, min_{j=1,2}\, \mathcal{Q}'_{\theta'_{je}}(s', \tilde{a}'_e)$
      Update critics $\theta_{je} \leftarrow argmin_{\theta_{je}}(\frac{1}{N}\sum(y_e - \mathcal{Q}_{\theta_{je}}(s,a))^2$
      if *k* mod *d* **then**
        Update ϕ_p and ϕ_e by deterministic policy gradient:
        $\nabla_{\phi_p} J(\phi_p) = \frac{1}{N}\sum \nabla_{\phi_p}\pi_{\phi_p}(s)\ \nabla_{a_p}\mathcal{Q}_{\theta_{1p}}(s,a_p)\,|_{a_p=\pi_{\phi_p}(s)}$
        $\nabla_{\phi_e} J(\phi_e) = \frac{1}{N}\sum \nabla_{\phi_e}\pi_{\phi_e}(s)\ \nabla_{a_e}\mathcal{Q}_{\theta_{1e}}(s,a_e)\,|_{a_e=\pi_{\phi_e}(s)}$
        Update target Networks
        $\theta'_{jp} \leftarrow \tau\theta_{jp} + (1-\tau)\theta'_{jp}$
        $\theta'_{je} \leftarrow \tau\theta_{je} + (1-\tau)\theta'_{je}$
        $\phi'_p \leftarrow \tau\phi_p + (1-\tau)\phi'_p$
        $\phi'_e \leftarrow \tau\phi_e + (1-\tau)\phi'_e$
      end if
    end if
  end for
end for

Table 1 provides the main parameters of the training.

Table 1 : Game parameters

Parameter	Value	Description
State dimension	6	
Action dimension	1	
M	10000	Training episodes
P	1000	Steps per episode
W	10000	Warmup steps
X_p	(-2,-8, π/2)	Pursuer initial conditions
X_e	(6,-6, 0)	Evader initial conditions
L	10 m	Game area length/2
b	0.4	Buffer zone width
ω_{max}	π/3 (rad/s)	Maximum steering velocity
v_p	0.5 m/s	Pursuer maximum velocity
v_e	0.48 m/s	Evader maximum velocity
Rrob	0.1 m	Robot radius
T_s	0.1	Sampling time
D capture	0.3 m	Capturing distance
g	1000	Reward scaling factor
noise	μ = 0, σ =0.1	Random noise of 3rd scenario

Since identical agents are used for both players, both agents share exactly the same architecture and hyperparameters described in Table2.

Table 2 : TD3 hyper-parameters

Parameter	Value
Discount factor γ	0.99
Soft update factor τ	0.005
Target policy noise σ	0.2
Policy noise clipping c	0.5
Actor update delay d	2
Actor layers sizes	[256, 256, 1]
Actor activation	[Relu, Relu, tanh]
Critic layers sizes	[256, 256, 1]
Critic activation	[Relu, Relu, None]
Learning rate for actor	0.0003
Learning rate for critic	0.0003
Optimizer	Adam
Replay memory size B	1000000
Minibatch size	256

In the first training scenario, we used the penalty reward described in (14), However, as shown in figure 4, the number of evader wins was very low compared to the failures. Therefore, the reward described in (15) was introduced to indicate to the agent that it is getting close to the boundaries, and that it must escape this zone. The width of the buffer zone influences the training. In fact, the buffer zone reduces the game area, and discourage both agents from exploring trajectories passing by the buffer zone. Therefore, the buffer zone was narrowed to b=0.4 m. In this case the number of the evader's wins increased, but unexpectedly the number of failures increased also.

Another training scenario was also considered. In fact, in real life practice, despite taking the right actions, a robot can find itself out of the desired position, due to several internal and external factors influencing the motion. Therefore, the agents were trained in the case imperfect actions altered by the addition of random normal error (noise), which generates new positions different from the intended ones. This training scenario will obviously reduce the number of wins for both agents and increase the number of failures, especially for the evader which takes riskier paths.

VI.SIMULATION AND COMPARATIVE RESULTS

One of the main challenges encountered in this work was how to evaluate if an agent is performing well. As explained earlier, policy gradient algorithms can diverge at any time during learning even if trained against a fixed trajectory, and as shown in Fig. 5, TD3 makes no exception.

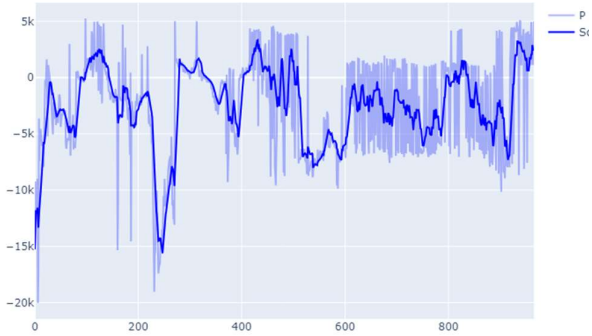

Figure 1: score per episode when p is trained against a fixed evasion trajectory

When training a single agent, it is possible to rely on the episode score to find the best performing agent. But, when training simultaneously 2 agents having different goals, it is harder to tell if the agent with a high score found a better policy or if the opponent diverged. Nevertheless, despite the occasional divergences, the later episodes had much better trajectories than the earlier episodes.

Moreover, since the game step-up is exactly the one described in [7]. This work showcases that, given the linear velocities and sampling time, and with the use of MPC with a prediction horizon N=5, the pursuer will inevitably capture the evader at around 47 seconds and that the evader delays capture by moving as close as possible to the game frontiers. These results are taken as a benchmark for comparison.

Therefore, best agents were selected according to the following criteria:

- Consider only the last 200 episodes
- The number of steps must be above 400, equivalent to 40 seconds.
- Take the agent with the highest score
- Additional check of the validity of the episode

Next the simulation for the evader then the pursuer best episodes are presented, for each training scenario.

A.Self-play scenario

Figure 6 shows the evader moving along the game area limit, until it approaches the corner, where it decides to make a turn following the horizontal area limit. Approaching the capture, the evader tries a last maneuver. At this point the pursuer is so well trained that it anticipates the maneuver:

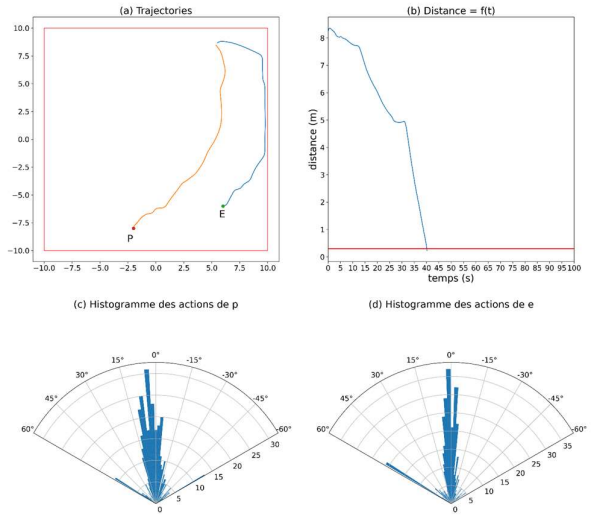

Figure 2: Simulation of agents trained using self-play: (a) game trajectories; (b) distance evolution through game; (c) Circular histogram of Pursuer actions; (d) Histogram of evader actions

The evader, similar to [7], tries to keep as close as possible to the game area limit. However, the pursuer acts a little different than [7], in fact the pursuer does not follow the exact path of the evader, but rather follows a trajectory that traps the evader into the corner in order to close the gap, after the evader makes the turn, to make a capture at around 40 seconds. This is confirmed by the histograms as most of the actions are condensed around 0 rad/s while few extreme actions are taken.

B.Self-play scenario with a buffer zone

Figure 7 shows the result of the simulation when the agents are trained via self-play but with a buffer zone.

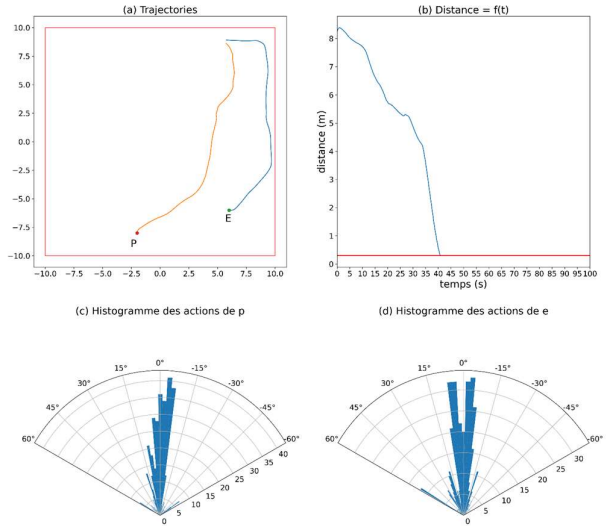

Figure 3: Simulation the PEG with agents trained using self-play + buffer zone approach

As illustrated in Fig. 7, the same behavior seen in **Erreur ! Source du renvoi introuvable.** is confirmed. Nevertheless, it is important to notice that both agents have trajectories slightly skewed to the center of the game area.

C.Self-play scenario with noise

Figure 8 shows the result of the simulation when the agents are trained via self-play but in the presence of noise:

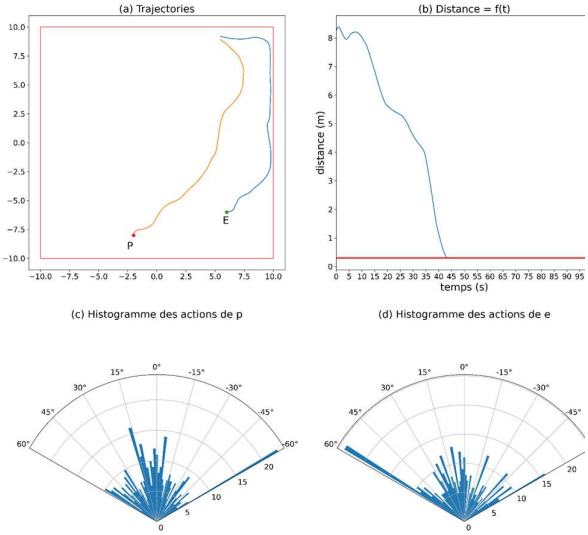

Figure 4: Simulation of the PEG with agents trained via self-play with noisy actions

As illustrated in the figure, the histograms are more dispersed while more extreme actions are taken by both agents, compared to the previous scenarios. It is explained by the noise injection on the actions.

In this case the evader tried also a last chance maneuver just before capture, but it failed to escape.

The agents succeed to find similar trajectories as the two previous training cases. Also, the capture time is slightly higher than the self-play capture time, but it remains lower than the benchmark.

It is justified by the fact that the training considered noise injection, resulting in agents robust to uncertain positions.

CONCLUSION

In this work an algorithm to train two twin delayed deep deterministic policy gradient agents was developed to play a pursuit evasion game in a bounded game area. Two alternate reward functions were introduced to improve the training. The game results were compared to a benchmark using a more conventional, yet efficient, technique.

By choosing the self-play approach, the agents were trained simultaneously, avoiding a 2-step training process generally encountered in other research papers. This self-play approach has the advantage of eliminating human intervention and bias for training, and the need to engineer fixed trajectories for the pursuer pre-training.

Our TD3 agents were able to surpass the performances of the benchmark, by reducing the capture time. Our pursuer proved to be behaving intelligently, by taking advantage of the constraints of the game area, instead of just following the evader path. Our evader shows also an intelligent behavior by maneuvering in the approach of capture.

Moreover, we proved, that agents trained in the presence of noise, perform as good as the normal agents and adjust their policy to account for the noise. The later scenario paves the path for a real-life implementation on real robots.

REFERENCES

[1] ISAACS, Rufus. Differential games: a mathematical theory with applications to warfare and pursuit, control and optimization. Courier Corporation, 1999.

[2] PACHTER, Meir et YAVIN, Yaakov. A stochastic homicidal chauffeur pursuit-evasion differential game. Journal of Optimization Theory and Applications, 1981, vol. 34, p. 405-424.

[3] BASIMANEBOTLHE, Othusitse et XUE, Xiaoping. Stochastic optimal control to a nonlinear differential game. Advances in Difference Equations, 2014, vol. 2014, no 1, p. 1-14.

[4] FISAC, Jaime F. et SASTRY, S. Shankar. The pursuit-evasion-defense differential game in dynamic constrained environments. In : 2015 54th IEEE Conference on Decision and Control (CDC). IEEE, 2015. p. 4549-4556.

[5] EKLUND, J. Mikael, SPRINKLE, Jonathan, et SASTRY, S. Shankar. Switched and symmetric pursuit/evasion games using online model predictive control with application to autonomous aircraft. IEEE Transactions on Control Systems Technology, 2011, vol. 20, no 3, p. 604-620.

[6] TZANNETOS, George, MARANTOS, Panos, et KYRIAKOPOULOS, Kostas J. A competitive differential game between an unmanned aerial and a ground vehicle using model predictive control. In : 2016 24th Mediterranean Conference on Control and Automation (MED). IEEE, 2016. p. 1053-1058.

[7] SANI, Mukhtar, ROBU, Bogdan, and HABLY, Ahmad. Limited Information Model Predictive Control for Pursuit-evasion Games. In : 2021 60th IEEE Conference on Decision and Control (CDC). IEEE, 2021. p. 265-270.

[8] ISHIWAKA, Yuko, SATO, Takamasa, et KAKAZU, Yukinori. An approach to the pursuit problem on a heterogeneous multiagent system using reinforcement learning. Robotics and Autonomous Systems, 2003, vol. 43, no 4, p. 245-256.

[9] DESOUKY, Sameh F. et SCHWARTZ, Howard M. Self-learning fuzzy logic controllers for pursuit–evasion differential games. Robotics and Autonomous Systems, 2011, vol. 59, no 1, p. 22-33.

[10] DE SOUZA, Cristino, NEWBURY, Rhys, COSGUN, Akansel, et al. Decentralized multi-agent pursuit using deep reinforcement learning. IEEE Robotics and Automation Letters, 2021, vol. 6, no 3, p. 4552-4559.

[11] WANG, Maolin, WANG, Lixin, et YUE, Ting. An application of continuous deep reinforcement learning approach to pursuit-evasion differential game. In : 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), 2019. p. 1150-1156.

[12] QI, Qi, ZHANG, Xuebo, et GUO, Xian. A deep reinforcement learning approach for the pursuit evasion game in the presence of obstacles. In : 2020 IEEE International Conference on Real-time Computing and Robotics (RCAR), 2020. p. 68-73.

[13] SUTTON, Richard S., and BARTO, Andrew G., "Reinforcement Leraning, an introduction", 2nd edition, MIT Press, Cambridge, MA, 2018, P 13,49,58,131,437

[14] MNIH, Volodymyr, KAVUKCUOGLU, Koray, SILVER, David, et al. Human-level control through deep reinforcement learning. nature, 2015, vol. 518, no 7540, p. 529-533.

[15] SILVER, David, LEVER, Guy, HEESS, Nicolas, et al. "Deterministic policy gradient algorithms", International conference on machine learning, 2014. p. 387-395.

[16] PROKHOROV, Danil V. et WUNSCH, Donald C. Adaptive critic designs. IEEE transactions on Neural Networks, 1997, vol. 8, no 5, p. 997-1007.

[17] LILLICRAP, Timothy P., HUNT, Jonathan J., PRITZEL, Alexander, et al. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.

[18] FUJIMOTO, Scott, HOOF, Herke, et MEGER, David. Addressing function approximation error in actor-critic methods. In : International conference on machine learning. PMLR, 2018. p. 1587-1596.

[19] JUALIN Luc, "Mobile Robobtics", 6th Ed. Elsevier, 2015. p. 17-17