EasyChair Preprint

№ 7352

Crafting a Robotic Swarm Pursuit-Evasion Capture Strategy Using Deep Reinforcement Learning

Charles H. Wu, Donald A. Sofge and Daniel M. Lofaro

January 19, 2022

Noname manuscript No.
(will be inserted by the editor)

Crafting a Robotic Swarm Pursuit-Evasion Capture Strategy using Deep Reinforcement Learning

Charles H. Wu · Donald A. Sofge · Daniel M. Lofaro

Received: date / Accepted: date

Abstract In this paper we study the multi-agent pursuit-evasion problem, and present an extension of the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) deep reinforcement learning algorithm. Previous pursuit-evasion advancements with MADDPG have focused on training capture strategies dependent on the restriction of evader movement with environmental features. We demonstrate a method to train pursuer agents to collaboratively surround and encircle an evader for reliable capture without a strategy rooted in environment entrapment (i.e. cornering). Our method utilizes a novel two-stage, variable-aggression, continuous reward function based on geometrical inscribed circles (incircles), along with a corresponding observation space, with agents operating in an entrapment-disadvantaged environment. Our results show reliable capture of an intelligent, superior evader by three trained pursuers in open space with our encircling strategy. A key novelty of our work is demonstrating the ability to transition behaviors learned using deep reinforcement learning from a simulated robotic system with imperfect world assumptions to a real-world robotic agents.

Keywords Reinforcement Learning · Swarm Robotics · MADDPG · Hardware

This work was presented in part at the 4th International Symposium on Swarm Behavior and Bio-Inspired Robotics (Online, June 1-4, 2021) [17]

C. Wu - E-mail: chw59@cornell.edu
Cornell University, Ithaca, NY, USA

D. Sofge - E-mail: donald.sofge@nrl.navy.mil
Distributed Autonomous System Group
U.S. Naval Research Laboratory, Washington DC, USA

D. Lofaro - E-mail: daniel.lofaro@nrl.navy.mil
Navy Center for Applied Research in Artificial Intelligence
U.S. Naval Research Laboratory, Washington DC, USA

1 Introduction

The capability for multiple entities to collaborate to achieve a singular goal is studied in the field of swarm robotics, where many homogeneous agents must work together to accomplish complex tasks which cannot be done alone. Such collaborative behavior is prevalent in nature - such as honeybees working together in a hive or sharks teaming up to hunt prey. The multi-agent pursuit-evasion problem (also known as predator-prey), where a swarm of pursuers are tasked with the goal of capturing an adversary, is extremely applicable in many real-world situations. One such application is search-and-rescue, where the exact position of the target may be elusive and constantly changing. Other applications include time-sensitive interception of adversarial combatants, where the primary goal of the pursuers is efficient capture of an opponent trying to escape.

Many of these pursuit-evasion tasks are currently carried out by humans, although unknown environments, unsafe conditions, and response time are some of the many factors that contribute to a need for robot swarms to be deployed to take on these challenges instead. In order for a swarm of robot pursuers to be effective against an evader, they must be able to work collaboratively to execute a dominant strategy. In a collaborative swarm of agents, each agent must be decentralized, making decisions and carrying out actions without commands or information from any central controller. The decentralized nature of swarms allows the success of the swarm to not be dependent solely on one or a few individual agents, but upon the collective whole instead. Therefore, each action is carried out without explicit knowledge of simultaneous actions by other agents. Each pursuer agent relies only on its own observations when selecting actions.

The evader strategy is a key determinant in the pursuers' success. Successful evaders must be intelligent in avoiding pursuers. While fast pursuers with simple autonomies can be effective against a slow evader (i.e. directly following the evader), intelligent autonomous collaboration between pursuers is a necessity to ensure fast and reliable capture against superior evaders.

In real-world pursuit-evasion there are many commonly known capture strategies. One such strategy is cornering, extremely effective when the evader is not actively aware of the strategy or does not have substantial knowledge of its environment. Another strategy is surrounding and encircling. Here pursuers initially move to establish a wide perimeter around an evader before seeking inwards towards the evader. This strategy does not require the use of boundaries to entrap the evader. Instead, precise collaboration between pursuers is necessary to completely encircle the evader. While the lack of environmental assistance shapes this strategy to be more difficult, especially with an intelligent and superior evader, it is promising as a reliable strategy in open areas. We believe it is possible to train a swarm of pursuers to quickly and reliably capture a superior evader in an entrapment-disadvantaged environment with an encircling strategy.

We take a reinforcement learning approach to train the capture strategy of our pursuer agents. Many difficulties arise when training collaborative behavior, such as continuous-control action spaces, where each agent is not limited to a discrete set of actions. When training multiple agents independently, the environment is not stationary from the perspective of any single agent, hindering training efficiency. To combat these issues we implement the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm [7]. This algorithm, based in actor-critic reinforcement learning, builds off of Deep Deterministic Policy Gradient (DDPG) [6] with multiple actor and critic networks to train multiple agents simultaneously. With MADDPG, training for all agents is centralized but agent actors execute actions in a decentralized manner. Note: This work is based off of our previous work Wu et al. [18].

2 Background

2.1 Previous Research

Previous research on variable, multi-objective reward functions for similar multi-agent reinforcement learning tasks has also yielded promising results. In the VIP-bodyguards problem defined in Sheikh et al. [11], each bodyguard agent receives an exponential reward based on the proximity of threats to a VIP. With multiple distinct factors contributing to the final total reward, complex behaviors can emerge in the final learned strategy. This reward function proved effective, encouraging aggressive, collaborative attempts from the agents to achieve their goal of maximizing VIP safety.

Decentralized fuzzy control reinforcement learning actor-critic algorithms have also been studied, based on the Circles of Apollonius created by multiple pursuers and a single evader [2]. This study focused on an evader with multiple objectives: reaching a designated location in the environment as the primary goal, and avoiding capture as a secondary objective. Evaders were also trained via reinforcement learning, and could learn a non-ideal evasion strategy. The evader is superior only in linear speed, and the evader starting position is constant, in the center of the environment. This could contribute to less adaptable strategies learned by the pursuers, which likely spawn already encircling the evader.

A similar problem of multi-agent robotic swarm 2D construction [1] has also been considered, where large swarms of robots must form structures from many prepositioned building blocks in a planar environment. Here, a wide number of simple, real-world robotic agents are mechanically programmed and a hardware setup for monitoring the interactions and collaborative behavior of agents is introduced.

In pursuit-evasion problems, appropriate formation control of pursuers is necessary to ensure that pursuers are spread out in a uniform angular distribution around an evader [15], allowing pursuers to complete the encircling task of surrounding an evader evenly from all directions.

With respect to environment observability, various experiments with partial visibility have been highlighted in discussion [16]. However, studying the learning behaviors of pursuers is simplified in a full-visibility environment, where pursuers can train more efficiently without restrictions on knowledge about the environment or the evader.

2.2 Reinforcement Learning

The field of reinforcement learning (RL) has widely progressed over recent years with advancements in computing technology capable of running many episodes, e.g. examples, to train more complex and functional behaviors in agents.

In the general reinforcement learning environment, agents view a state s_i, determine an action a_i to execute, and receive a reward r_i from the environment. The agent then progresses into state s_{i+1}, where the process continues until the episode terminates. The agent's decision-making process can be described as a policy π, and the whole of the environment that the agent operates in can be represented by a Markov Decision Process

(MDP) with transition probabilities of $P(s_{i+1}|s_i, a_i)$ between subsequent states given possible actions. Policy π must determine the best possible action for an agent to take at a given state. An optimal RL policy will maximize an agent's total expected future reward.

Many advancements in RL, specifically in continuous-control actor-critic methods, have paved the way for collaborative training algorithms.

2.3 DDPG

Deep Deterministic Policy Gradient (DDPG) [6] introduced by Lillicrap et al. extends the commonly known deep reinforcement learning pioneer Deep-Q Network (DQN) [9] with a Deterministic Policy Gradient (DPG) [12] for use on continuous-control problems. An actor-critic algorithm, DDPG is composed of two neural networks, a critic network which predicts a state-action value from a given state s_i and action a_i taken at s_i, and an actor network, the agent policy returning actions a given states s. This state-action value, Q, can be represented by $Q(s, a|\theta^Q)$. The actor network, which predicts a continuous action from s, is given as $\mu(s|\theta^\mu)$.

The critic network is updated with the loss:

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

The actor network is subsequently updated by the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|^{a=\mu(s_i)}_{s=s_i} \nabla_{\theta\mu}\mu(s|\theta^\mu)|_{s_i} \tag{2}$$

To increase the long-term stability of the actor and critic networks during training, target networks with a soft target update are used. The target networks are initialized identical to their counterparts, but their weights are updated gradually over time as the networks train. Where $\theta^{'}$ and θ are the weights of the target networks and normal networks and $\tau < 1$ is the target update rate, $\theta^{'} \leftarrow \tau\theta + (1-\tau)\theta^{'}$. The critic network update utilizes a process of the Bellman Equation:

$$y_i = r_i + \gamma Q^{'}(s_{i+1}, \mu^{'}(s_{i+1}|\theta^{\mu^{'}})|\theta^{Q^{'}}) \tag{3}$$

Here, γ is the discount factor for future reward estimations and the target actor and critic networks are used to keep predictions stable while training.

2.4 MADDPG

Multi-Agent Deep Deterministic Policy Gradient, known as MADDPG, [7] furthers the advancement of DDPG by introducing a framework for training multiple agents simultaneously.

The MADDPG algorithm has an actor and critic network for each agent. Each agent's critic network is centralized and has knowledge of every agent's observations and actions while training. Since the critics learn independently, differing reward functions could be used to train unique strategies among agents. However, when working with homogeneous agents, the critics train similarly. The decentralized actor only has local observations as input, and during execution, only the actor's policy is used to predict agent actions. With MADDPG, the critic network trains by:

$$y^j = r_i^j + \gamma Q_i^{\mu^{'}}(x^{'j}, a_1^{'}, a_2^{'}, a_3^{'}, ..., a_N^{'})|_{a_k^{'}=\mu_k^{'}(o_k^j)} \tag{4}$$

and updates with the critic loss given by:

$$\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j (y^j - Q_i^\mu(x^j, a_1^j, a_2^j, a_3^j, ..., a_N^j))^2 \tag{5}$$

The sampled policy gradient then updates the actor network:

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i}\mu_i(o_i^j)\nabla_{a_i} Q_i^{\boldsymbol{\mu}}(x^j, a_1^j, a_2^j, ..., a_N^j) \tag{6}$$

where

$$a_i = \mu_i(o_i^j) \tag{7}$$

Both MADDPG and DDPG utilize experience replay buffers to train. This buffer stores past experiences consisting of states, actions taken by all agents at given states, the resulting rewards, and the following states. Experience at state s_i with N agents:

$$\begin{aligned} Experience(s_i, N) = \{&s_i, a_1^i, a_2^i, a_3^i, ..., a_N^i, \\ &r_1^i, r_2^i, r_3^i, ..., r_N^i, s_{i+1}\} \end{aligned} \tag{8}$$

Attempts at using MADDPG (and various extensions) for pursuit-evasion [13][8] have shown great success. In Singh et al., from randomized initial positions on a square, 2D, bounded planar simulation environment, four pursuers learned to quickly capture a superior evader. While a reliable capture strategy was learned, capture occurred frequently near the bounds of the environment, against the sides and within the square corners. The evader strategy used by Hüttenrauch et al. [5] has the evader maximize its own Voronoi region area with respect to Voronoi points of all agent positions. In contrast, our method stands out as unique compared with these other methods in that it does not rely upon environmental entrapment (e.g., cornering or pinning the evader against a wall or obstacle); instead, the agents learn to capture the evader by encircling it.

3 Methodology

Our approach is to model the agents and the environment using the SCRIMMAGE multiagent simulator. Trials and scenarios are created and simulated in SCRIMMAGE, with the results of each (including control actions, agent positions, etc.) recorded and saved for each run. We use the Tensorflow 2 framework to train our pursuer agents using MADDPG. The evader is programmed using a predefined (not learned) strategy such that the evader always takes the ideal evasive action against the pursuers. A novel reward function is crafted in order to encourage the pursuers to encircle the evader, thereby minimizing the dependence of the technique on environmental features such as corners, walls, or obstacles.

3.1 Deep RL Neural Network Training

The training and testing of the pursuer agents is supervised by a Python3.6 program. Using the Tensorflow 2 framework, along with optimized Tensorflow functions, replay buffers, and simulation environment wrappers provided by the Tensorflow-Agents library [4], we created a custom program to train our three pursuer agent MADDPG.

During training, noise generated using a Ornstein-Uhlenbeck process is introduced to the output of the actor network. This encourages exploration of the action space - important to achieve more varied training examples.

All layers in the agent actor networks are fully connected. For the critic networks, the state observation input runs through a fully connected layer of 256 nodes independently. After this layer, the agent action input is concatenated with the state layer and subsequently run through a fully connected 128 node layer, which is finally connected with the output layer.

During every training timestep, the experience replay buffer memory is updated. Once per training step, a batch of experiences is sampled from this buffer to train the critic networks, and subsequently the actor networks. The training environment updates once every 0.1 seconds in simulation, and we consider each update a simulation step, with agents reevaluating actions with their actor networks at each of these intervals. The full network training parameters can be found in Table 1. Figure 1 shows an example of training graphs and the network structures.

3.2 The SCRIMMAGE Simulator

We run training and evaluation of agents in the SCRIMMAGE [3] simulation environment, a multi-agent three-dimensional robotics simulator. We created SCRIMMAGE plugins in C++ to interface with our Python3 reinforcement learning training scripts. Custom sensor plugins for pursuer agents allow for individual sensing and reporting of surroundings to be aggregated and fed into the actor neural networks. Custom collision metrics allow for the detection of collisions between pursuer and evader agents only, while pursuer agents are allowed to intersect one another to streamline simulation dynamics. While SCRIMMAGE has the capability for 3D simulation, our custom agent motion controllers restrict movement to a 2D plane with the agent height locked at $z = 0$. This is because we desire to observe agent behavior in a 2D environment. With SCRIMMAGE, we are able to randomize starting locations throughout the environment area for all agents - pursuers and evader - at the beginning of each episode. We chose SCRIMMAGE because of its flexibility in customization for agents and environments and its robust scalability.

Table 1 Network training parameters for MADDPG

Network Training Parameters	
Name	Value
Replay Buffer Size	$1,000,000$
Ornstein-Uhlenbeck Noise σ	0.2
Ornstein-Uhlenbeck Noise ζ	0.15
Target Network Update Rate	0.001
Target Network Update Period	100
Batch Size	256
Actor Learning Rate	0.0001
Critic Learning Rate	0.0003
Reward Discount γ	0.99
Max Episode Length	250.0 (simulation sec)
Max Episode Steps	2500

3.3 Pursuit-Evasion Environment

The environment model within SCRIMMAGE was designed to maximize an evader's escape possibilities while still maintaining boundaries for environment limits, ensuring a superior evader cannot escape from the realm of capture possibility. Our goal is to train pursuers to fully encircle evaders without the assistance of environment boundaries. Because of this, we chose to implement a circular 2D environment to deter evader cornering, commonly observed within polygonal environments with sharp vertices. The environment is centered at $(0, 0)$ in Cartesian coordinates and has a radius of 70 meters. Attempts to move across the boundary are restricted.

All pursuer agents are identical. All agents in the environment operate with a unicycle model with linear speed kept constant. Agents therefore have control only over angular velocity. This was done to simplify the output of the actor neural networks with only one output variable to control the pursuer behavior. We define an evader as superior if it has both a greater maximum linear speed and maximum angular velocity than its

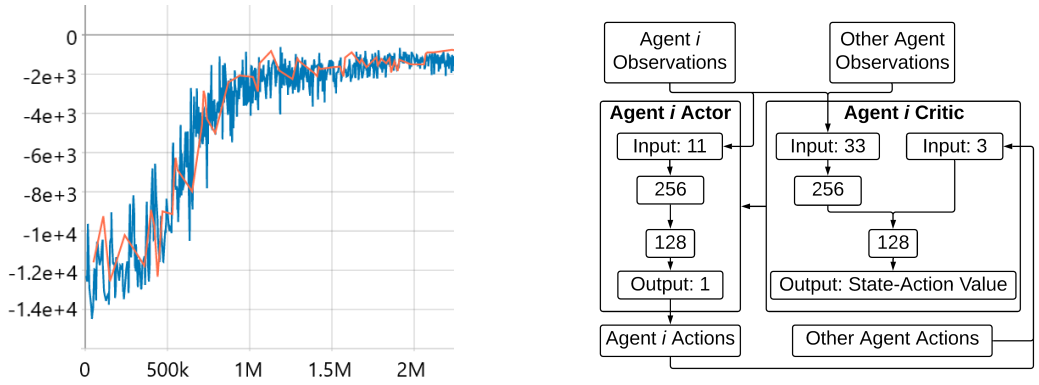

Fig. 1 Training Overview: **(LEFT):** Reward over training steps, **(RIGHT):** network structure for single agent. Converges at around 2M steps (7000 episodes). Graph shows a rolling average of total episode reward over a 10-episode frame. Blue graph indicates training episode results (with noise), orange graph indicates intermediate testing episodes (no noise). Note: The data transferred from the "Agent i Critic" to the "Agent i Actor" is the critic error from the previous epoch.

respective pursuers. We define a collision between the evader and any pursuer (see 3.2) as a successful capture. Each agent's pose is defined by a yaw orientation and x/y position. Agents are simulated as a spherical model with a radius of 2.5 meters. This gives the pursuer agents a capture reach of 5 meters directly from pursuer to evader location.

During training, evader abilities are temporarily weakened (max angular velocity limited to 50% of actual value) while pursuers learn to capture. This is done to provide more initial training examples of the evader being within the control circle, speeding up the overall training process. Testing and evaluation is done with the evader at full strength, with no restriction on maximum abilities.

3.4 Evader Autonomy

We created a predefined evader autonomy that utilizes Voronoi regions to determine actions. Voronoi regions in 2D space are areas A_i defined by Voronoi points $v_{x,y}$ where all points $p_{x,y} \in A_i$ given the Euclidean distance between $p_{x,y}$ and $v_{x,y}$ is less than the distance to any other $v_{x,y}$. Using the Voro++ library[10], Voronoi tessellations are calculated in the environment at each timestep. The positions of the evader and pursuers are given as Voronoi points, and the final diagram is bounded by the circular environment boundary. The evader defines its ideal target position as the centroid of the Voronoi region it occupies. By seeking towards the centroid, the evader moves to increase its distance from pursuers.

Since this evader autonomy is a predefined function and not learned, it is reliable and infallible towards previously unseen environment state examples. The Voronoi regions are calculated with full observability, and the evader is guaranteed to be taking its most ideal evasive action.

After initial testing, we observed that pursuers would consistently use the circular wall of the environment to restrict evader movement, leading to simpler capture. To discourage this behavior, we modified the evader autonomy to recognize a virtual set of Voronoi points on the entire boundary circumference, along with the previously considered points from all agent positions. This final autonomy successfully discouraged the evader from nearing the environment boundary unnecessarily.

3.5 Novel Reward Function

To encourage encircling behavior by pursuers, we designed a two-stage, variable-aggression, continuous reward function for pursuers based on geometrical incircles. The three pursuers that we implement can be viewed as vertices that form a triangle. We consider this triangle's inscribed circle to represent a "control circle" for the pursuers. This control circle can also be regarded as a search perimeter set by the pursuer positions.

Inspired by real-life search tactics, we aim to create a strategy for the pursuers that works in multiple stages to achieve eventual capture. First, the pursuers establish a wide perimeter around the evader by quickly seeking outwards in a strategic manner until the evader is encompassed within the control circle. Afterwards, the pursuer strategy aims to shift the control circle to center the evader. At the same time, the pursuers move inwards, contracting the perimeter. As the evader tries to escape and create distance, the pursuers dilate and contract the perimeter accordingly to keep the evader within the bounds of the control circle while still attempting to close the gap. To capture the evader, the pursuers must shrink the control circle enough for at least one agent to make contact with the evader. The success of the pursuers depends on balancing the aggressiveness in shrinking the perimeter with the reliability of keeping the evader within control.

Fig. 2 (LEFT): Example of pursuer agents (green) encircling evader (red) within control circle. **(MIDDLE):** Real-time overhead view of Vector robots from external tracking camera, tracking robots via AprilTags. **(RIGHT):** Anki Vector robots in testing environment. The Vector robots are used to test our algorithm in the real-world in real-time.

Table 2 Parameters used to shape the pursuer agent reward.

Reward Parameters	
Name	Description
$\Delta_{ic\to e}$	Distance between evader and incircle center
Δ_{max}	Maximum possible distance between two points in the environment
w_c	Weight of evader centering while pursuing perimeter contraction
$\Delta_{p_i\to ic}$	Pursuer p_i's distance to incircle center
Δ_{avg}	Average distance to incircle center for all pursuers
N	Number of pursuers
$\Delta_{in\to e}$	Normal distance between evader and closest point on incircle circumference
$\Delta_{max_{ic\to p_i}}$	Maximum possible distance between pursuer and incircle center (all pursuers are widely spread out)
α	Variable aggression modifier constant

If pursuers lack aggression in shrinking the perimeter ("playing it safe"), capture would be slow and infrequent. However, if pursuers are too aggressive, evader control could be sacrificed needlessly if pursuers are too willing to attempt capture in risky situations. If the evader does manage to escape the control circle, pursuers must immediately recognize the fault and transition back into the behavior of rapidly expanding the search perimeter to include the evader. Table 2 shows the parameters used to shape the pursuer agent reward.

From these parameters, β, the reward behavior modifier, can be calculated. β is derived from the sign of $\Delta_{in\to e}$, and $\beta = 1$ if the evader is outside the pursuers' control circle, and $\beta = -1$ if the evader is within control.

The reward is composed of three main factors: a formation position metric, a formation evenness metric, and a measure of pursuer closeness to the evader.

The formation position metric P_r, meant to encourage the pursuers to move the control circle center near the evader, is calculated as:

$$P_r = \frac{\Delta_{ic\to e}}{\Delta_{max}} \cdot w_c \tag{9}$$

The formation evenness metric keeps pursuers in even separation and direction around the evader through an incentive for equidistant spacing to the control circle center relative to the average spacing of all pursuers. This metric, E_r is calculated as:

$$E_r = \frac{|\Delta_{p_i\to ic} - \Delta_{avg}|}{\Delta_{avg} \cdot (N-1)} \tag{10}$$

The pursuer closeness metric, a variable factor depending on the state of the evader location relative to the control circle, encourages pursuers to close the gap to the evader by tightening the perimeter and is calculated as C_r:

$$C_r = 1 - \frac{\Delta_{p_i\to ic}}{\Delta_{max_{ic\to p_i}}} \tag{11}$$

This closeness metric is implemented in an exponential manner to increase motivation to capture the evader as the pursuers close in, even if a tightened control circle results in more risk for losing control of the evader. If a pursuer succeeds in capture, an additional bonus of 100 is added onto the final reward.

Where c is a constant that ensures the total base reward is negative, this reward R for each pursuer is therefore given by:

$$R = -P_r - E_r - c\begin{cases} +0, & \text{if } \beta >= 0. \\ +\alpha(C_r \cdot e^{C_r}), & \text{otherwise.} \end{cases} \tag{12}$$

3.6 Observation/Action Space

During the training phase the observations of each pursuer are aggregated into an input for the MADDPG critic networks. These same observations are fed independently to each pursuer's actor network to determine their specific actions. During testing and evaluation, only the actor network is used, and therefore each pursuer operates with knowledge of only its own observations.

The observation taken by each pursuer from its environment surroundings is as follows. For a pursuer p_i against evader e with incircle center ic:

$$Obs(p_i, e, ic) = (x_i^{self}, y_i^{self}, \theta_i^{self}, x^e, y^e, \theta^e, x^{ic}, y^{ic}, \Delta_{avg}, \Delta_{p_i\to ic}, \Delta_{in\to e})$$

(13)

The action space for the actor network of each pursuer agent is a single output of angular velocity, limited from -1 to 1, scaled to the agent's max angular velocity in the simulation environment.

4 Experimentation

4.1 Simulation

Through our experiments we aim to evaluate the learned coordinated capture strategy, and thereby the training algorithm. A total of 500 testing episodes are run per condition. All episodes are run and recorded in the SCRIMMAGE simulation environment, with custom output statistics for our testing metrics. With the exception of removing the output noise on the pursuer actions (implemented during training), all network and environment features remain constant between training and testing episodes. Any episode where capture is not achieved within the time limit is terminated, progressing to the next testing episode. We do not count aforementioned episodes in our average episode length metric, but these episodes are considered in our capture success metric.

Trials on three conditions are run. Each condition has varying pursuer and evader max speeds. Condition 1 pairs pursuers against evaders with equal strength, while Conditions 2 and 3 challenge the pursuers with evaders of superior ability.

To establish a baseline, another MADDPG model is trained in the same environment, with a observation method and reward function as presented in a previous, non-encircling implementation of MADDPG [13], which we will refer to as a standard solution to compare with our encircling solution.

4.2 Hardware

Trials were also run in the real world (discussed in Results section below), with robot pursuers implementing the trained simulation policy deployed in a closed ground environment. The robots we use are COTS Anki Vector robots. An AprilTag [14] tracking marker is attached to each robot for localization, and an external overhead camera is used to determine the position and orientation of each robot within the environment, as seen in Figure 2. This setup is similar to [1], and a full overhead view allows for accurate tracking of all agents simultaneously.

This observation is fed into a controller that evaluates the trained policies of each pursuer agent independently, retaining the decentralized behavior of all agents. Robots are controlled via the Anki Vector Python SDK. Each robot has two tank treads and a maximum speed of 220 mm/sec. The angular velocity output of the pursuer policy is translated to fit the differential drive capability of the robots. The robots respond similarly to the simulation agents, only with constant linear speeds scaled down to match their physical limitations.

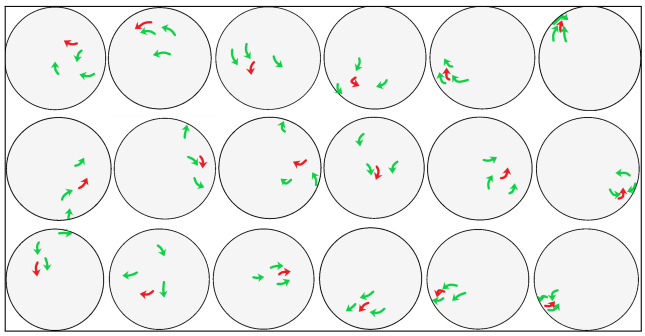

Fig. 3 Sample trials from Conditions 1, 2, and 3 (shown in respective rows). Each frame is a snapshot from an episode. Graphs are taken from the SCRIMMAGE simulator and edited for clarity. Green and red trails are drawn to clarify pursuer and evader motion.

5 Results

Results from trials on all three simulation conditions with our encircling solution can be seen in Table 3. The validation condition for pursuer efficacy, Condition 1, where the pursuer and evader abilities are equal, yielded a capture rate of 100%. Capture was also fully successful on Condition 2 trials, and capture in Condition 3 remained reliable, with a capture rate of 93.2% against a very superior evader.

The average time to capture increases as the conditions progress in difficulty, resulting from multiple attempts at capture within a single episode and a slower initial control circle formation process. As the evader becomes more superior, the pursuers are able to recognize and rebound from failed capture attempts to immediately reset and try again with their capture strategies.

As seen in Figure 3, the pursuers initially position and orient themselves in a wide perimeter around the evader, before proceeding to move inwards, collaboratively shrinking the perimeter to surround the evader and achieve capture. Through the simulation episodes we observed, there was no direct evidence of capture where pursuers directly utilized environment bounds to restrict movement or block the evader. All captures we observed occurred in open space. It is possible that the environment bounds were used to restrict movement or block the evader even though there was no direct evidence of such events occurring. This remains open for further study.

In comparison, the results from these trials in the same simulation environment and environmental setup evaluated on the standard solution MADDPG model

Table 3 Results and statistics from trials with the encircling solution: Where C# is the Condition #, V_p - Pursuer's maximum translational velocity (m/s), ω_p - Pursuer's maximum rotational velocity (rad/s), V_e - Evader's maximum translational velocity (m/s), ω_e - Evader's maximum rotational velocity (rad/s), C_r - Capture Rate (%), and t_a - Average Time to Capture (sec). Each of these statistics were gathered for each of the three conditions.

C#	V_p (m/s)	ω_p (rad/s)	V_e (m/s)	ω_e (rad/s)	C_r (%)	t_a (sec)
1	10.0	1.0π	10.0	1.0π	100	22.33
2	10.0	1.0π	11.0	1.1π	100	34.74
3	10.0	1.0π	12.5	1.25π	93.2	78.22

can be seen in Table 4. While the average time to capture for each of the three conditions is lower than its counterpart with the encircling solution due to the additional time required to position and maintain a control circle, the standard solution exhibits a degradation in capture rate as the conditions progress in difficulty and the evader gains more superior abilities. In Condition 3, the standard solution's capture rate of 79.8%, when compared to the encircling solution's 93.2% capture rate, highlights the increased efficacy of the encircling strategy against superior evaders in this environment.

The encircling approach is of course limited by the relative maneuverability of the evader with respect to that of the pursuers. For evaders with a significantly higher maximum translational velocity, when compaired to the pursuers, in an open arena we expect that the superior evader, using an idealized evasion strategy, will always succeed regardless of what pursuit strategy is employed. In our work we determined that if the evader's translation and rotational velocities are 125% faster than the pursuers then the pursuers will achieve a capture rate of 93.2%. If the evader's translation and rotational velocities are 110% faster than the pursuers then the pursuers will achieve a capture rate of 100% (see Table 3).

Due to space constraints and latency introduced from a remote, virtual test setting, it was difficult to run capture trials to scale within the limited testing environment, and full scale real-world tests were unable to be completed. However, initial observations of the Vector robots running the implemented, scaled-down pursuer autonomy were promising, with the agents exhibiting the same control circle formation behavior as seen in successful simulations. This encouraging reflection paves the way for more complete and complex trials to be carried out in the future with the Vector robots and their constructed testbed.

6 Conclusion

We have shown that our extension of the MADDPG deep reinforcement learning algorithm with a novel encircling reward function is effective in training pursuer agents to capture a superior evader. Our reward function allows pursuers to learn to succeed with capture in a circular, entrapment-disadvantaged environment against an intelligent evader with a predefined algorithmic autonomy actively avoiding environment bounds. We tested against evaders of varying superiority, all resulting in efficient capture with only three pursuers. When compared to a standard solution, the encircling solution demonstrates greater reliability in capturing increasingly superior pursuers, despite this capability leading to a longer average time to capture.

Table 4 Results and statistics from trials with the standard solution. Where C# is the Condition # [13].

C#	V_p (m/s)	ω_p (rad/s)	V_e (m/s)	ω_e (rad/s)	C_r (%)	t_a (sec)
1	10.0	1.0π	10.0	1.0π	100	13.02
2	10.0	1.0π	11.0	1.1π	99.2	30.22
3	10.0	1.0π	12.5	1.25π	79.8	59.48

Through our testing, no form of pursuers cornering or using the bounds of the circular environment was observed. Instead, capture was achieved in open space.

In conclusion, unlike previous pursuit-evasion strategies trained using MADDPG, we have found that our method is an adaptable solution that encourages pursuers to collaboratively establish a search perimeter and precisely encircle an intelligent evader. Additionally, though our method is tested on groups of three pursuing agents, it will work with N number of agents where $N \geq 3$ and cause the agents to form a crystalline structure around the evader. Each of the N agents will not have to be trained individually and only needs to behaviors obtained from the three on one training described in this work. Finally, a key novelty of our work is demonstrating the ability to transition behaviors learned using deep reinforcement learning from a simulated robotic system with imperfect world assumptions to real-world robotic agents.

Future works could involve different metrics to determine a control circle or expand the actions of the pursuers and evader to allow for controlled speeds. Applications of this encircling algorithm to 3D spaces (where agents would have to spherically encircle) and other real-world robotic platforms could also be explored.

Acknowledgements This work was performed at the U.S. Naval Research Laboratory and was funded by the U.S. Naval Research Laboratory under the project "Adaptive Real-Time Algorithms for Multiagent Cooperation in Adversarial Environments".

The views, positions, and conclusions expressed herein reflect only the authors' opinions and expressly do not reflect those of the U.S. Naval Research Laboratory, nor those of the Office of Naval Research.

References

1. Andreen, D., Jenning, P., Napp, N., Petersen, K.: Emergent structures assembled by large swarms of simple robots. In: Posthuman Frontiers (2016)
2. Awheda, M.D., Schwartz, H.M.: A decentralized fuzzy learning algorithm for pursuit-evasion differential games with superior evaders. J Intell Robot Syst (2016)
3. DeMarco, K., Squires, E., Day, M., Pippin, C.: Simulating collaborative robots in a massive multi-agent game environment (SCRIMMAGE). In: Int. Symp. on Distributed Autonomous Robotic Systems (2018)
4. Guadarrama, S., Korattikara, A., Ramirez, O., Castro, P., Holly, E., Fishman, S., Wang, K., Gonina, E., Wu, N., Kokiopoulou, E., Sbaiz, L., Smith, J., Bartók, G., Berent, J., Harris, C., Vanhoucke, V., Brevdo, E.: TF-Agents: A library for reinforcement learning in tensorflow (2018). URL https://github.com/tensorflow/agents
5. Hüttenrauch, M., Sosic, A., Neumann, G.: Deep reinforcement learning for swarm systems. CoRR (2018)
6. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. CoRR (2016)
7. Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., Mordatch, I.: Multi-agent actor-critic for mixed cooperative-competitive environments. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, p. 6382–6393. Curran Associates Inc., Red Hook, NY, USA (2017)
8. Mao, H., Zhang, Z., Xiao, Z., Gong, Z.: Modelling the dynamic joint policy of teammates with attention multi-agent DDPG. CoRR (2018)
9. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature (2015)
10. Rycroft, C.: Voro++: A three-dimensional voronoi cell library in c++. Chaos: An Interdisciplinary Journal of Nonlinear Science (2009)
11. Sheikh, H.U., Bölöni, L.: Designing a multi-objective reward function for creating teams of robotic bodyguards using deep reinforcement learning. ArXiv (2019)
12. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M.: Deterministic Policy Gradient Algorithms. In: ICML (2014)
13. Singh., G., Lofaro., D., Sofge., D.: Pursuit-evasion with decentralized robotic swarm in continuous state space and action space via deep reinforcement learning. In: Proceedings of the 12th International Conference on Agents and Artificial Intelligence - Volume 1: ICAART,, pp. 226–233. INSTICC, SciTePress (2020)
14. Wang, J., Olson, E.: Apriltag 2: Efficient and robust fiducial detection. In: 2016 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems (IROS) (2016)
15. Wang, X., Cruz, J., Chen, G., Pham, K., Blasch, E.: Formation control in multi-player pursuit evasion game with superior evaders. Proceedings of SPIE - The International Society for Optical Engineering (2007)
16. Weintraub, I.E., Pachter, M., Garcia, E.: An introduction to pursuit-evasion differential games (2020)
17. Wu, C., Lofaro, D., Sofge, D.: A Learned Encircling Strategy for Robot Swarm Pursuit-Evasion Against a Superior Evader. In: The 4th International Symposium on Swarm Behavior and Bio-Inspired Robotics (2021)
18. Wu, C., Lofaro, D., Sofge, D.: A learned encircling strategy for robot swarm pursuit-evasion against a superior evader. In: The 15th International Symposium on Distributed Autonomous Robotic Systems (DARS) (2021)