Decentralized Multi-Agent Pursuit Using Deep Reinforcement Learning

Presenter: Liu Xinzi

2023.4.24

How to achieve efficient pursuit of moving targets

A high-level description of the problem to be solved

- With multiple-pursuers, decentralized systems are beneficial to avoid single points of failure. Classical algorithms for decentralized multi-agent pursuit often assume omnidirectional pursuers, derive the local interaction rules from simple geometry and do not learn or adapt to evader behavior. For multi-agent teams consisting of wheeled robots or fixed-wing airplanes, the non-holonomic kinematic constraints on the motion also need to be considered.

AncoraSIR.com

SUSTech
Southern University of Science and Technology

How to achieve efficient pursuit of moving targets

Its significance and application to general robot autonomy

- In a specific environment, it can optimize the way of cooperation between multiple robots and improve efficiency. The specific scenario of capturing a finite speed but faster evader with multiple, non-holonomic pursuers in a bounded arena without obstacles.
-
- In the field of military air force, it can optimize the combat efficiency of UAV and other military aircraft and improve the combat capability

AncoraSIR.com

SUSTech
Southern University of Science and Technology

How to achieve efficient pursuit of moving targets

The role of artificial intelligence and machine learning in solving this problem

- Using deep reinforcement learning for pursuing an omnidirectional target with multiple, homogeneous agents that are subject to unicycle kinematic constraints. They use shared experience to train a policy for a given number of pursuers, executed independently by each agent at run-time.

- Most approaches to date did not consider real-world limitations such as local measurements and non-holonomic motion constraints and did not offer a thorough analysis of the system on operational metrics.

AncoraSIR.com

SUSTech
Southern University of Science and Technology

No classical approaches in the literature

Real-world limitations

- Deep Reinforcement Learning (DRL) has also been successfully applied to multi-agent pursuit-evasion, however, most approaches to date did not consider real-world limitations such as local measurements and non-holonomic motion constraints and did not offer a thorough analysis of the system on operational metrics.

AncoraSIR.com

SUSTech
Southern University of Science and Technology

THE PURSUIT-EVASION SCENARIO

Easy to transplant to the real world

Multiple homogeneous, slower pursuers chasing a single, faster target. We consider a trial successful if, at any point during the trial, the distance between the evader and at least one of the pursuers is less than a given collision radius ($d_{i,T} < d_{cap}$). If the target is not captured within a fixed time then the trial is considered unsuccessful. In addition, collision of pursuers are not considered as failure, but this is avoided by defining a value function during training.

$$\dot{x}_i = v\cos\psi_i$$
$$\dot{y}_i = v\sin\psi_i$$
$$\dot{\psi}_i = \omega$$

The motion of the target is unconstrained, but the linear velocity of the pursuers are assumed to be constant and the angular velocity is variable within a certain range.

All pursuers are exactly the same, each pursuer has global information (relative position of other pursuers and target)

AncoraSIR.com

SUSTech
Southern University of Science and Technology

DEEP REINFORCEMENT LEARNING FOR PURSUIT

Adopted algorithm and modeling the problem

As the Deep RL algorithm, use Twin Delayed Deep Deterministic Policy Gradient approach (TD3), which is an improvement over DDPG , designed to reduce the overestimation of the value function.

For each number of agents training a different policy, which results in a total of nmax policies, where nmax is the maximum number of pursuers that been analyzed.

The task for a single pursuer is formulate to be a Markov Decision Process (MDP) defined by tuple $\{S, A, R, P, \gamma\}$ where $s_t \in S$, $a_t \in A$, $r_t \in R$ are state, action and reward observed at time t, P is an unknown transition probability from s_t to s_{t+1} taking a_t, and γ is a discount factor. The DRL goal is to maximise the sum of future rewards

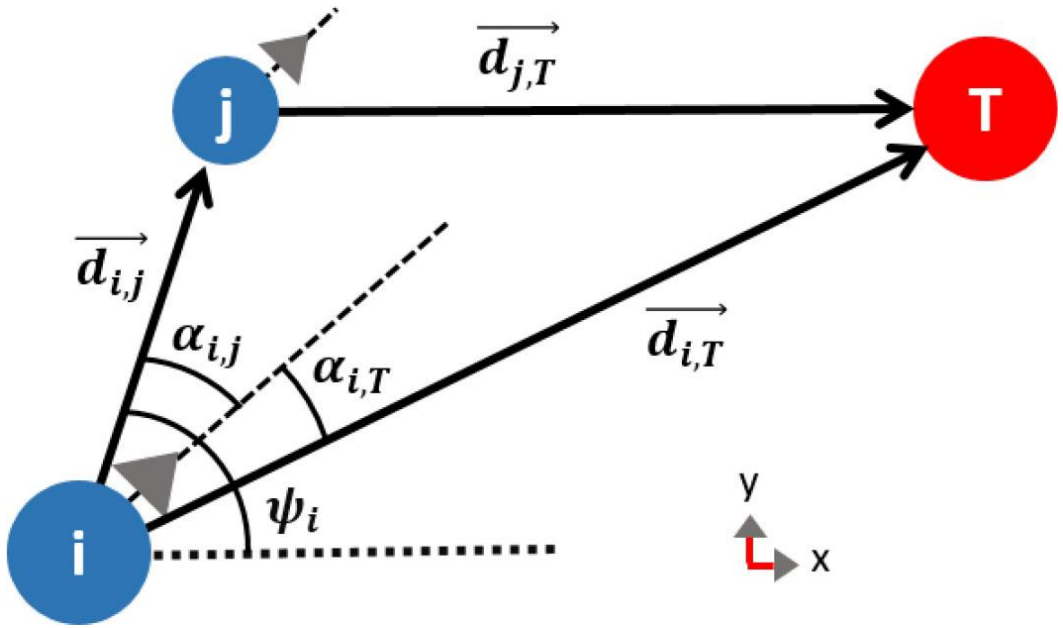
$$R = \sum_{t=0}^{T} \gamma^t r_t$$

AncoraSIR.com

DEEP REINFORCEMENT LEARNING FOR PURSUIT

State representation

The state of a pursuer i, assuming a total of n pursuers, is given by $s_i = [\psi_i, \dot{\psi}_i, s_{i,T}, s_{i,1}, s_{i,2}, .., s_{i,n-1}]$, where ψ is the heading with respect to a fixed world frame, $s_{i,T}$ is the state of the target relative to pursuer i and $s_{i,j}$ is the state of pursuer j relative to pursuer i. (Time indices are dropped for the sake of clarity). The relative state of the target with respect to pursuer i is $s_{i,T} = [d_{i,j}, \dot{d}_{i,T}, \alpha_{i,T}, \dot{\alpha}_{i,T}]$ and the relative state of pursuer j with respect to pursuer i $(i \neq j)$ is $s_{i,j} = [d_{i,j}, \alpha_{i,j}]$, where $d_{i,j}$ is the Euclidean distance between pursuers i and j and $\alpha_{i,j}$ is the heading error defined as the angle between the heading of pursuer i, and the vector between i and j, as shown in in Fig 1. The state representation consists of a total of $2n + 4$ variables, which scales linearly with the number of pursuers n.



As neural networks typically are not permutation invariant when operating on sets, so they assign j values for each observation by sorting each other pursuer with respect to their relative angle α.

AncoraSIR.com

DEEP REINFORCEMENT LEARNING FOR PURSUIT

Reward Structure

At each time step, each agent individually receives a reward designed to incentivize the capture of the evader and encouragea good formation of pursuers.

$$r_i = \begin{cases} r_{captor}, & \text{if } d_{i,T} \leq d_{cap} \\ r_{helper}, & \text{if } d_{j,T} \leq d_{cap}, \exists j \neq i \\ -w_q\, q - w_d\, d_{i,target}, & \text{otherwise} \end{cases}$$

The formation score is a scalar number in the range [0,2], which provides a metric for evaluating the fitness of a formation of the pursuers (lower is better). The formation score(q) is defined as:

$$q = \frac{1}{n} \sum_{i=1}^{n} (\hat{\boldsymbol{d}}_{\mathbf{0T}} \cdot \hat{\boldsymbol{d}}_{\mathbf{iT}} + 1)$$

where $\mathbf{d_{iT}}$ denotes a unit vector pointing in the direction from agent i and the target, the closest agent to the target is defined as agent 0. When the agents are close to the target, the reward is dominated by the formation score; when the agents are far from the target, the reward is dominated by the distance to the target.

If the target is captured at a time step, then the pursuer who captures the evader receives the reward r_{captor}, while the rest of the agents receive r_{helper}, such that $r_{captor} > r_{helper}$.

AncoraSIR.com

DEEP REINFORCEMENT LEARNING FOR PURSUIT

Curriculum Learning

This letter apply a curriculum for learning by starting from an easier version of the task and gradually increasing the difficulty until the actual difficulty is achieved.

The capture radius varys by starting from a large radius, then gradually making it smaller. This encourages agents to not adopt a straightforward chasing tactic at the beginning of learning but to form more sophisticated behaviors, which could be transferred to smaller capture radii.

AncoraSIR.com

SUSTech
Southern University of Science and Technology

Experimental Initial

1. Multi-Agent Deep Reinforcement Learning：

We consider all agents to be homogeneous which allows us to use shared experience to train all agents. This allows the agents to train faster, as well as gathering more information from every step in the environment. All agents are governed with the same policy, however, at each time step the agents use their local observations to individually takeactions, resulting in a decentralized system.

(1) persuer :Timeout = 500 , Vp=10pixels, The number of pursuers n varied between 1 and 8, initialized at random positions within a circular area with a radius of 100 pixels

(2) evader: Ve=0~20pixels initialized at a random position between the arena boundary and an inner circle with a radius of 300 pixels

Fixed Paths: We propose a benchmark where the evader follows three predefined paths, as shown in Fig. 2.

Repulsive: We use a potential field method with repulsive forces only to find a motion vector. Each pursuer exerts a repulsive force in the direction of the vector between the pursuer and the evader. The arena boundary also exerts a force so that the evader can avoid the wall. These forces decrease proportionally to the distance squared. The resultant vector is calculated by:

$$\vec{v} = \sum_j \left(\frac{\vec{a}_j - \vec{e}}{d_j^2} \right) + \frac{\hat{\gamma} R_{arena} - \vec{e}}{d_w^2} \qquad (3)$$

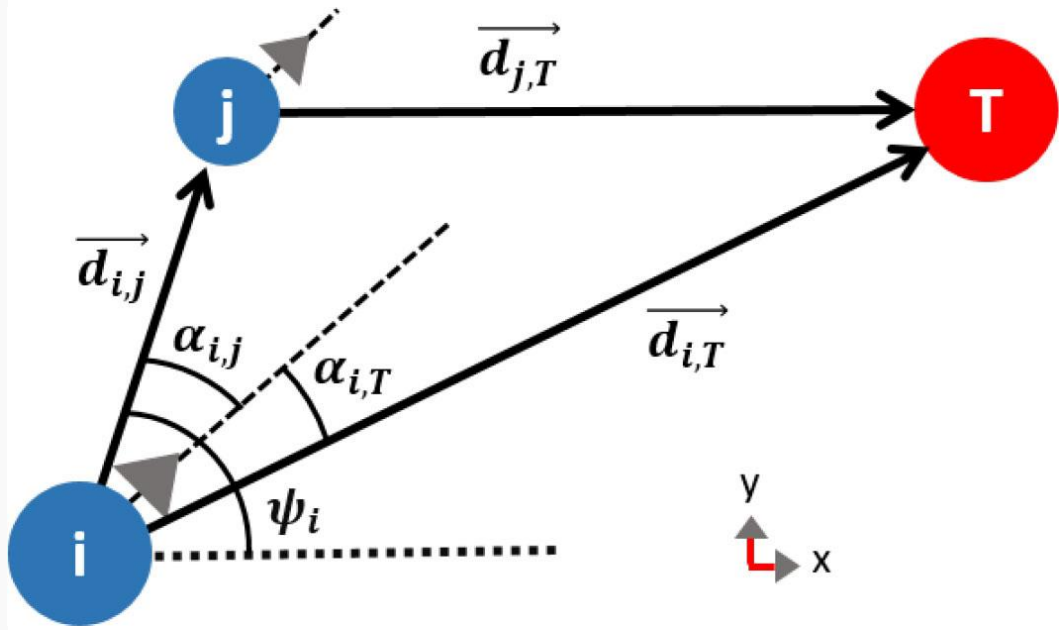
AncoraSIR.com

SUSTech
Southern University of Science and Technology

Experimental Initial

2. state representation:

The state of a pursuer i, assuming a total of n pursuers, is given by $s_i = [\psi_i, \dot{\psi}_i, s_{i,T}, s_{i,1}, s_{i,2}, .., s_{i,n-1}]$, where ψ is the heading with respect to a fixed world frame, $s_{i,T}$ is the state of the target relative to pursuer i and $s_{i,j}$ is the state of pursuer j relative to pursuer i. (Time indices are dropped for the sake of clarity). The relative state of the target with respect to pursuer i is $s_{i,T} = [d_{i,j}, \dot{d}_{i,T}, \alpha_{i,T}, \dot{\alpha}_{i,T}]$ and the relative state of pursuer j with respect to pursuer i $(i \neq j)$ is $s_{i,j} = [d_{i,j}, \alpha_{i,j}]$, where $d_{i,j}$ is the Euclidean distance between pursuers i and j and $\alpha_{i,j}$ is the heading error defined as the angle between the heading of pursuer i, and the vector between i and j, as shown in in Fig 1. The state representation consists of a total of $2n + 4$ variables, which scales linearly with the number of pursuers n.



AncoraSIR.com

Experimental Initial

3. reward structure:

If the target is captured at a time step, then the pursuer who captures the evader receives the reward r_{captor}, while the rest of the agents receive r_{helper}, such that $r_{captor} > r_{helper}$. This encourages each pursuer to go for the final capture while encouraging collaboration.

At each step that the target is not captured, each and every agent receives a negative reward that is a weighted linear combination of an individual reward (its distance to the target $d_{i,target}$) and a group reward (q-score [13], which we call the formation score in our work). The formation score is a scalar number in the range [0,2], which provides a metric for evaluating the fitness of a formation of the pursuers (lower is better). The formation score (q) is defined as:

$$q = \frac{1}{n}\sum_{i=1}^{n}(\hat{\boldsymbol{d}}_{\mathbf{0T}} \cdot \hat{\boldsymbol{d}}_{\mathbf{iT}} + 1) \qquad (2)$$

$$r_i = \begin{cases} r_{captor}, & \text{if } d_{i,T} \le d_{cap} \\ r_{helper}, & \text{if } d_{j,T} \le d_{cap}, \exists j \neq i \\ -w_q\, q - w_d\, d_{i,target}, & \text{otherwise} \end{cases}$$

AncoraSIR.com

Experimental Results(Their)

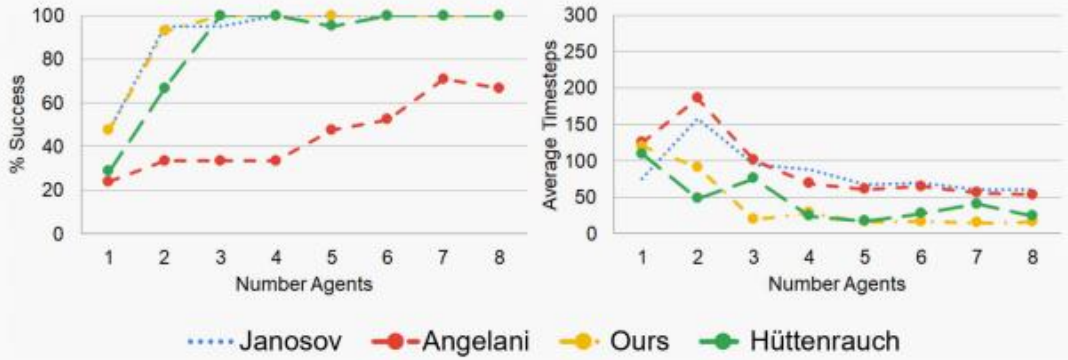


Fig. 3. Success rate and the average timesteps using the fixed paths benchmark, for different number of agents.

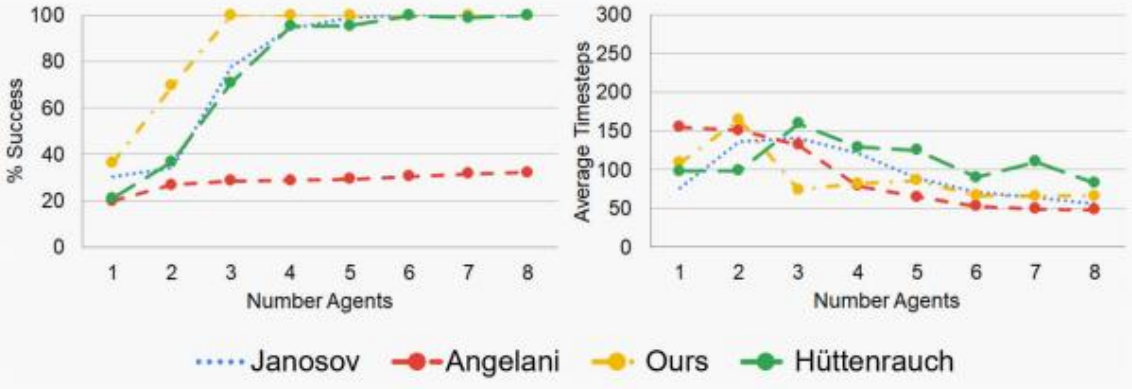


Fig. 4. Success rate and the average timesteps taken for the repulsive evader, for different number of agents.

AncoraSIR.com

Experimental Results(Our)

```python
class Actor(nn.Module):
    def __init__(self, input_size, output_size):
        super(Actor, self).__init__()
        self.fc1 = nn.Linear(input_size, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, outp

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        return torch.tanh(self.fc3(x)


class Critic(nn.Module):
    def __init__(self, input_size):
        super(Critic, self).__init__(
        self.fc1 = nn.Linear(input_si
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        return self.fc3(x)
```

```python
class PEEnv:
    def __init__(self, size=5, speed=0.2, radius=0.4):
        self.size = size
        self.speed = speed
        self.radius = radius
        self.reset()

    def reset(self):
        # self.evader = np.random.rand(2) * self.size
        self.evader = np.array([2.5, 2.5])
        # self.pursuers = np.random.rand(2, 2) * self.size
        self.pursuers = np.array([[4.5,4.5],[1.0,1.0]])
        self.wd=0.1
        self.wq=0.1
        self.captor=1
        self.helper=0.8
        self.capture=-1
        return self.get_observation()
```

AncoraSIR.com

SUSTech
Southern University of Science and Technology

Experimental Results(Our)

```
def reward(self,capture):
    rewards=[]
    if capture == -1:
        if len(self.pursuers)==1:
            reward=-np
            rewards.ap
            return rew
        else:
            mindis=0
            minobj=-1
            for i in r
                dis=np
                if dis
                    mi
                    mi
            q=0
            for i in r
                dis=np
                q+=min
            q=q/len(se
            for i in r
                reward=
                rewards
    if capture != -1:
        if len(self.pu
            rewards.append(self.captor)
            return rewards
        else:
            for i in range (len(self.pursuers)):
                if i == capture:
                    reward=self.captor
                    rewards.append(reward)
                else: rewards.append(self.helper)
    return rewards
```

```
    def step(self, actions):
        for i, action in enumerate(actions):
            action = action.squeeze()  # Add this line to remove the extra dimension
            self.pursuers[i] = self._apply_action(self.pursuers[i], action)
            self.pursuers[i] = np.clip(self.pursuers[i], 0, self.size)

        self.evader = self._apply_action(self.evader, np.random.uniform(-1, 1, 2))
        self.evader = np.clip(self.evader, 0, self.size)

        #rewards = [-np.linalg.norm(pursuer - self.evader) for pursuer in self.pursuers]
        done = any(np.linalg.norm(pursuer - self.evader) <= self.radius for pursuer in self.pursuers)
        if done:
            for i in range(len(self.pursuers)):
                if np.linalg.norm(self.pursuers[i] - self.evader) <= self.radius:
                    self.capture=i
        rewards=self.reward(self.capture)
        return self.get_observation(), rewards, done
```

An

Pres

SUSTech
Southern University of Science and Technology

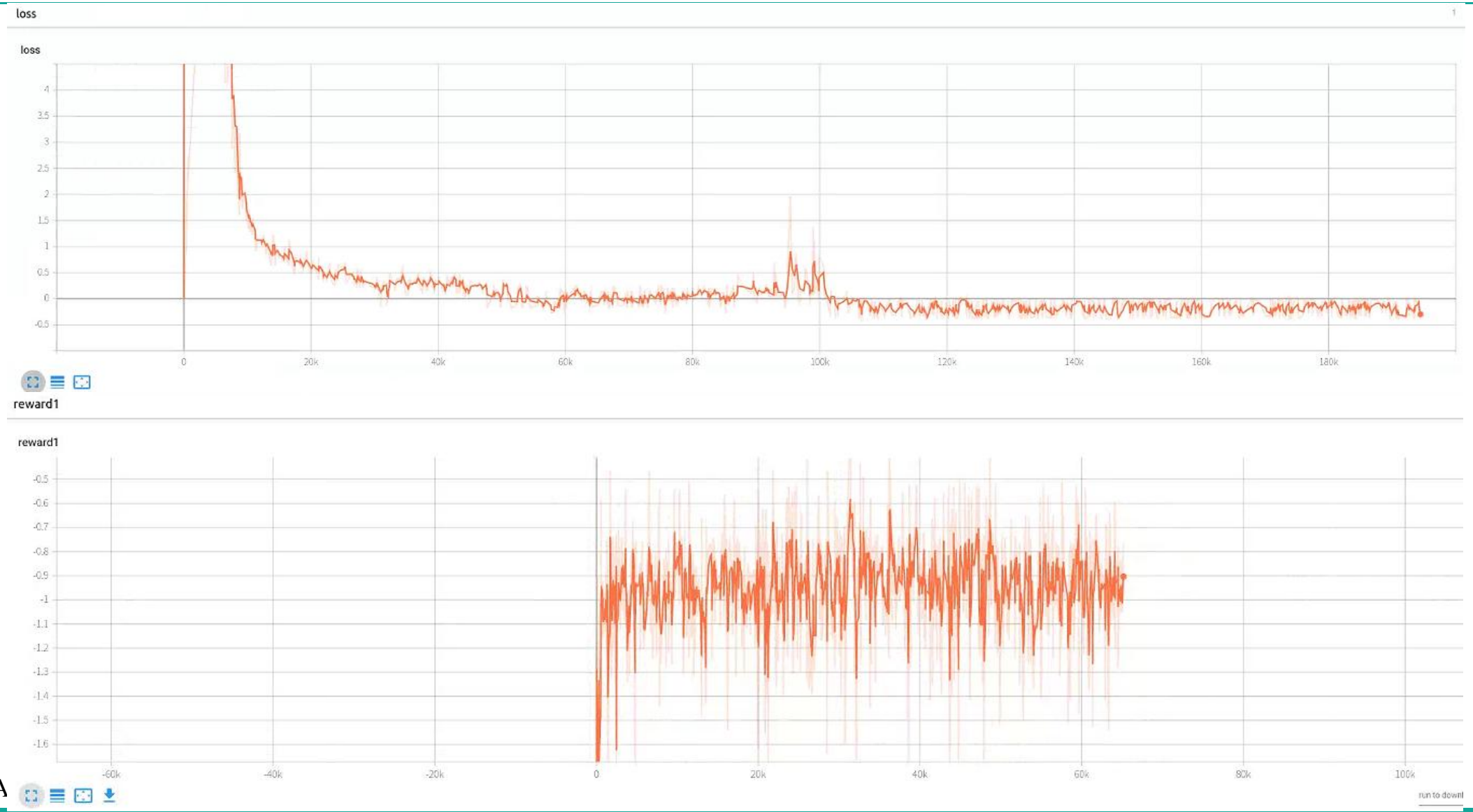
Experimental Results(Our)

```
def train():
    env = PEEnv()
    agents def run_episode(env, agents, train=True):
                obs = env.re
    n_episo     episode_traj
    best_re     episode_rewa
    best_ti     itera=1
    for epi     done = False
        epi     while not do
                    actions
        if          next_obs
                    loss=0
                    if train
                        for
        if
                    writer.a
        if          obs = ne
                    episode_
        wri         episode_
        wri         itera+=1
    plot_ti         if itera
                        retu

                return episo
```

```
def plot_trajectory(trajectory):
    trajectory = np.array(trajectory)
    capture_radius = 0.4  # Assuming the capture distance is equal to the speed

    plt.figure(figsize=(6, 6))
    plt.xlim(0, 5)
    plt.ylim(0, 5)
    plt.title("Best Trajectory")
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.grid()

    plt.plot(trajectory[:, 0, 0], trajectory[:, 0, 1], 'ro-', label="Pursuer 1")
    plt.plot(trajectory[:, 1, 0], trajectory[:, 1, 1], 'bo-', label="Pursuer 2")
    plt.plot(trajectory[:, 0, 2], trajectory[:, 0, 3], 'g*-', label="Evader")

    # Draw capture regions for each predator at the final position
    for predator in trajectory[-1, :2]:
        circle = plt.Circle(predator, capture_radius, color='gray', alpha=0.3)
        plt.gca().add_patch(circle)

    plt.legend()
    plt.show()
```

AncoraSIR.com

Experimental Results(Our)



A

Discussion of Results

Our results show that multi-agent pursuit benefits from curriculum learning and a reward based on agent formation, which we borrowed from the group-pursuit literature

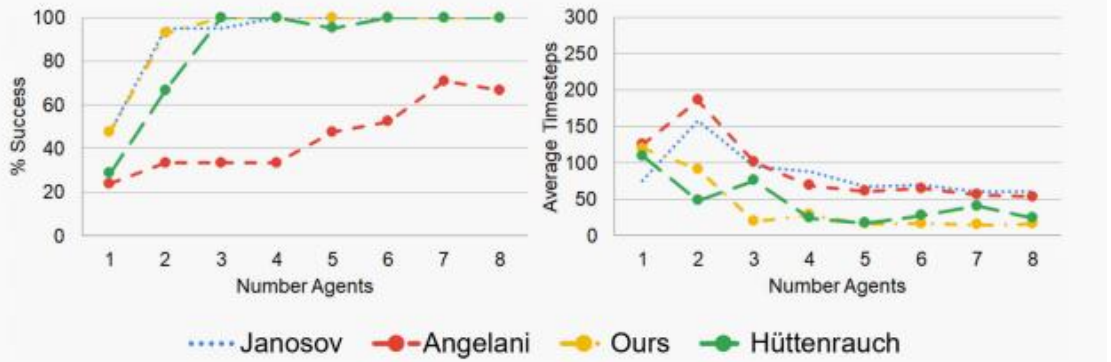


Fig. 3. Success rate and the average timesteps using the fixed paths benchmark, for different number of agents.

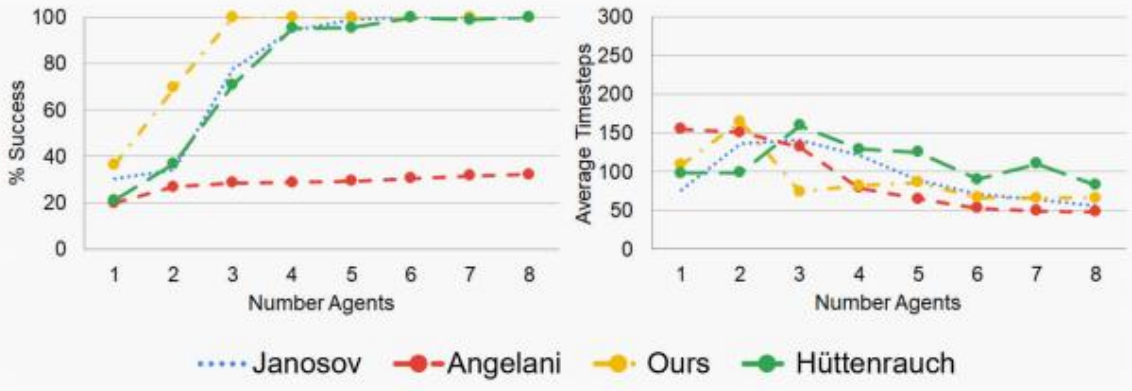


Fig. 4. Success rate and the average timesteps taken for the repulsive evader, for different number of agents.

AncoraSIR.com

Critique / Limitations / Open Issues

Further explaination of the title with supporting evidence

1、The key limitations of the proposed approach should be the limitation of the current work related with the need to train a network for each number of observable agents. Yet this can be partially mitigated by using the same network and fixing the number of observable pursuers.

2、The current application is restricted to 2D motion because of the existance of representational singularities

3、As we have mentioned before, most approaches to date did not consider real-world limitations suchas local measurements and non-holonomic motion constraintsand did not offer a thorough analysis of the system on operationalmetrics.

AncoraSIR.com

SUSTech
Southern University of Science and Technology

Future Work for Paper / Reading

Further explanation of the title with supporting evidence

The possibility of realizing fixed-size state representation for neighboring agents

Maybe by using deep sets , mean embeddings or making use of Graph Neural Networks

Others :

1、Exploring scenarios with numerous evaders

2、Integrating smarter evader strategies by using the idea of safe-reachability

3、Implementing the evader as an RL agent and training both the evader and pursuer simultaneously

AncoraSIR.com

SUSTech
Southern University of Science and Technology

Extended Readings

Further explaination of the title with supporting evidence

1、M. Hüttenrauch, A. Šoši´c, and G. Neumann, "Deep reinforcement learning for swarm systems," J. Mach. Learn. Res., vol. 20, no. 54, pp. 1–31, 2019.

2、M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov,and A. J. Smola, "Deep sets," in Proc. Adv. Neural Inf. Process. Syst.,2017, pp. 3391–3401.

3、J. Zhou et al., "Graph neural networks: A review of methods and applica_x0002_tions," 2019, arXiv:1812.08434.

4、A. Pierson, Z. Wang, and M. Schwager, "Intercepting rogue robots: An algorithm for capturing multiple evaders with multiple pursuers," IEEE Trans. Robot. Autom. Lett., vol. 2, no. 2, pp. 530–537, 2017.

5、Z. Zhou, W. Zhang, J. Ding, H. Huang, D. M. Stipanovi´c, and C. J.Tomlin, "Cooperative pursuit with voronoi partitions," Automatica, vol. 72, pp. 64–72, 2016.

AncoraSIR.com

SUSTech
Southern University of Science and Technology

Summary

Further explaination of the title with supporting evidence

Problem the reading is discussing

- The specific meaning, related principles, applications and prospects of DRL method

Why is it important and hard

- Simulation experiments show that our approach, applied to non-holonomic agents, outperforms the state-of-the-art in heuristic multi-agent pursuit methods and a recent DRL based approach

What is the key limitation of prior work

- The key limitations of the proposed approach should be the limitation of the current work related with the need to train a network for each number of observable agents

AncoraSIR.com

SUSTech
Southern University of Science and Technology