2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 27 - October 1, 2021. Prague, Czech Republic

Scalable Reinforcement Learning Policies for Multi-Agent Control

Christopher D. Hsu, Heejin Jeong, George J. Pappas, and Pratik Chaudhari

2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 978-1-6654-1714-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/IROS51168.2021.9636344

***Abstract*— We develop a Multi-Agent Reinforcement Learning (MARL) method to learn scalable control policies for target tracking. Our method can handle an arbitrary number of pursuers and targets; we show results for tasks consisting up to 1000 pursuers tracking 1000 targets. We use a decentralized, partially-observable Markov Decision Process framework to model pursuers as agents receiving partial observations (range and bearing) about targets which move using fixed, unknown policies. An attention mechanism is used to parameterize the value function of the agents; this mechanism allows us to handle an arbitrary number of targets. Entropy-regularized off-policy RL methods are used to train a stochastic policy, and we discuss how it enables a hedging behavior between pursuers that leads to a weak form of cooperation in spite of completely decentralized control execution. We further develop a masking heuristic that allows training on smaller problems with few pursuers-targets and execution on much larger problems. Thorough simulation experiments and comparisons to state of the art algorithms are performed to study the scalability of the approach and robustness of performance to varying numbers of agents and targets.**

I. Introduction

This paper studies a multi-agent control problem where a team of pursuers tracks multiple targets. This problem has a wide variety of applications that range from defense [1], [2], search and rescue [3], to environmental monitoring [4]. Our work is motivated by the following challenges in this problem. First, the number of targets, or even the number of pursuers may not be known a priori. In particular, for learning-based approaches which the present paper focuses on, the number of agents at deployment can be different (typically *larger*) than that at training time. Second, control polices should ideally be decentralized to enable execution without communication overheads and latency. However, a trade-off in this setting is that it is difficult to design controllers that can benefit from cooperative strategies to improve performance. Our approach to resolving these challenges is based on the following key ideas.

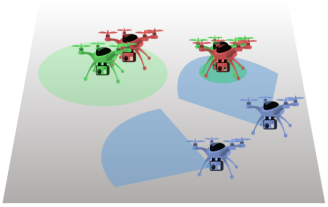


Fig. 1: Multi-agent pursuer, multi-target tracking task: a team of blue agents with limited range and bearing sensing (blue cones) track red targets; the belief of target states, shown in green, is computed using the observation history of the targets.

1. Off-policy Reinforcement Learning for multi-agent control problems. We model pursuers as agents that observe the distance and bearing of targets within their sensing radius; their goal is to localize target states while targets follow a fixed, but unknown policy. We use entropy-regularized off-policy reinforcement learning (RL) to learn the pursuers' value function which aims to minimize the uncertainty of the targets' states maintained by the pursuers' belief system. For the large-scale multi-agent problems we are interested in, continuous-control policies are challenging to learn. Therefore, we have the pursuers choose actions from a discrete set of motion primitives; our experiments indicate that discrete actions are enough to learn performant policies.

2. Building policies that can handle an arbitrary number of pursuer agents and targets. Each pursuer follows a control policy with the goal of maintaining a Gaussian belief distribution for each target; pursuers receive noisy measurements about target locations when targets are within its sensing radius. Given this setting, it is desirable for the RL-based policy to not be limited by a specific number of targets or pursuer agents. We achieve this by parameterizing a value function using a self-attention-based neural architecture and sharing it amongst agents. This architecture has two powerful properties: (i) it creates a permutation-invariant embedding of the input state-vector (the beliefs) which makes the value function invariant to the ordering of the targets in the pursuer's vicinity, and (ii) it can handle an input vector of an *arbitrary* size, which provides the ability of deploying the trained policy on larger problems, with many more targets. Since execution is decentralized across pursuers, the approach is naturally deployable with different numbers of pursuer agents. Furthermore, our approach can seamlessly handle tasks where the number of agents or targets changes during deployment.

3. Using stochastic policies to obtain a weak form of cooperation. Centralized planning/execution and communication are bottlenecks for cooperation in large-scale multi-agent problems. We develop an effective workaround for this issue by training a *stochastic* RL policy in centralized fashion but executing it in decentralized fashion; neither local observations nor control actions are shared amongst pursuers during deployment. This paradigm forces multiple pursuers to *hedge* their control actions. For example, two pursuers have two targets in their vicinity, since the policy is trained to minimize the average uncertainty across all targets, instead of both non-communicating pursuers heading towards the closest target, entropic regularization in the policy encourages them to also seek the further target with a small probability. This weak, but extremely useful, form of cooperation improves tracking performance as compared to deterministic policies.

4. Heuristics for Scalability. Training large multi-agent policies is difficult: we observe that even state of the art sample-efficient off-policy RL algorithms fail to learn performant policies in the MARL setting. We develop a heuristic that takes a policy trained on a smaller problem, say 4 agents and 4 targets, and executes this policy on a larger problem by masking far away targets. We show that this masking is particularly effective; we can execute policies trained on the smaller problem, with minimal degradation, on problems as large as 1000 pursuers tracking 1000 targets. This heuristic allows us to address a dramatic distribution shift in the number of targets: it essentially scales down the large and

Department of Electrical & Systems Engineering and the GRASP Laboratory at the University of Pennsylvania. chsu8@seas.upenn.edu, heejinj@alumni.upenn.edu, pappasg@seas.upenn.edu, pratikac@seas.upenn.edu

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 15:03:28 UTC from IEEE Xplore. Restrictions apply.

complex observation space at test time into something that is close to what the pursuer policy was trained for. Experiments show that this heuristic performs much better than state of the art MARL algorithms for large problems.

II. Problem Formulation

Consider the multi-agent multi-target setup where the goal of a team of N homogeneous pursuers is to acquire information about M homogeneous targets. The i^{th} agent with state $x_t^i \in \mathbb{R}^2 \times SO(2)$, and the j^{th} target with state $y_t^j \in \mathbb{R}^2 \times SO(2)$, follow a discrete-time dynamical model. Given the initial state for the pursuer $x_0^i \in \mathcal{X}$, an initial target state y_0^j and a horizon T, the pursuer agent chooses control actions u_t^i to maximize the mutual information between y_t^j and the measurement history denoted by $z_{1:t}$. Mutual information between the i^{th} pursuer and the j^{th} target $I(y_t^j, z_{1:t}) = \mathrm{KL}\big(p(y_t^j, z_{1:t}),\ p(y_t^j)p(z_{1:t})\big)$ is the Kullback-Leibler (KL) divergence $\mathrm{KL}(q,p) = \int dq \log(q/p)$ of the joint distribution and the product of the marginals. The objective for localizing target states is

$$\max_{u(\cdot)} \sum_{i=1}^{N} \sum_{t=0}^{T} \sum_{j=1}^{M} I(y_t^j, z_{1:t} \mid x_{1:t}^i) \tag{1}$$

such that $x_{t+1}^i = f(x_t^i, u_t^i)$, $y_{t+1}^j = g(y_t^j)$ and $z_t = h(x_t^i, y_t^1, \ldots, y_t^M)$ and $u_t^i = u(z_{1:t})$ for $t \in \{0, T\}$. The functions $f(\cdot)$ and $g(\cdot)$ are the dynamical models of the pursuers and the targets respectively, and $h(\cdot)$ is the observation model. Note that the measurement history $z_{1:t}$ is shared among the agents. Further decentralization can occur by performing distributed information filtering [5].

A. Multi-agent Formulation

We consider the decentralized partially observable multi-agent setting where pursuers cannot observe the full state of the environment. Each agent receives observations of targets in its vicinity. Therefore, the task is defined as a decentralized partially observable Markov Decision Process (Dec-POMDP) represented by the tuple, $(G, S, A_i, \mathcal{Z}_i, P, R, \gamma)$, where G is a set of agents, S is a set a states, A_i is a discrete set of actions for each agent i, $\mathcal{Z}_i$ is a set of observations for each agent i, and P, R, γ are the state transition probability kernel, the global reward, and the discount factor, respectively. In this work, we consider problems where G, S, and $\mathcal{Z}$ can be infinite to account for scalable teams in a continuous domain. Optimally solving Dec-POMDPs is a hard combinatorial problem that involves searching through all the agent's histories. This process results in an exponentially growing policy space as the time horizon increases, making the policy search quickly intractable [6]. The following simplifying assumptions are made to keep the problem tractable.

1) Partial Observability: In the partially observable setting, the agent does not have access to the ground truth target states. As a substitute, the agent maintains a belief on target states and solves a continuous belief MDP [7]. We denote such a belief distribution for the j^{th} target as $B(y_{j,t}) = p(y_{j,t}|z_{1:t}, x_{1:t})$ and its predicted distribution for the subsequent step as $\bar{B}(y_{j,t+1}) = p(y_{j,t+1}|z_{1:t}, x_{1:t})$ [8]. Assuming y_{t+1} is independent of $x_{1:t}$, the optimization problem can be reduced to minimizing the cumulative differential entropy, $H(y_{t+1}|z_{1:t}, x_{1:t})$ [9]. When the belief is Gaussian, $B(y_{j,t}) = \mathcal{N}(\hat{y}_{j,t}, \Sigma_{j,t})$,

$$H(y_{t+1}|z_{1:t}, x_{1:t}) = \frac{1}{2} \log\left((2\pi e)^M \det(\Sigma_t)\right) \tag{2}$$

the optimization problem (1) becomes:

$$\min_{\pi} \sum_{t=0}^{T} \log\det(\Sigma_t) \tag{3}$$

At each time step t, the agent at state x_t, chooses a control input u_t from the policy π, based on the predicted belief of the target, $\bar{B}(y_{t+1})$. In order to maximize the objective (1), the agent receives a measurement z_{t+1} from the sensor. At the same time, the true target state transitions from $y_t \to y_{t+1}$ and if it is observed by the agent, the corresponding belief distribution is updated with the new measurement. This process is repeated until the time horizon T. In the case of multiple agents or multiple targets, at each time step, all agents and targets will transition before the environment transitions from $t \to t+1$.

2) Decentralization: Consider a team of N agents pursuing M targets where each agent can make control actions independently of its team. A tractable solution to this problem is through the use of the centralized training, decentralized execution parameter sharing approach [10]. Assuming that the agents are homogeneous, during centralized training, agents share the parameters of a single value network and an experience replay buffer. This formulation reduces the policy space that the algorithm has to search through because it optimizes a single, shared policy rather than multiple individual policies [11]. This approach also mitigates the non-stationarity issue in multi-agent training by exposing the policy to all of the agents' experiences simultaneously [12]. A shared policy also alleviates computational issues in multi-agent centralized critic algorithms that suffer from the curse of dimensionality problem as the number of agents increases [13]. A parameter-shared policy can be executed in decentralized fashion at test time for *any* number of agents.

B. Off-policy Reinforcement Learning

In reinforcement learning (RL), the goal is to learn the action-value function which is defined as the expected reward obtained using the policy π, from an initial state x, with control action a. Off-policy RL is a popular technique to estimate the action-value function. It minimizes the expectation of the 1-step temporal difference (TD) error. Off-policy methods are named so because they maintain a dataset $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})\}_{t=0,\ldots}$ collected using a (behavior) policy and estimate the value function of the current policy π using this data. We will parametrize the value function using a deep network with parameters θ. The optimal parameters θ^* can be found by performing stochastic gradient descent to minimize the Huber loss applied to the TD error. In the sequel, we design a simulation environment for multi-agent problems which allows us to study complex multi-agent interactions across diverse scenarios in the target tracking problem. This simulator is used to train our off-policy RL methods. We next discuss RL-specific details of this simulation setup.

1) Observations: In the partially observable target-tracking task, exact target states are not known to the agents. Each agent maintains a belief over target states [8], [14] that is updated using a Kalman filter as new observations about the target are received. The dynamics of targets is unknown to the tracking agent, and more importantly, target states are independent of the agent's control input. The value function and the control policy of the i^{th} agent therefore take the predicted belief of target j at the *next* time instant $t+1$ as the input at time t. This is neat way to ensure that the RL policy

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 15:03:28 UTC from IEEE Xplore. Restrictions apply.

has access to the target tracking information maintained in the Kalman filter without having to relearn it from scratch, which would require a large amount of data. We also transform the belief of the j^{th} target in agent i's local frame ($\hat{y}^{(x_i)}$), this ensures that there is no distribution shift between observations collected by different agents. The state is

$$s_j := \big[\hat{y}^{(x_i)}_{j,r}, \hat{y}^{(x_i)}_{j,\theta}, \hat{\dot{y}}^{(x_i)}_{j,r}, \hat{\dot{y}}^{(x_i)}_{j,\theta}, \log\det\Sigma_j, \mathbb{I}(y_j \in \mathcal{O}(x_t))\big].$$

The quantities $\hat{y}^{(x_i)}_{j,r}$ and $\hat{y}^{(x_i)}_{j,\theta}$ denote polar coordinates of the mean of target j's belief. We also use their derivatives as a part of the state. Target belief covariance, Σ_j, is represented with the differential entropy formulation (3). We model agents with a limited sensing range using the Boolean function $\mathbb{I}(\cdot)$: it returns 1 if the true target state is in the vicinity of the agent, and zero otherwise. The combined state denoted by s_t of M targets is the concatenation of the states of individual targets, and it is a vector of $6M$ entries.

We are motivated by the realization that if one were to use a multi-layer perceptron (MLP) for parameterizing the value function, the number of targets M would have to be fixed between training and testing. Further, and this is much more debilitating, the order in which states of individual targets are concatenated together would also have to be fixed between training and testing. Ideally, we would rather train on smaller problems with few agents where it is easy to gather rich data from the simulation and execute the same policy on problems with more agents at test time. We therefore think of the observations as a set of M elements and design an embedding of this set as the state for the RL policy, (i) that is permutation-invariant, i.e., it does not depend upon the order in which observations are concatenated together, and (ii) that can handle an arbitrary number of targets. Section III-A discusses how a self-attention-based architecture is used to achieve these desiderata.

2) Rewards: We would like to find the optimal policy π^* that maximizes the cumulative mutual information (1) between the i^{th} pursuer and the j^{th} target, this amounts to minimizing the objective (3). We approximate the objective as a discounted sum by defining the reward as the average uncertainty over all the targets such that $R(s_t, a_t) = -\frac{1}{M}\sum_{j=0}^{M}\log\det(\Sigma^j_{t+1})$, resulting in a value function:

$$V^{\pi}(s) = -\mathbb{E}_{\pi}\Big[\frac{1}{M}\sum_{t=0}^{T-1}\sum_{j=0}^{M}\gamma^t \log\det(\Sigma^j_{t+1}) \mid s_0 = s\Big] \qquad (4)$$

Observe that the reward given to each agent depends upon the performance of all the other agents, this is desirable because the value function and the policy are shared between agents and also because doing so makes the policy π^* responsive to how the other agents undertaking the same policy are tracking their respective targets.

3) Discrete Action Space: Continuous-control RL policies are challenging to learn in multi-agent settings. Our goal is to understand the multi-agent aspects of the problem and in order to so, we choose a discrete action space using motion primitives: $A = \big\{(v,\omega) | v \in \{0, 0.67, 1.33, 2.0\}[m/s], \omega \in \{-\pi/4, 0, \pi/4\}[\text{rad}/s]\big\}$. This set of actions is rich enough to lead to performant tracking policies, and yet, it is small enough for us to be able to learn RL policies for large, challenging problems.

III. Approach

A. Self-Attention-based Model Architecture

We require a policy that can be trained on smaller problems and executed at test time on larger problems with more targets in order to improve sample efficiency and alleviate debilitating restrictions such as requiring specific input ordering [15]. Therefore, for this setting, we think of the observations as a set of elements that is fed into an embedding (i) that is permutation-invariant, i.e., the embedding is invariant to changes in the ordering of observations, and (ii) that can handle an arbitrary number of elements in the observation set. In this work, the policy is built with an encoder-decoder style model architecture as described by DeepSets [16], filled with self-attention layers [17]. Self-attention layers provide the desired property of permutation-invariance by highlighting the actionable information in the observation [18]. The second property of handling an arbitrary number of elements is taken care of by the summation in the DeepSets architecture, i.e., any additional elements are simply part of the linear combination of the embedding space.

1) Permutation input representations and embeddings of sets: DeepSets [16] show that a function $\phi(A)$ operating on a set A is invariant to permutation of the elements in A iff it can be decomposed as $\phi(A) \equiv \rho(\sum_{a\in A}\psi(a))$ for some functions ψ and ρ. Both ψ and ρ can be learnt to build invariance and the summation enables $\phi(A)$ to handle sets with varying number of elements.

2) Attention: The self-attention mechanism [17] is a powerful way for the value function to learn to pay attention to parts of the input that are more relevant to the output. Through the structure of learning embeddings, attention amounts to mapping a query, Q, and a set of key, K, value, V pairs to an output. Structured as a linear combination of V, attention is able to learn these embeddings by computing the dot product QK^T which measures how compatible they are. The attention module $\omega(d_q^{-\frac{1}{2}}QK^T)V$ gives more weight to the key that has a large dot product with the query vector, i.e., the output is computed as a weighted sum of the values, a measure of the compatibility between the query and the corresponding key. ω is an activation function, e.g. softmax, and $\frac{1}{\sqrt{d_q}}$ is a scaling factor. Finally, the self-attention module can be improved by considering higher order interactions of Q and K by projecting the inner product across multiple sub-spaces, creating the Multi-Head Attention Block (MAB).

B. Maximum entropy policies

In order to learn a stochastic policy, the maximum entropy-regularized objective [19] looks to balance maximizing the expected return and entropy, i.e, to be successful at the task while acting as randomly as possible. The entropy of the policy is controlled by a temperature parameter, α, that reflects the idea that high entropy equates to high temperature and high randomness and vice versa. The entropy H of a distribution P over a random variable x is given as:

$$H(P) = \mathbb{E}_{x\sim P}[-\log P(x)] \qquad (5)$$

At each time step, the agent gets a bonus reward proportional to the entropy of the policy. The RL problem can be reformulated with a maximum entropy objective:

$$J(\pi) = \sum_{t=0}^{T-1}\mathbb{E}_{(s_t,a_t)\sim\pi}[r(s_t,a_t) - \alpha\log\pi(a_t|s_t)] \qquad (6)$$

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 15:03:28 UTC from IEEE Xplore. Restrictions apply.

This objective favors stochastic policies by augmenting the equation with an expected entropy over the policy. Stochastic policies have conceptual and practical advantages. First, the policy is incentivized to explore more widely, while giving up on unpromising trajectories. Secondly, the policy can capture multiple modes of optimal behavior, i.e., when there are equally attractive actions, the policy places an equal probability mass to each of those actions. Lastly, these stochastic policies have shown to substantially improve exploration and are therefore beneficial for cooperative tasks.

C. Details of the off-policy training implementation

We utilize and augment the state-of-the-art off-policy method Clipped Double Q-Learning [20] to provide the structure in which we learn the deterministic and stochastic policies. The original Double Q-Learning algorithm learns two independent estimates of the true Q value parameterized by neural networks denoted as Q_{ϕ_1} and Q_{ϕ_2}. This framework is able to reduce overestimation of the Q value by taking the minimum value of the two independent estimates. The author suggests that this method provides higher values to states that have lower variance estimation error, leading to more stable learning. The double Q-learning update is highlighted below.

$$Q^*(s,a) \leftarrow r + \gamma \min_{k=1,2} Q_{\phi'_k}(s', \arg\max_{a'} \sum_{k=1,2} Q_{\phi_k}(s',a')) \quad (7)$$

Double Q-learning learns a deterministic policy due to the action being chosen by taking the $\arg\max(Q(s,a))$. Standard deterministic policies utilize an ϵ-greedy exploration schedule during training.

To learn a stochastic policy, we augment the standard double Q-learning formulation with an entropy-regularized objective [21]. This algorithm (see Alg. 1) is denoted as Soft Double Q-learning and balances the expected return and entropy during exploration. The soft double Q-learning update is

$$Q^*(s,a) \leftarrow r + \gamma \min_{k=1,2} \mathbb{E}_{a'\sim\pi} \left[Q_{\phi'_k}(s',a') - \alpha \log \pi(a'|s') \right] \quad (8)$$

With a stochastic policy, actions are chosen by sampling from a multinomial defined by the log softmax of the Q function values. During exploration, a larger entropy is used to encourage the policy to sample actions that might not necessarily have the largest value. This helps the policy learn about less frequented parts of the state space thereby improving exploration. Other implementations of maximum entropy objectives have used fixed or learned schedules to control α in order to promote exploration vs. exploitation.

D. Algorithm

We adjust the standard single-agent learning algorithm, Soft Double Q-learning, with the parameter sharing approach [10] to provide the benefits of centralized training with decentralized execution for multi-agent learning. In Alg. 1, the addition of parameter sharing to Soft Double Q-learning can be seen in the for loop starting in line 8, where we are sampling the environment with each agent. For each environment step, we feed each of the agent's local observations through the shared networks to produce a unique experience to be stored. Only when all the agent's have taken their action with the current policy, does the full environment transition from $t \rightarrow t+1$. The second half of the algorithm, starting line 14, updates the value functions with stochastic gradient descent. We sample from a replay buffer that includes the experiences from all the agents. This procedure helps to nullify the non-stationarity problem of decentrally trained systems. When training is complete, each of the agents receives a copy of the learned policy that can be used to execute decentrally.

In order to improve generalization, n number of agents in the current team and m number of targets are sampled uniformly randomly during training to create different numbers of cooperative agents in the team as seen in line 6. This has two benefits. First, it shows flexibility of the algorithm as n and m fluctuate during training. Second, it helps sample diverse training episodes and expands the task space, resulting in a more generalizable and robust policy.

Algorithm 1 Parameter-Shared Soft Double Q-Learning

1: Initialize value networks Q_{ϕ_1}, Q_{ϕ_2}
2: Initialize target networks $\phi'_1 \leftarrow \phi_1, \phi'_2 \leftarrow \phi_2$
3: Initialize N and M, max number of agents and targets
4: Initialize replay buffer $\mathbb{G}$
5: **for** each iteration **do**
6: Randomly sample $n \in [1, N]$ and $m \in [1, M]$
7: **for** each environment step **do**
8: **for** $i \leftarrow 0, 1, \ldots n$ **do**
9: Observe state s_t and sample $a_t \sim \pi(a_t|s_t)$
10: Execute a_i; observe next state s'_i, reward r_t
11: Store (s_i, a_i, r_t, s'_i) in replay buffer $\mathbb{G}$
12: **end for**
13: **end for**
14: **for** each update step **do**
15: Sample mini-batch of N:
16: $g_t = (s, a, r, s') \sim \mathbb{G}$
17: Compute soft target Q value using 8
18: Perform clipped gradient descent step on
19: $\sum_{k=1,2} \text{huber}(Q_{\phi_k}(s,a) - Q_{\text{targ}}(s,a))$
20: Update target network parameters:
21: $\phi'_k \leftarrow \tau\phi_k + (1-\tau)\phi'_k$
22: **end for**
23: **end for**

E. Heuristic Mask

In this paper, the heuristic mask is used during evaluation to improve scalability of trained policies. At every time step, each agent's observation state is masked to expose the k nearest target beliefs based on its range measurement, $\hat{y}_{j,r}$.

Algorithm 2 Heuristic Mask for Scalability

for each environment step **do**
  for $i \leftarrow 0, 1, \ldots, n$ **do**
    $s_t \leftarrow \text{mask}_{k,\hat{y}_{j,r}}\{s_t(\hat{y}_{1,r}, \ldots, \hat{y}_{M,r})\}$
  end for
end for

IV. Experimental Validation

A. Setup

The multi-agent multi-target tracking task results in a highly stochastic environment. We randomly initialize agent, target, and belief locations. Targets move about randomly with their fixed double integrator with Gaussian noise model. The belief model and the observation model also have noise components, compounding the randomness of the environment. With this in mind, to lower training variance and focus on tracking rather than a search task, we define a restricted random initialization for target locations and their corresponding beliefs. All the agents' positions, $x_{0,\ldots,N}$, are randomly initialized within the given map, avoiding other teammates. The targets' positional component of $y_{0,\ldots,M}$ are randomly initialized 5-10 meters away from an uniformly randomly sampled agent. The targets' initial velocity components are set to 0.0. The Gaussian belief of the targets' locations are initialized 0-5 meters away from their respective target with covariance Σ. Both agents and targets have a maximum velocity of 2 m/s. The time horizon T is 200 steps. Finally, each agent has an observation range

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 15:03:28 UTC from IEEE Xplore. Restrictions apply.

of $10m$ and bearing of $\frac{\pi}{4}$ for a total of $25\pi m^2$ of coverage. See [14] for more details of the target tracking environment, including agent, observation, and target models.

1) Framework: The framework shown in Fig. 2, expands upon a single agent strategy [8] with the use of parameter sharing for multiple agents.

2) Evaluation methodology: In multi-agent multi-target domains, the balance between the number of agents and the number of targets is critical to the behaviors. We will train on the tasks defined by $n \in [1, N]$ agents and $m \in [1, M]$ targets and evaluate, without retraining, on other task spaces with or without a heuristic mask. To denote the difference between policies, for example, a deterministic self-attention-based policy trained on the task space of $[1, 4]$ agents and $[1, 4]$ targets will be "det-4a4t" and a stochastic self-attention-based policy trained on the same task space will be "stoch-4a4t". When evaluating policies on larger task spaces, we also scale the map to fit the density of the originally trained 4a4t task where the empty map has an area of $2500m^2$. With each agent able to observe an area of $25\pi m^2$, we maintain the same density when scaling to 8, 20, 100, and 1000 agents. It is important in RL to train the policies on multiple seeds in order to reduce the possible variance caused by random initialization [22]. We train each policy on multiple random seeds and evaluate on 50 episodes per task for 5 seeds.

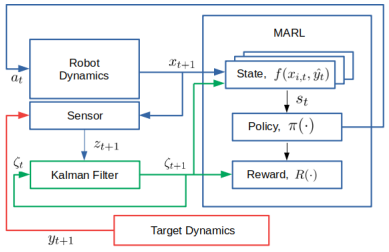


Fig. 2: Multi-Agent Target Tracking Framework. The MARL block is built of any number of homogeneous agents with the same dynamics. Each robot receives a local observation state of target beliefs from the centrally stored Kalman Filters. The policy, π, can easily be exchanged with models such as the DeepSets self-attention, deterministic and stochastic models or the MLP model.

3) Assumptions: In the current framework, we designate a centrally stored Kalman filter that each of the agents has access to and can update. We assume that the data association problem is solved, i.e. if an agent observes a target, the corresponding belief of that target is updated correctly. When a different agent queries the central Kalman filter, it receives the updated information from the rest of the agents. Although this part of the system is centralized, local observations and control actions are not shared amongst the team. Therefore, this formulation allows for an easy extension to a distributed Kalman filtering paradigm [5] for further decentralization.

Targets move around with a double integrator model with Gaussian noise and an additional noise term when faced with obstacles, i.e. the walls of the map. The Kalman filter is updated with the double integrator, however, agents are unaware of this extra noise term. Therefore, when targets bounce off the walls, belief updates become considerably inaccurate leading to a more challenging task.

B. Baseline methods and experiments

Greedy heuristic. Our first baseline is a greedy heuristic mask that reduces the multi-agent multi-target tracking problem into a single-agent single-target problem; each agent's observation state is masked to only have the nearest target's belief. This is a simple baseline and comparisons against it are a sanity check. With the greedy heuristic, note that based on the random initial locations of the targets, two agents can greedily track the same target, leaving targets free. This phenomenon can be seen in the variance of the middle sized tasks in Fig. 3. The greedy heuristic opens up the possibility for performance improvement. *However, we observe in Fig. 3 that even this baseline is better than state-of-the-art graph-based policy gradient (GPG [23]).* Note that these authors employ an idea that has a similar setting: training a graph CNN on a smaller task and evaluating the policy on larger tasks without retraining.

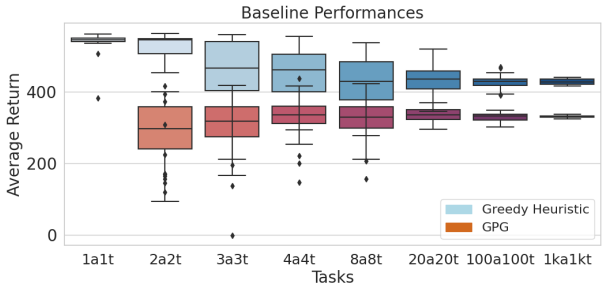


Fig. 3: Task labels, "1a1t" and "1ka1kt", denote the task of the environment where one agent tracks one target and 1000 agents track 1000 targets, respectively. In blue, the trained stochastic-4a4t policy, during evaluation, is masked with the greedy heuristic of $k = 1$, where k is the number of targets to consider. It can be seen that the greedy heuristic baseline does a very good job of extrapolating to tasks outside of the original training space without much performance degradation by reducing the task to a single-agent single target tracking environment. In red, a state-of-the-art graph policy gradient (GPG) algorithm [23] seemingly scales, but suffers in performance. (evaluation on 50 episodes/task for 5 seeds).

Performance of the graph policy gradient (GPG) algorithm. In Fig. 3, GPG is not evaluated on the 1a1t task as with 1 agent there is no graph to define. Although GPG seemingly scales, it consistently falls into a local optima even when trained on multiple seeds and training lengths. We surmise that GPG, designed for the fully observable setting, cannot handle policy learning in a partially observable setting where training is less stable.

Multi-layer perceptron (MLP). We train MLP policies with parameter sharing between pursuer agents using double Q-learning. A different MLP policy is needed depending on the number of targets (1-4), however, parameter sharing allows for a varying number of agents. Baseline performance is shown in Fig. 4. As expected, when the number of targets increase, the expected average return received decreases. Intuitively, this result stems from the fact that even if the uncertainty of 3 of 4 targets is low, a high uncertainty of the last target strongly affects the overall averaged reward.

C. Main results

1) Deterministic vs Stochastic Policies: In real-time, requiring the knowledge of the number of targets at initialization is a severe limitation, therefore, we present deterministic and stochastic self-attention-based policies. Both policies are trained with parameter sharing while the former uses double Q-learning and the latter uses soft double Q-learning. This architecture has the property to allow for changes in number of agents or targets during evaluation. We train a deterministic policy and a stochastic policy that each generalizes over the 16 tasks of $n \in [1, 4]$ agents and $m \in [1, 4]$ targets in the

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 15:03:28 UTC from IEEE Xplore. Restrictions apply.

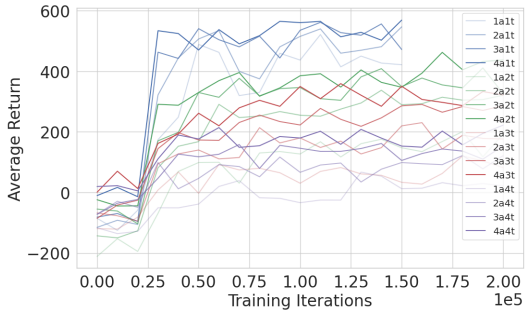


Fig. 4: Online returns of 4 different MLP-based policies based on the number of targets, averaged across 5 seeds. The curve associated with "1a1t" is the cumulative sum of returns collected on an single-agent single-target task. Teams are more successful with more agents (color shades) and suffer when tracking more targets (color-coded).

multi-agent multi-target tracking environment.

From Fig. 5 and Fig. 6, we surmise that the self-attention-based policy that takes deterministic actions, results in a policy that can be extremely greedy and sub-optimal; each agent acting individually acquiring short term rewards rather than as a cooperative team gaining long term high rewards. Although it outperforms the 4 MLP-based policies, it under performs against the greedy heuristic baseline which has an intrinsic sense of target assignment. Therefore, we also developed a stochastic self-attention-based policy which outperforms the greedy heuristic across all tasks with much lower variance. This phenomenon occurs due to the policy acting more intelligently to minimize uncertainty of all targets and forgoes short term rewards for long term higher team rewards, whereas the greedy heuristic policy focuses on the closest target (highly dependent on initializations) and allows the uncertainty of the rest of the targets to explode. We observe a weak form of cooperation from the stochastic policy as it is able to hedge it control actions while robustly tracking across different tasks. Stochastic policies separate itself from the rest of the policies in the more difficult task space of $n < m$ and in simpler parts of the task space, it performs the optimal policy. Episodic visualization can be seen in Fig. 7.

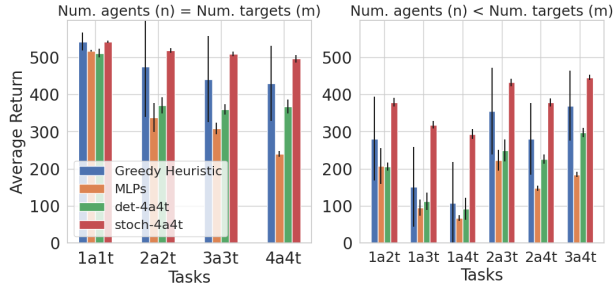


Fig. 5: Comparing the architecturally different policies when $n \leq m$. The stochastic policy (stoch-4a4t) outperforms all other policies with minimal variance. (evaluation on 50 episodes/task for 5 seeds).

In Fig. 7, we illustrate the stochastic policy in a 1 agent tracking 3 targets scenario. The single agent acts intelligently, rotating its sensing region amongst the targets to maintain a strong belief. The greedy heuristic policy (not shown) would focus on the closest target in question and maintain constant observation as it only considers the one-on-one scenario, leaving the other two targets to gain large uncertainties.

V. Related Work

Multi-agent control for pursuit-evasion has seen a wide range of perspectives, see for instance sampling-based al-

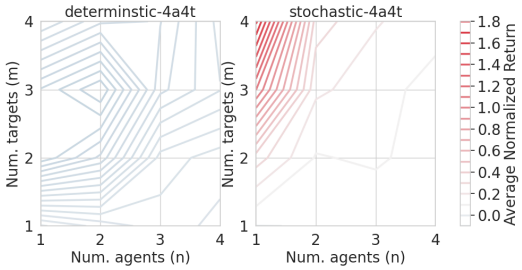


Fig. 6: Contour plots of the deterministic and stochastic policy trained and evaluated on $n \in [1, 4]$ agents tracking $m \in [1, 4]$ targets. The returns are normalized against the greedy heuristic baseline and averaged across the seeds evaluated on 50 episodes/task. Lines on the red spectrum denote returns greater than the baseline (white, 0.0) while blue is less than. Stochastic policies stand out against the baseline in the most difficult parts of the task space.

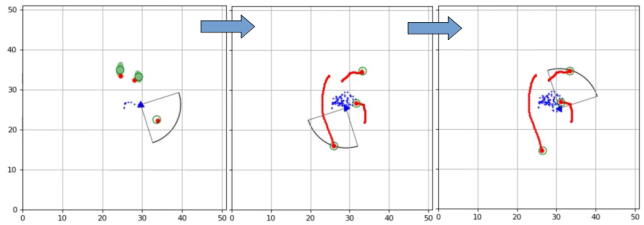


Fig. 7: Example evaluation episode of the stochastic policy maintaining low uncertainty of the 3 targets.

gorithms for few agents which allow for real-time implementation [24]. Analytical solutions for general dynamical systems are difficult to compute, except for special cases, e.g., the homicidal chauffeur [25]. In pursuit-evasion problems, perimeter defense and multi-agent settings have been handled by decomposing the task into two-player games and reducing the defense strategy to an assignment problem [26]. More explicit cooperation can be performed by decomposing into smaller local-games played by a subset of agents [27].

Information acquisition methods that build control policies to minimize the entropy in the estimation task are close to our methods; [9], [28]. These works develop non-myopic control policies by minimizing the uncertainty in the target state, conditioned on past measurements. Decentralized policies can be obtained in MARL using independent Q-functions [11], but training these policies is challenging due to non-stationarity. Centralized policies alleviate non-stationarity, but have to pay the cost of exponentially large multi-agent action-spaces [13], [29]. Works like QMIX [30], and QTRAN [31], formulate a centralized value representation to approximate the collection of individual action-values obtained by each agent during training. Wider literature using graphs networks [23], [32] and permutation invariant structures [33] provides similar claims of scalability and cooperation. However, these works evaluate their algorithms in fully observable settings where training is stable. We observe that when [23] is executed with partial observations, it is difficult to learn a performant policy. There are also directions that focus on emergent behaviors in multi-agent systems to build cooperative and competitive strategies [34], identify emergent roles [35], coordination and emergent use of tools [36].

Similar to us, [37] utilized an attention network in an MARL framework to get permutation invariance. However, they used attention in a centralized critic which forces the input space to scale linearly with the number of agents whereas ours is independent of that dimension. [38] uses

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 15:03:28 UTC from IEEE Xplore. Restrictions apply.

a structured prediction approach to assign agents to tasks. The assignment policy is well learned on small problem instances and evaluated on problems with more agents; they show results for about 80 agents. Other strategies for task allocation in swarm intelligence with multi-agent and multi-target tracking have been also investigated [39], [40]. Another strategy uses a curriculum learning paradigm to scale MARL [41]. Through an evolutionary approach, they are able to introduce more agents into the systems and select the policies that best adapt; they show results with 24 agents.

VI. Conclusions

We presented an off-policy reinforcement learning method for multi-agent target tracking and demonstrated that our decentralized policy is able to scale to an arbitrary number of agents, e.g., up to 1000 pursuers tracking 1000 targets. The key innovation of this approach it to train on a lower dimensional task, e.g., with fewer agents, and execute—without retraining—on larger tasks. Architectural innovations in the policy such as self-attention layers and a masking heuristic at test time enable this and demonstrate how off-policy RL methods can be used for large-scale multi-agent problems.

The central challenge in multi-agent problems lies in the fact that agents need to take decentralized control decisions to work within the constraints of communication latency and yet accomplish some form of cooperation or information sharing to be able to track multiple targets. This paper explores a heuristic to strike such a tradeoff between communication and cooperation. It trains a stochastic policy, as opposed to a deterministic policy, that helps agents hedge their control actions at test time when multiple targets are in the vicinity. This is a very weak form of cooperation, but it comes at zero communication cost. This heuristic is not well-suited for more challenging scenarios which fundamentally require cooperative tracking using multiple agents, e.g., targets whose uncertainty increases quickly. Such environments are good avenues for further exploration.

References

[1] R. Wise and R. Rysdyk, *UAV Coordination for Autonomous Target Tracking*. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2006-6453

[2] A. R. Hilal, "An intelligent sensor management framework for pervasive surveillance," 2013.

[3] V. Kumar, D. Rus, and S. Singh, "Robot and sensor networks for first responders," *IEEE Pervasive Computing*, vol. 3, no. 4, p. 24–33, Oct. 2004. [Online]. Available: https://doi.org/10.1109/MPRV.2004.17

[4] M. Dunbabin and L. Marques, "Robots for environmental monitoring: Significant advancements and applications," *IEEE Robotics and Automation Magazine*, vol. 19, no. 1, pp. 24–39, 2012. [Online]. Available: https://eprints.qut.edu.au/63029/

[5] B. Schlotfeldt, D. Thakur, N. Atanasov, V. Kumar, and G. J. Pappas, "Anytime planning for decentralized multirobot active information gathering," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1025–1032, 2018.

[6] J. Dibangoye, C. Amato, O. Buffet, and F. Charpillet, "Optimally solving dec-pomdps as continuous-state mdps," vol. 55, 08 2013.

[7] R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable markov processes over a finite horizon," *Operations Research*, vol. 21, no. 5, pp. 1071–1088, 1973. [Online]. Available: https://doi.org/10.1287/opre.21.5.1071

[8] H. Jeong, B. Schlotfeldt, H. Hassani, M. Morari, D. D. Lee, and G. J. Pappas, "Learning q-network for active information acquisition," 2019.

[9] N. Atanasov, J. L. Ny, K. Daniilidis, and G. J. Pappas, "Information acquisition with sensing robots: Algorithms and error bounds," 2013.

[10] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Autonomous Agents and Multiagent Systems*, G. Sukthankar and J. A. Rodriguez-Aguilar, Eds. Cham: Springer International Publishing, 2017, pp. 66–83.

[11] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *In Proceedings of the Tenth International Conference on Machine Learning*. Morgan Kaufmann, 1993, pp. 330–337.

[12] J. K. Terry, N. Grammel, A. Hari, L. Santos, and B. Black, "Revisiting parameter sharing in multi-agent deep reinforcement learning," 2020.

[13] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," 2020.

[14] H. Jeong, H. Hassani, M. Morari, D. D. Lee, and G. J. Pappas, "Learning to track dynamic targets in partially known environments," 2020.

[15] J. Mern, D. Sadigh, and M. J. Kochenderfer, "Exchangeable input representations for reinforcement learning," in *2020 American Control Conference (ACC)*, 2020, pp. 3971–3976.

[16] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov, and A. Smola, "Deep sets," 2017.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[18] S. Soatto, "Actionable information in vision," in *2009 IEEE 12th International Conference on Computer Vision*, 2009, pp. 2138–2145.

[19] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," 2008.

[20] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018.

[21] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018.

[22] H. Mania, A. Guy, and B. Recht, "Simple random search provides a competitive approach to reinforcement learning," 03 2018.

[23] A. Khan, E. Tolstaya, A. Ribeiro, and V. Kumar, "Graph policy gradients for large scale robot control," 2019.

[24] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for a class of pursuit-evasion games," vol. 68, 01 2010, pp. 71–87.

[25] R. Isaacs, *Differential Games I: Introduction*. Santa Monica, CA: RAND Corporation, 1954.

[26] M. Chen, Z. Zhou, and C. J. Tomlin, "Multiplayer reach-avoid games via low dimensional solutions and maximum matching," in *2014 American Control Conference*, 2014, pp. 1444–1449.

[27] D. Shishika, J. Paulos, M. R. Dorothy, M. Ani Hsieh, and V. Kumar, "Team composition for perimeter defense with patrollers and defenders," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 7325–7332.

[28] N. Atanasov, J. Le Ny, K. Daniilidis, and G. J. Pappas, "Decentralized active information acquisition: Theory and application to multi-robot slam," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 4775–4782.

[29] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," 2017.

[30] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. Foerster, and S. Whiteson, "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning," 2018.

[31] K. Son, D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi, "Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning," 2019.

[32] S. Li, J. K. Gupta, P. Morales, R. Allen, and M. J. Kochenderfer, "Deep implicit coordination graphs for multi-agent reinforcement learning," 2021.

[33] I.-J. Liu, R. A. Yeh, and A. G. Schwing, "Pic: Permutation invariant critic for multi-agent deep reinforcement learning," 2019.

[34] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," 2015.

[35] T. Wang, H. Dong, V. Lesser, and C. Zhang, "Roma: Multi-agent reinforcement learning with emergent roles," 2020.

[36] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," 2020.

[37] S. Iqbal and F. Sha, "Actor-attention-critic for multi-agent reinforcement learning," 2019.

[38] N. Carion, G. Synnaeve, A. Lazaric, and N. Usunier, "A structured prediction approach for generalization in cooperative multi-agent reinforcement learning," 2019.

[39] K. H. Low, W. K. Leow, and M. Jr, "Task allocation via self-organizing swarm coalitions in distributed mobile sensor network," *Proceedings of the National Conference on Artificial Intelligence*, 10 2004.

[40] K. H. Low, W. K. Leow, and M. H. Ang, "Autonomic mobile sensor network with self-coordinated task allocation and execution," *IEEE Transactions on Systems, Man, and Cypernetics–Part C: Applications and Reviews*, vol. 36, pp. 315–327, 2005.

[41] Q. Long, Z. Zhou, A. Gupta, F. Fang, Y. Wu, and X. Wang, "Evolutionary population curriculum for scaling multi-agent reinforcement learning," 2020.

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 15:03:28 UTC from IEEE Xplore. Restrictions apply.