Hierarchical Reinforcement Learning with Opponent Modeling for Distributed Multi-agent Cooperation

Zhixuan Liang, Jiannong Cao, *Fellow, IEEE*, Shan Jiang[§], Divya Saxena, Huafeng Xu
Department of Computing, The Hong Kong Polytechnic University

***Abstract*—Many real-world applications can be formulated as multi-agent cooperation problems, such as network packet routing and coordination of autonomous vehicles. The emergence of deep reinforcement learning (DRL) provides a promising approach for multi-agent cooperation through the interaction of the agents and environments. However, traditional DRL solutions suffer from the high dimensions of multiple agents with continuous action space during policy search. Besides, the dynamicity of agents' policies makes the training non-stationary. To tackle the issues, we propose a hierarchical reinforcement learning approach with high-level decision-making and low-level individual control for efficient policy search. In particular, the cooperation of multiple agents can be learned in high-level discrete action space efficiently. At the same time, the low-level individual control can be reduced to single-agent reinforcement learning. In addition to hierarchical reinforcement learning, we propose an opponent modeling network to model other agents' policies during the learning process. In contrast to end-to-end DRL approaches, our approach reduces the learning complexity by decomposing the overall task into sub-tasks in a hierarchical way. To evaluate the efficiency of our approach, we conduct a real-world case study in the cooperative lane change scenario. Both simulation and real-world experiments show the superiority of our approach in the collision rate and convergence speed.**

***Index Terms*—Multi-agent Cooperation; Deep Reinforcement Learning; Hierarchical Reinforcement Learning**

I. Introduction

Many complex real-world applications can be modeled as multi-agent cooperation, such as network packet routing [1], energy distribution [2], and coordination of autonomous vehicles [3]. In these applications, the agents need to make individual decisions considering the mutual influence. Traditional solutions of multi-agent cooperation are mostly model-based, in which the agents and their interactions are modeled using physical formulas and prior knowledge [4] [5]. These approaches fail to adapt to dynamic, stochastic, and complex environments. Recently, the development of deep reinforcement learning (DRL) [6] provides a promising solution through a *trial-and-error* process [7]. At each step, the agent observes the environment, selects the optimal action, and receives rewards as feedback signals related to the team performance. The goal of each agent is to learn the policies maximizing the accumulated rewards received from the environment.

Existing DRL approaches for multi-agent cooperation can be classified into three categories: centralized reinforcement learning, centralized training with decentralized execution, and distributed reinforcement learning. Centralized reinforcement learning aims to train a centralized value function that selects actions for all the agents. Such a method is hard to be extended to large-scale scenarios due to the exponential action spaces when the number of agents increases. An alternative approach is centralized training with decentralized execution (CTDE), which trains a critic network to estimate state-action pairs' values. At the same time, each agent maintains an individual actor network for decentralized action selection. By training such a critic network considering the states and actions of other agents, all the agents can learn to cooperate in certain states. Some previous works have applied CTDE for real-world applications, such as flocking [8] and pathfinding [9]. However, the number of features in the critic network needs to be scaled up linearly (in the best case) or exponentially (in the worse case) as the number of agents increases. Besides, the gain of learning a centralized critic is likely to be minimal in the sparse interaction scenarios [10].

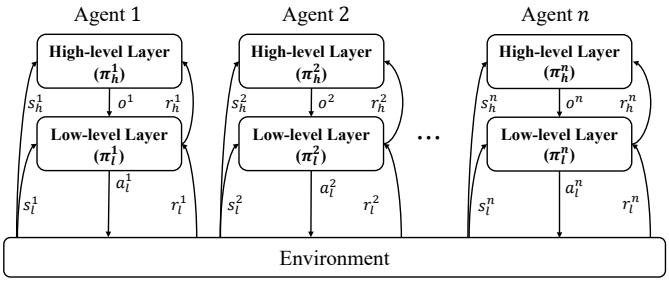


Fig. 1. Illustration of hierarchical reinforcement learning for distributed multi-agent cooperation. Each agent maintains a high-level cooperation layer and a low-level individual control layer.

Recent developments in DRL address the above limitations through decentralized training with decentralized execution (DTDE), where each agent acts and learns to cooperate in a distributed manner [11] [12] [13]. However, it is non-trivial to learn in a distributed way because the training is non-stationary, and it is hard to represent individual agents' policies. In particular, standard DRL approaches relying on end-to-end models suffer from the curse of dimensionality of multiple agents in continuous action spaces. It demands large-scale models and a long time for training to learn a mapping function from the state to the continuous action space. Furthermore, the independence of individual policy learning leads to poor coordination. Finally, the dynamicity of the agents' policies makes it nearly impossible to employ experience replay [14] that is crucial for stabilizing DRL.

[§]Corresponding author: Shan Jiang

In this paper, we propose a novel hierarchical deep reinforcement learning approach for distributed multi-agent cooperation. First, we decompose the policy space of each agent into a high-level cooperation layer and a low-level individual control layer, which is shown in Fig. 1. The cooperation of multiple agents can be efficiently learned in high-level discrete action space, while the low-level individual control can be reduced to independent reinforcement learning. Besides, we introduce an opponent modeling mechanism to model other agents' high-level decisions. The learned opponent models will encourage cooperation behaviors and stabilize DRL. In contrast to the standard end-to-end DRL model, we reduce the learning complexity by decomposing the overall task into subtasks that are easier to solve hierarchically. Furthermore, we conduct a case study on cooperative lane change and present the hierarchical decision-making model with the specific design of states, actions, and rewards. Finally, we evaluate our approach in a simulation environment and a real-world testbed. The main contributions of this paper are as follows:

- We study the problem of distributed multi-agent cooperation in continuous action space and propose a hierarchical deep reinforcement learning approach, which decomposes the overall task into high-level option selection and low-level individual control.
- To address the non-stationary training issue in distributed multi-agent learning, we introduce an opponent modeling mechanism to model other agents' high-level policies.
- We conduct extensive experiments on cooperative lane change and compare our approach with state-of-art baselines in simulation and a real-world testbed. The experimental results show the superiority of our approach in both task performance and convergence speed.

The rest of the paper is organized as follows. Sec. VI summarizes the previous works of multi-agent reinforcement learning and hierarchical reinforcement learning. Sec. II introduces the notations and preliminaries of deep reinforcement learning. Sec. III presents a novel hierarchical reinforcement learning algorithm with an opponent modeling mechanism. Sec. IV showcases a real-world case study of hierarchical reinforcement learning on the multi-vehicle cooperative lance change scenario. Sec. V shows the experimental results in simulation and a real-world bed. Finally, Sec. VII concludes the paper and discusses the future directions.

II. Preliminaries

In this section, we introduce the definitions of Markov decision process (MDP), reinforcement learning (RL), and hierarchical reinforcement learning (HRL).

A. Markov Decision Process

Many decision-making problems can be mathematically modeled as a Markov decision process (MDP). An MDP is defined as a four-tuple (S, A, r, T), in which S is the set of states, A is the set of available actions, $r : S \times A \times S \to R$ is the reward function, and $\mathcal{T} : S \times A \times S \times \to [0, 1]$ is the probability function of state-action-state transition. A stochastic policy function is defined as $\pi : S \times A \to [0, 1]$ and a deterministic policy function $\mu : S \to R^{|A|}$ is defined to select an action for current state. The goal of solving MDP is to find a policy which selects actions maximizing the cumulative rewards $R = \sum_{t=0}^{N} r_t$.

Given a policy π and a state s, the value function $V : S \to R$ calculates the expected sum of discounted rewards using the following formula:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum\nolimits_{t=0}^{N} \gamma^t r_t \mid s\right]$$

where γ is the discount factor and r_t is the reward received at each time step. In addition, given a policy π, a state s, and an action a, the state-action value function $Q : S \times A \to R$ calculates the expected sum of discounted rewards from a given state as follows:

$$Q^{\pi}(s, a) = \sum\nolimits_{\pi}\left[\sum\nolimits_{t=0}^{N} \gamma^t r_t \mid s, a\right]$$

Solving an MDP requires computing the state-value function and state-action value function. The reward function and state transition are entirely known, these functions can be solved by iterating the Bellman equations, and the decision-making problem can be solved by using dynamic programming [15]. However, the reward function and state transition are unknown to agents in most cases.

B. Reinforcement learning

Reinforcement learning provides multiple methods for solving MDPs, classified as value-based and policy-based methods.

In value-based method, the state-action-value function (or state-value function) is estimated using temporal different (TD) methods as follows:

$$\begin{aligned} &Q(s_t, a_t) \\ =&Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a^{t+1}} Q(s^{t+1}, a^{t+1}) - Q(s_t, a_t)) \end{aligned}$$

Then the greedy policy can be derived by selecting the action with highest Q-value at each time step as follows:

$$\pi(s_t) = \arg\max_{a} Q(s_t, a_t)$$

Such a method is referred to as the Q-learning [16].

Traditional tabular Q-learning suffers from the issue of high dimensionality in state space. Deep Q-learning (DQN) [17] addresses the issue by employing a deep neural network to approximate the Q-function. Besides, DQN is a class of off-policy reinforcement learning methods that uses a *replay memory* to store the transition four-tuple (s, a, r, s') and these data can be sampled to train the Q-network. The loss function of training Q-network is as follows:

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s'}\left[\left(Q\left(s, a; \theta\right) - y\right)^2\right]$$

where $y = r + \gamma \max_{a \in \mathcal{A}} \left(Q\left(s\prime, a; \theta^{-}\right)\right)$ and $\gamma \in [0, 1)$ is the discount factor. In y, the variable θ^{-} is the parameter of the *target network* that is periodically copied from θ that is kept constant for a number of iterations.

An alternative model-free approach is the policy-based method, which directly learns a policy π parameterized by θ^{π}. The objective of these approaches is to adjust the parameters θ^{π} in order to maximize the function $J(\theta) = \mathbb{E}_{s\sim p^{\pi}, a\sim\pi_{\theta}}[R]$, where R denotes the expected accumulative rewards which are usually approximated by $Q(s, a)$. In the actor-critic theory, the policy network π_{θ} can be considered an actor network, and the state-action value function $Q(s, a)$ can be considered a critic network. According to the policy gradient theorem [15], the gradient of this objective function is defined as follows:

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{s\sim\rho^{\pi}, a\sim\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a|s) Q^{\pi}(s, a)]$$

where ρ^{π} is the state distribution under policy π_{θ} and $\pi_{\theta}(a|s)$ is the probability of select the action a under the state s. At each time step, the agent samples an action from the distribution generated from the policy network.

Deep deterministic policy gradient (DDPG) is an actor-critic algorithm extended from stochastic policy gradient to deterministic policy gradient [18] [19]. It employs a deep neural network parameterized by θ^{μ} to approximate the deterministic policy $\mu_{\theta} : S \rightarrow A$ and a neural network parameterized by θ^{Q} to approximate the action-value function $Q(s, a|\theta^{Q})$. The critic network is learned using the Bellman equation as in Q-learning, and the actor network is updated by applying the chain rule to the objective function J:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{s\sim\mathcal{D}} \left[\nabla_{\theta} \mu_{\theta}(a \mid s) \nabla_{a} Q^{\mu}(s, a)|_{a=\mu_{\theta}(s)} \right]$$

C. Hierarchical Reinforcement learning with Temporal Abstraction

Human decision-making often involves choosing among temporally extended courses of action over a broad range of time scales [20]. Learning and operating over different levels of temporal abstraction is a critical challenge in tasks involving long-term planning. Sutton et al. [21] extended reinforcement learning framework to include temporally abstract actions, representations that group together a set of interrelated actions (for example, moving to block A, driving another lane, passing the ball to another person). These representations can be translated into a series of individual actions, which are described as *temporal abstraction*. A recent extension of this direction is hierarchical deep reinforcement learning with temporal abstraction proposed by Kulkarni et al. [22]. In their approach, the agent uses a two-level decision-making model, i.e., a *meta-controller* and a *controller*. The meta-controller receives a state s_t and chooses a goal $g_t \in \mathcal{G}$, where $\mathcal{G}$ denotes the set of all possible current goals. The g_t will remain for T time steps, or the terminated state is reached. The Q-value function for the controller is:

$$Q_l(s, a; g) = \mathrm{E}\left[r_l + \gamma \max_{a\prime} Q_l(s\prime, a\prime; g) \right]$$

where g is the goal and r_l is low-level reward given by a goal-driven reward function $R(s_t, a_t, g_t)$. This low-level reward is related to performing the goal (temporal abstraction), which is also regards as *intrinsic reward*. The design of intrinsic reward and termination is still an open question in hierarchical reinforcement learning.

Similarly, the Q-value function of the meta-controller is:

$$Q(s, g) = \mathrm{E}[r_h + \gamma \max_{g'} Q(s', g')]$$

where $r_h = \sum_{t=0}^{T} r$ is the accumulative reward received from the environment during the T time steps.

III. HERO: Hierarchical Reinforcement Learning with Opponent Modeling

This section introduces the sequential decision-making problem in distributed multi-agent systems. Then, we propose HERO, a general hierarchical decision-making model with high-level cooperative decision-making and low-level individual control. Note that, in many applications, the primitive actions of agents are regarded as individual control.

A. Problem Formulation

Typically, multi-agent cooperation can be mathematically modeled as multi-agent Markov games, which extends MDP to multi-agent setting [23]. A Markov game is defined as a five-tuple (I, S, A, r, T), where I is the set of N agents, S is the set of states, $A = A_1 \times A_2 \ldots \times A_N$ is the set of actions of all agents, and $r = (r_1, r_2, \ldots, r_N)$ where $r_i : S \times A \times S \rightarrow \mathbb{R}$ is the reward function of agent i. In the fully cooperative setting, we have $r_1 = r_2, ..., = r_N$. Given the current state and the actions of all agents, $T : S \times A \times S \rightarrow [0, 1]$ is the probability distribution over the next states. Each agent i aims to learn a policy $\pi_i : S_i \times A_i \rightarrow [0, 1]$ to select the optimal action a_i maximizing the accumulative rewards $R_i = \sum_{t=0}^{T} \gamma^t r_i^t$, where γ is the is the discount factor and T is the time horizon. We can formulate the joint policy of other agents as:

$$\pi^{-i}\left(a_t^{-i} \mid s_t\right) = \prod\nolimits_{j\in\{-i\}} \pi^{j}\left(a_t^{j} \mid s_t\right)$$

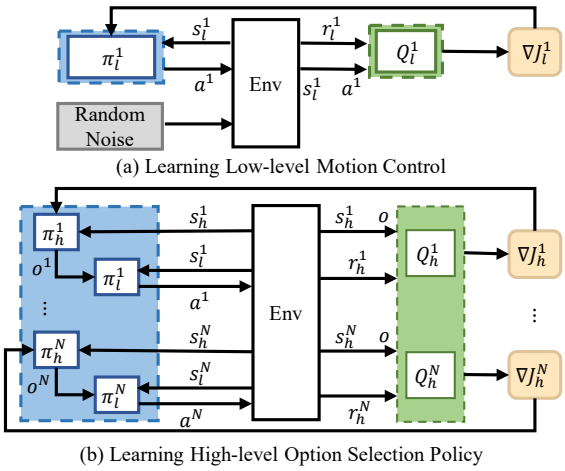
Therefore, the objective of each agent i is defined as follows:

$$\begin{aligned} &\max_{\pi^i} \eta_i \left[\pi^i, \pi^{-i}\right] \\ =&\mathbb{E}_{\left(s_t, a_t^i, a_t^{-i}\right)\sim T, \pi^i, \pi^{-i}} \left[\sum\nolimits_{t=1}^{\infty} \gamma^t R^i\left(s_t, a_t^i, a_t^{-i}\right) \right] \end{aligned}$$

Most existing multi-agent reinforcement learning methods assume that each agent can access all the other agents' policies during the training. In this paper, we consider a general scenario in a distributed manner, in which each agent i has no knowledge of other agents' policies, and can only observe the historical states and actions of other agents, i.e., $\left\{s_{1:t-1}, a_{1:t-1}^{-i}\right\}$. Such a setting is also found in other decentralized MARL works [11] [13].

B. Hierarchical Decision-making Model for Multi-agent Cooperation

Hierarchical Policy Representation. In this paper, we introduce a hierarchical reinforcement learning model for multi-agent cooperation, which decomposes an overall cooperation task into a hierarchy of discrete sub-tasks. The sub-tasks are also regarded as options, skills, or temporal abstractions



(a) Learning Low-level Motion Control

(b) Learning High-level Option Selection Policy

Fig. 2. Two-stage training structure of HERO. (a) Each individual agent learns different individual control policy with random noise in the first stage. (b) Multiple agents learn to select options in the second stage.

in previous studies. For simplicity, we use the notation of option in the rest of this paper. In a two-layer hierarchical reinforcement learning model, the policy space of each agent is defined as follows:

$$\pi^i = \left[\pi_h^i, \pi_l^i\right]$$

where π_h is the high-level policy to select the options and π_l is the low-level policy to select primitive actions to perform the selected option.

A high-level option is defined as a three-tuple $o = (I_o, \pi_h, \beta_o)$, where $I_o \in \mathcal{S}_h$ is the initiation set, π_h: the option selection policy, and $\beta_o : \delta \to [0, 1]$ specifies the termination condition while executing o.

In the definition of high-level option, S_h denotes the state space, and A_h denotes the action space of high-level policy. The option o can also be interpreted as the high-level action with the state set S_h. In the context of multi-agent cooperation, each agent i cooperatively selects the option based on the current state s_h^i and the inferred opponent options o_h^{-i}, which is shown in Fig. 2.

Asynchronous Option Termination. There are two modes for option termination: synchronous and asynchronous. Synchronous termination requires all agents to interrupt the existing option execution and select the next option synchronously, which is not feasible for fully distributed systems. Thus, we consider the asynchronous termination mode for each agent. At each time step t, the high-level layer of each agent i will verify whether the current state $s_{h,t}^i$ satisfies the termination condition. In particular, if β_o^i is equal to 1, the existing option will terminate, and the high-level policy will select another option to execute until the overall task is finished.

Hierarchical Reward. Also, the reward of each agent i can be divided into two parts for efficient training:

$$r^i = \left[r_h^i, r_l^i\right]$$

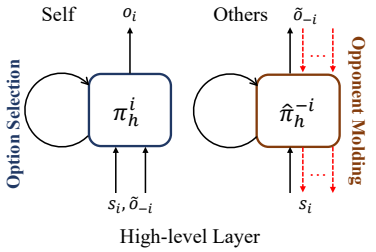


Fig. 3. Illustration of the high-level opponent modeling in high-level layer. Each agent maintain a self policy network for its option selection and a opponent modeling network for other agents' option prediction.

where r_h^i and r_l^i are the high-level and low-level rewards concerning team performance and individual control, respectively.

C. Learning High-level Option Selection with Opponent Modeling

Option-value Function. The value function of each agent's option selection can be defined as follows:

$$Q_h^i\left(s_{h,t}^i, o_t\right) = r_{h,t:t+c}^i + \gamma Q_h^i\left(s_{t+c}^i, o_{t+c}\right) - Q_h^i\left(s_{h,t}^i, o_t\right)$$

where $\mathbf{o_t} = [o_t^i, o_t^{-i}]$ denotes the option selected by agent i and the options selected by other agents. Here, we assume that each agent can observe all the previous option selections of others agents, i.e., $\left\{o_{1:t-1}^{-i}\right\}$, at each time step. Besides, $r_{h,t}^i$ is the high-level reward received from environment which indicates the progress of the overall task and $r_{h,t:t+c}^i = \sum_t^{t+c} r_{h,t}^i$ denotes the accumulated high-level reward when performs the option o_t^i. Thereafter, the high-level transition that will be stored in the replay buffer $\mathcal{D}_h^i$ is expressed as follows:

$$\{(s_{h,t+k}^i, o_{t+k}^i, o_{t+k}^{-i}, r_{h,t+k}^i, s_{h,t+k+1}^i)\}_{k=0}^{k=c}$$

Agent Learning. We propose a *decentralized actor-critic* method for high-level policy learning. Each agent trains a decentralized critic network $Q_h^i\left(s_h^i, o^i, o^{-i}; \theta_{i,h}^Q\right)$ to approximate the option-value function, while maintaining an actor network $\pi_h^i(s_h^i, o^{-i}; \theta_{i,h}^\pi)$ for option generation. Specifically, the critic network is trained by minimizing the loss:

$$\mathcal{L}\left(\theta_{i,h}^Q\right) = \mathbb{E}_{s_h^i, o^i, o^{-i}, r_h^i \sim \mathcal{D}_h^i}\left[\left(Q_{h,t}^i\left(s_{h,t}^i, o_t^i, o_t^{-i}\right) - y_h^i\right)^2\right]$$

$$y_h^i = r_{h,t:t+c}^i + \gamma Q_{h,t+c}^i\left(s^i, o_{t+c}^i, \hat{o}_{t+c}^{-i}\right)$$

where $\mathcal{D}_h^i$ is the experience replay buffer for offline high-level policy training. Then, the actor network can be trained with gradient ascent where the gradient is computed as:

$$\nabla_{\theta_{i,h}^\pi} J\left(\theta_{i,h}^\pi\right) =$$
$$\mathbb{E}_{s_h^i \sim p^\pi, o^i \sim \pi_h^i}\left[\nabla_{\theta_i} \log \pi_h^i\left(o^i \mid s_h^i, \hat{o}^{-i}\right) Q_h^i\left(s_h^i, o^i, o^{-i}\right)\right]$$

where p^π denotes the state transition under all agents' policies $\pi = \pi_1, ..., \pi_n$ and $\hat{o}^{-i}$ denotes the inferred options of other agents generated from the opponent modeling network $\hat{\pi}_h^{-i}$.

Opponent-modeling in Option Selection. Opponent modeling is a feasible solution to the non-stationary issue in distributed reinforcement learning. Instead of modeling other

Algorithm 1 Training High-level Cooperative Strategy with Opponent Modeling

initialize the experience replay buffer $\mathcal{D}_h^i$ and the parameters $\theta_{i,h}^Q$, $\theta_{i,h}^\pi$ for each agent i
initialize the experience replay buffer $\mathcal{D}_h^{-i}$ and the parameters π_h^{-i} for opponent modeling
for episode $\leftarrow 1$ **to** M **do**
  t $\leftarrow 0$
  reset the environment and receive an initial states $s_{h,t}^i$ and $s_{l,t}^i$ for each agent i
  for each agent i select high-level action (option) from the actor network π_h^i
  $o_t^i \leftarrow \pi_h^i$
  while the task is not completed **do**
    pass the high-level option o_t^i to low-level controller and generate low-level action a_t^i
    execute action a_t^i and receive next state $s_{h,t+1}^i$
    compute the high-level team reward r_h^i
    for agent $i \leftarrow 1$ **to** N **do**
      if $s_{h,t+1}^i$ is a terminal state for option o_t^i **then**
        choose next option o_{t+1}^i from actor network π_h^i
        store transition $(s_{h,t}^i, o_t^i, r_h^i, s_{h,t+1}^i)$ in $\mathcal{D}_h^i$
        $o_t^i \leftarrow o_{t+1}^i$, $s_{h,t}^i \leftarrow s_{h,t+1}^i$
      else
        $o_t^i \leftarrow o_{t+1}^i$
      end if
      update the high-level critic network
      update the high-level actor network
    end for
    store transition (s_t^i, o_t^{-i}) in $\mathcal{D}_h^{-i}$
    t $\leftarrow t+1$, $s_t \leftarrow s_{t+1}$
    update the opponent modeling networks
  end while
  update the target network parameters
end for

agents' primitive actions, we propose to model their option selections reflecting their temporal abstraction during a few time steps. Then, two questions arise: *how to use the opponent model* and *how to train the opponent model*.

For the model usage, we use the opponent model in the individual option selection to encourage the coordination behaviors. Each agent i makes a decision based on both the current state s_h^i and the inferred opponent options $\hat{o}^{-i}$, which is shown in Fig. 3. Additionally, we use the latest opponent model to update the TD-target y_h^i:

$$y_h^i = r_{h,t:t+c}^i + \gamma Q_{h,t+c}^i\left(s_h^i, \pi_h^i(s_{h,t+c}^i), \pi_h^{-i}(s_{h,t+c}^i)\right))$$

Note that we input the option log probabilities of other agents directly into $Q_{h,t+c}^i$, rather than sampling.

For the opponent model training, we use a deep neural network to approximate the option selection policies of other agents and train the network by maximizing the log probability

Algorithm 2 Training Low-level Driving Skills with Intrinsic Reward Functions

initialize the experience replay buffer $\mathcal{D}_l$ and the parameters θ_Q^l, θ_μ^l for low-level controller
for episode $\leftarrow 1$ **to** M **do**
  reset the environment and receive an initial state s_0
  t $\leftarrow 0$
  while s_t is not a terminal state **do**
    select action from low-level actor network
    $a_t \leftarrow \pi_l(\cdot \mid \mathbf{s}_t)$
    execute action a_t and receive next state s_{t+1} and intrinsic reward r_l
    store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}_l$
    t $\leftarrow t+1$
    $s_t \leftarrow s_{t+1}$
    update the low-level critic network
    update the low-level actor network
  end while
  update the target network parameters
end for

from the recent observation histories:

$$\mathcal{L}\left(\theta_{\pi,h}^{-i}\right) = -\mathbb{E}_{s_h^i, o_h^{-i}}\left[\log \pi_h^{-i}\left(o_h^{-i} \mid s_h^i\right) + \lambda H\left(\pi_h^{-i}\right)\right]$$

where $\pi_h^{-i}\left(o^{-i} \mid s_h^i\right)$ is the predicted probability of selected action o^{-i} given the observation of agent i and $H\left(\pi_h^{-i}\right)$ is the entropy of the policy distribution which is used to solve the over-fitting problem in deep learning.

D. Learning Low-level Policies with Intrinsic Reward Functions

Given the option, the objective of the low-level layer is to perform the option by selecting optimal primitive actions. The low-level individual control policy π_l^i is represented by an individual deep neural network θ_l^i and trained with independent and parallel deep reinforcement learning. To successfully learn to perform the option, the intrinsic reward function r_l^i is used for efficient training. In this paper, we adopt the *soft actor-critic* method. According to the maximum entropy reinforcement learning theorem [24], the objective function of low-level policy is defined as follow:

$$J(\pi_l^i) = \sum_{t=0}^{c} \mathbb{E}_{\left(s_l^i, a^i\right)}\left[r_l^i\left(s_l^i, a^i\right) + \alpha \mathcal{H}\left(\pi_l^i\left(\cdot \mid s_l^i\right)\right)\right]$$

where $\mathcal{H}$ is the regulation term to encourage exploration and learn diverse policies to perform the specific option and α is the hyperparameter denote the weight of action exploration. Consequently, the gradient of each agent is:

$$\nabla_{\theta_l^i} J\left(\theta_l^i\right) = \mathbb{E}\left[\nabla_{\theta_l^i} \epsilon\left[-\alpha\epsilon + Q_l^i\left(s_l^i, a^i; o_h^i\right)\right]\right]$$
$$\text{where } \epsilon = \log \boldsymbol{\pi}_l^i\left(a^i \mid s_l^i\right)$$

In practice, the training of critic can be realized by parameter sharing among distributed agents. Then, the critic network

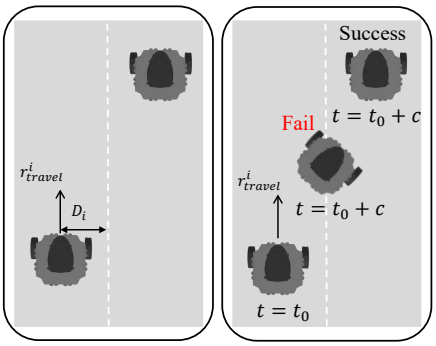


Fig. 4. The left side of this figure shows the reward calculation for the training of driving in lane, while the right side shows the reward calculation for training of lane change.

$Q_l^i\left(s_l^i, a^i; \theta_l^i, o_h^i\right)$ can be trained with deep Q-learning and the loss function is:

$$\mathcal{L}(\theta_l^i) = \mathbb{E}_{s_l^i, a^i, r_l^i, s_l^{i,'} \sim \mathcal{D}_l^i}\left[\left(Q_l^i\left(s_l^i, a^i; \theta_l^i, o_h^i\right) - y_l^i\right)^2\right]$$

$$y_l^i = r_l^i + \gamma Q_l^i\left(s^{i,'}, a_i^{'}; \theta_l^i; o_h^i\right)$$

where D_l^i is the experience reply buffer for off-policy training and r_l^i is the intrinsic reward designed for the option learning.

IV. Case Study: HERO for Multi-vehicle Cooperative Lane Change

In this section, we present a case study of HERO on cooperative lane change to demonstrate the practicability of HERO in real-world applications.

A. Cooperative Lane Change

Lane change is one of the most essential and fundamental behaviors in autonomous driving, which involves the interaction of multiple vehicles in different lanes. Successful lane changes require drivers to account for several safety-related factors, including the road geometries, positions of ego vehicles, and cooperation with other vehicles. Inaccurate lane change leads to car accidents, congestion, and waste of energy. This paper aims to solve decision-making problems under different lane change scenarios. Many existing works only consider the single-agent driving scenario, while this work aims to leverage the cooperativeness of multiple vehicles to improve road safety and efficiency.

In this paper, we propose a two-layer hierarchical decision-making model, which includes a high-level controller and a low-level controller for cooperative driving. As shown in Fig. 5, we first decompose the driving task into several sub-tasks, i.e., lane change, lane-keeping, and slow down. Each sub-task can be regarded as an option that the low-level controller can perform. It takes several time steps for the low-level controller to finish the option successfully. The objective of the high-level controller is to learn to select different options according to the current state that can maximize the traffic throughput while avoiding collisions. Specifically, the high-level controller's state, option, and reward function are defined in the following subsection.

B. High-level State, Option and Reward Function Design

High-level State Space. Here, we separate the state space of each vehicle into s_h^i and s_l^i, which are the state for high-level controller and low-level controller, respectively. The high-level state s_h^i is defined as follows:

$$s_h^i = \left[s_{lidar}^i, s_{speed}^i, s_{laneID}^i\right]$$

where s_{lidar}^i is the raw observation from the lidar sensor, which denotes the distance with other vehicles from 360 degrees, s_{speed}^i is the vehicle's speed, and s_{laneID}^i is the vehicle's current lane identifier.

High-level Option Space. The action space of high-level controller is to choose *which option to execute*. Here, we consider the discrete action space defined as follow:

$$A_h^i = [\text{keep lane}, \text{slow down}, \text{accelerate}, \text{lane change}]$$

We define four types of high-level options according to human driving behaviors, where each option corresponds to a learned low-level control policy.

High-level Team Reward. The design of high-level team reward is related to the overall objective. Typically, safety and efficiency are two main concerns in cooperative merge scenarios. In terms of safety, each vehicle should not only avoid collision with the front vehicle when driving in the current lane but also pay attention to the vehicles in the merged lane. Once the collision happens, we will assign a negative reward r_{col} to the vehicles as the penalty, and the current training episode will be ended and restarted. To avoid traffic congestion, we encourage the vehicle to run as fast as possible by giving a positive reward r_{travel} to vehicles. Hence, the total reward of each vehicle is designed as follows:

$$r_h^i = \alpha r_{col} + (1-\alpha) r_{travel}^i$$

where r_{col} is the penalty of collision, r_{travel} is the reward for moving forward, and α is a hyperparameter to control the weight of collision avoidance and move forward.

C. Low-level State, Action and Intrinsic Reward Function Design

Low-level State Space. Previous works rely on the given waypoint data of the vehicles and use PID controller to drive. In this paper, we consider learning driving skills based on vision, which is similar to how human drive. Therefore the low-level state space s_l^i is defined as follows:

$$s_l^i = \left[s_{img}^i, s_{speed}^i, s_{laneID}^i\right]$$

where s_{img}^i denotes the captured image, s_{speed}^i represents the speed, and s_{laneID}^i is the identifier of current lane.

Low-level Action Space. Because the low-level controller directly determines the motion of the vehicle, the action space of the low-level controller is the continuous linear speed

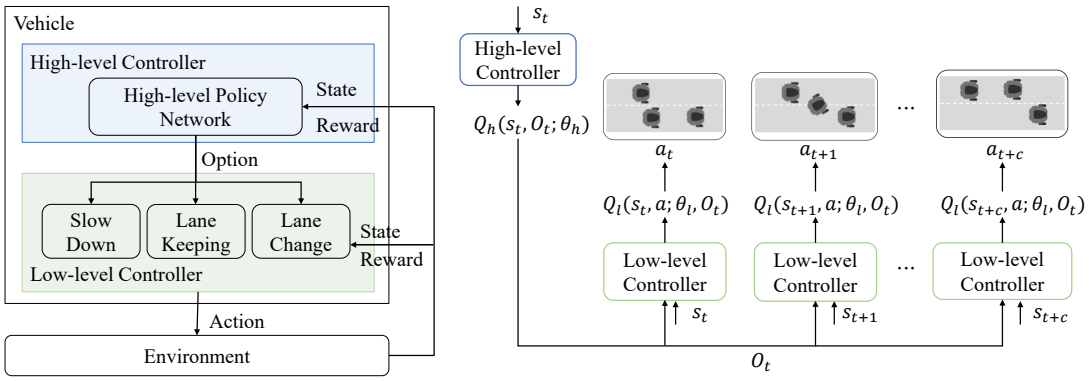


Fig. 5. Hierarchical decision-making model for each vehicle.

and angular speed. If the high-level option selection is lane-keeping, then the linear and angular speeds will remain the same compared to the last step. Otherwise, the action spaces of different skills are defined as follows:

$$A^i_{\text{linear}} = \begin{cases} 0.04 : 0.08 & \text{slow down} \\ 0.08 : 0.14 & \text{accelerate} \\ 0.1 : 0.2 & \text{lane change} \end{cases}$$

$$A^i_{\text{angular}} = \begin{cases} -0.1 : 0.1 & \text{slow down} \\ -0.1 : 0.1 & \text{accelerate} \\ 0.12 : 0.25 & \text{lane change} \end{cases}$$

where A^i_{linear} denotes the action space for linear speed and $A^i_{angular}$ is the action space for angular speed. To prevent unsafe actions and improve exploration efficiency, we set different lower and upper bounds of the linear speed and angular speed during the training.

Low-level Intrinsic Reward Functions. In contrast to the high-level team reward which directly related to the success of the overall task, the principle of *intrinsic reward* is to provide feedback for learning different skills. Here, we categorize the intrinsic rewards into two types: $r^i_{\text{driving in lane}}$ and $r^i_{\text{lane change}}$. An illustration of reward calculation is shown in Fig. 4. In particular, $r^i_{\text{driving in lane}}$ is used for $o^i_h = [\text{slow down}, \text{accelerate}, \text{keep lane}]$ calculated as follows:

$$r^i_{\text{driving in lane}} = \beta \cdot r^i_{deviate} + (1 - \beta) \cdot r^i_{travel}$$

where $r^i_{deviate}$ denotes the distance deviated from the center of current lane, r^i_{travel} is the travel distance used to encourage the movement of each vehicle, and β is a hyper parameter controlling the weights of deviated distance and travel distance.

Similarly, the intrinsic reward for lane change skill is designed as follows:

$$r^i_{\text{lane change}} = \begin{cases} 20 & \text{success} \\ -20 & \text{fail} \\ r^i_{travel} & \text{else} \end{cases}$$

For the lane change skill, we assign a positive reward of 20 when the vehicle successfully changes the lane and assign a negative penalty if the vehicle fails to change lane within a predefined maximum time step.

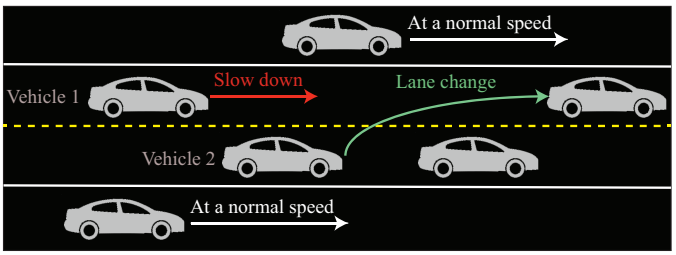


Fig. 6. Illustration of the cooperative lane change scenario, where the vehicle 1 should coordinate with vehicle 2 to avoid the collision when vehicle 2 is performing lane change

TABLE I
HYPERPARAMETERS FOR TRAINING

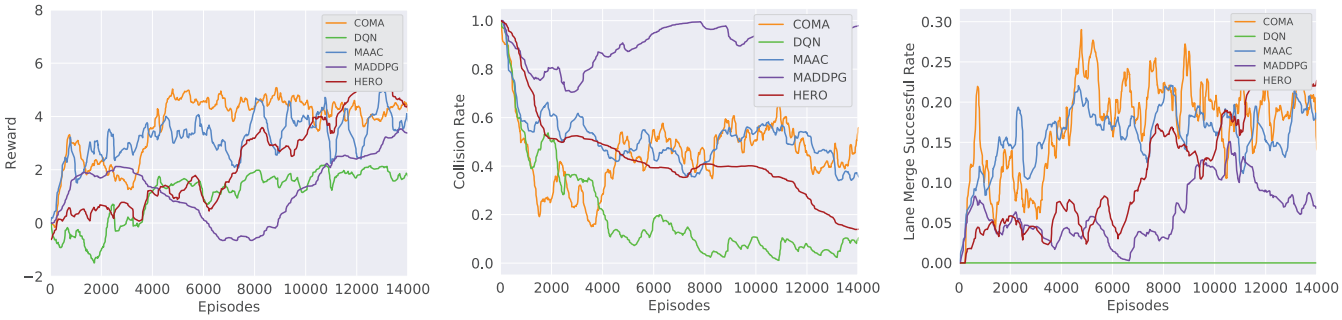
Hyperparameter	Value
Training episode	$14,000$
Episode length	30
Buffer capacity	$100,000$
Batch size	1024
Learning rate	0.01
Discount factor γ	0.95
Dimension of the hidden layer	32
Target network update rate	0.01

V. PERFORMANCE EVALUATION

This section presents the results of applying our approach in cooperative lane change scenarios, which is shown in Fig. 6. We first create a simulation environment for training and then build a real-world testbed for further evaluation. We compare the performance of our approach with several well-known reinforcement learning approaches.

A. Baseline Approaches

In this paper, we apply different MARL approaches to cooperative lane change scenarios, including distributed approaches (DTDE) and centralized training with decentralized execution



(a) Mean episode reward (b) Collision rate (c) Lane change successful rate

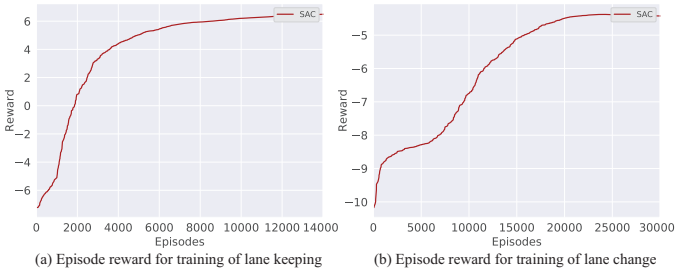
Fig. 7. Comparison of the learning curve of different approaches in the cooperative lane change scenarios.

(CTDE) approaches. Specifically, the considered methods for comparison include:

- *Independent Deep Q-learning (DQN)*. It is a distributed learning approach, in which each agent trains a Q-network using its local observation and *shared team reward*. Each agent applies the ϵ-greedy strategy for action exploration during the training.
- *Counterfactual policy gradient (COMA)*. It is a standard CTDE approach where the centralized critic is trained with Q-learning. The actor network is trained with the counterfactual policy gradient theorem.
- *Multi-agent mixed actor-critic (MADDPG)*. It is another approach adopting the framework of CTDE. Unlike COMA, it considers the environment with explicit communication and trains a centralized critic for each agent, allowing each agent to have different reward functions.
- *Multi-Actor-Attention-Critic (MAAC)*. It is the state-of-the-art approach that extends the attention mechanism, which is widely used in image processing and natural language processing tasks, to multi-agent reinforcement learning. It trains an actor-attention-critic network for each agent and allows parameter sharing to improve the learning efficiency. Noted that, MAAC uses decentralized critics with a decentralized actor with parameter sharing.

B. Training Environment and Evaluation Metrics

To evaluate our approach, we first create a simulation environment based on Gazebo, a popular physical simulation engine in robotics [25]. The simulation environment includes the lane scenario and multiple vehicles equipped with cameras and lidar sensors. Besides, we provide several RL-friendly APIs that are convenient for algorithm implementation. Fig. 12 shows the simulation scenarios of the cooperative lane change. When the vehicle in front of vehicle 2 stops or moves slowly, vehicle 2 needs to control itself to perform lane change which needs the coordination among vehicle 1 and vehicle 2. We use a conventional neural network to encode the image data and a multi-layer fully-connected neural network as the critic network for policy implementation. The dimension of the hidden layer is set to 32 in all the algorithms.



(a) Episode reward for training of lane keeping (b) Episode reward for training of lane change

Fig. 8. Episode reward of learning different low-level skills.

At the beginning of each episode, we randomly initialize several positions of the vehicles and set the same initial speed to them. Besides, the episode length is set to 18 time steps. The hyper-parameters of our network architecture and learning algorithms are presented in Tab. I. When the vehicles get collision, or the max episode length is reached, the program will call the *reset* function provided by our testbed to reset the environment and continue the training.

In this paper, we consider four evaluation metrics to measure the performance of different approaches, including:

- *Mean episode reward*: the average reward of each time step when sampling the replay buffer from the different episodes.
- *Collision rate*: the collision ratio of the vehicles in each episode.
- *Lane merge successful rate*: the proportion of vehicles that successfully merge in each episode.
- *Mean speed*: the average of the vehicle's speeds at each time step.

C. Learning Low-level Skills

Before training cooperative strategies, we first train the low-level skills, including lane tracking and lane change. We create parallel training environments with different intrinsic reward functions so that the low-level learned skills can be further shared. To improve learning efficiency, we set a lower angular speed in training the lane-keeping skill and set a higher angular speed when training the lane change skill. Fig. 8

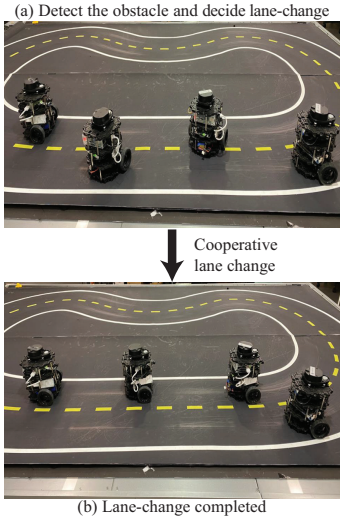
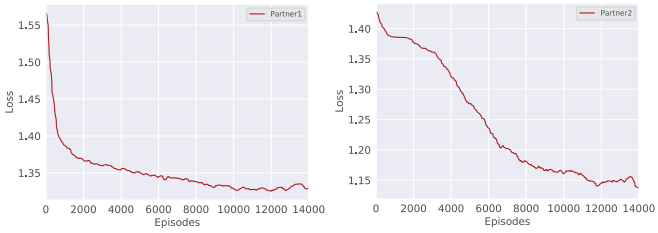


Fig. 9. Real-world evaluation



(a) Policy modeling loss of vehicle1 (b) Policy modeling loss of vehicle2

Fig. 10. Learning loss of modeling other partners' policies.

shows the episode rewards during the training process. The experimental results imply that the soft actor-critic algorithm can successfully converge in training the lane tracking and lane change tasks. To be noticed, the episode reward of learning lane change policy remains a low value before $5,000$ episodes. The reason is that the agent will explore the action space at the beginning to maximize the entropy of action probability. The exploration strategy makes the low-level skill model robust.

D. Learning High-level Cooperative Policy

After training the low-level skills in the single-vehicle environment, we gradually increase the number of vehicles to learn the coordination policy. As shown in Fig. 9, we set up four vehicles in a double-lane track scenario. Vehicle 4 is set with a plodding speed to simulate traffic congestion or traffic accident. The other vehicles are initialized with an average speed and random positions. Once the collision happens during lane-changing or lane-keeping, each vehicle will receive a negative reward of -20, and the episode will end and restart.

During the training, each vehicle will train opponent models to predict others' actions and use the opponent model to stabilize the offline training of the Q-value network. Fig. 10 shows the loss of modeling other vehicles' policies from vehicle 2's perspective. As we can see, the vehicle 1's action prediction model become converged at a fast speed while the

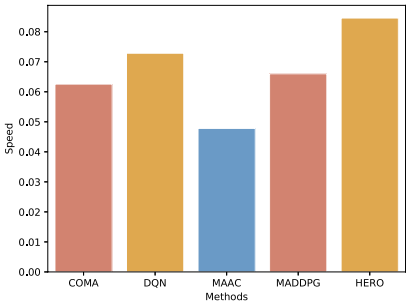


Fig. 11. Learning high-level cooperative policy in the simulation environment we created.

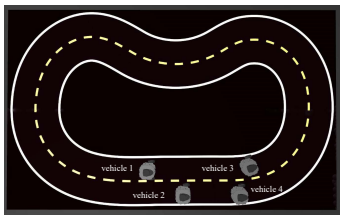


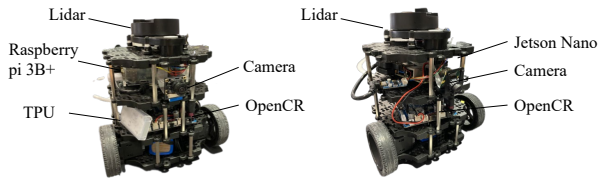
Fig. 12. Training in the Gazebo simulator.

vehicle 3' action prediction model become converged after 12000 episodes. The difference in convergence speed also illustrates the different interactions among vehicles.

Fig. 7 shows the performance of different RL approaches in terms of *mean episode reward, collision rate* and *lane change successful rate*. As shown in Fig. 7(a), our method achieved the highest episode reward than other baselines. Besides, the lower bound of our method is highest than others which also implicitly shows the stability of our approach. Fig. 7(b) and Fig. 7(c) shows the collision rate and lane change successful rate among different approaches. Almost all the RL approaches can reduce the collision rate after 14000 training episodes except MADDPG. We found that MADDPG has a lower learning efficiency as it still maintains a higher collision rate than others. Both DQN and our method reach the lowest collision rate. However, the lane change success rate is near 0. We found that vehicle 2 learn a policy to drive slowly instead of changing the lane during the training. Instead, the vehicle trained with our method can successfully perform cooperative lane change without collision with others. Moreover, Fig. 11 shows that our method achieves the highest mean speed, which is 0.08, and the vehicles trained with MAAC received the lowest speed of 0.048. In general, our method shows advantages in terms of *safety* and *efficiency*.

E. Real-world Experiments

To investigate the gap between simulation and reality, we deploy the learned policies in simulation to a real-world testbed, which consists of a two-lane track and multiple vehicles shown in Fig. 13. Each vehicle is equipped with a camera, lidar, and edge server. We run 20 episodes for each MRRL method with several random initial positions. Besides, we run a master node on the server to monitor the state of



(a) Smartbot-alpha (b) Smartbot-beta

Fig. 13. We develop two prototypes of vehicles in the real-world testbed. In this experiment, we use the first prototype in the testing.

TABLE II
PERFORMANCE EVALUATION IN REAL-WORLD PLATFORM

Method \ Metrics	Collision Rate	Successful Rate	Mean Speed
COMA	0.35	0.65	0.06344
Independent DQN	1.0	0.0	0.05395
MAAC	0.25	0.65	0.0625
MADDPG	0.95	0.5	0.07029
Ours	0.2	0.8	0.072

each vehicle and calculate the collision rate, lane-merging successful rate, and mean speed.

As shown in Tab. II, our approach reaches a low collision rate and high speed than others, which is 0.2 and 0.07, respectively. Besides, the collision rate of MAAC is lower than other baselines such as COMA and MADDPG. However, the performance of Independent DQN is different in simulation. We argue that the diversity and robustness of DQN are poor than other policy-based approaches.

VI. Related Work

This section summarizes the previous works related to multi-agent reinforcement learning and hierarchical reinforcement learning in Sec. VI-A and Sec. VI-B, respectively.

A. Multi-agent Reinforcement Learning

Recent works on reinforcement learning (RL) have shifted from single-agent reinforcement learning to multi-agent reinforcement learning (MARL). MARL corresponds to the learning problem in multi-agent systems in which multiple agents learn simultaneously in a shared environment. Existing MARL approaches can be classified into centralized RL, centralized training with decentralized execution (CTDE), and distributed RL. Centralized RL trains a centralized network mapping from the state space to all the agents' joint action space. Such an approach suffers from the high dimension issue of joint state and action spaces, making it hard to be extended to large-scale scenarios.

Some previous works adopt the paradigm of centralized training with decentralized execution (CTDE). Lowe et al. proposed a multi-agent actor-critic (MADDPG) approach and applied it for cooperative navigation [26]. Foerster et al. proposed counterfactual policy gradient (COMA), using a counterfactual baseline to address the credit assignment problem in MARL [27]. Similar works of CTDE can also be found in flocking [8] and pathfinding [9]. However, the number of features of the centralized critic network needs to be scaled up linearly (in the best case) or exponentially (in the worse case) with the increases of the number of agents [10]. Besides, different agents may have different influences, and decentralized critics are expected to perform better in these scenarios.

Recent developments in DRL can be further divided into two types: independent RL and decentralized critics with decentralized actors. Independent RL is self-interested and does not consider the states and actions of other agents [28]. Independent RL has good scalability but poor cooperation performance. An alternative method is to use decentralized critics with decentralized actors(DTDE). For example, Zhang et al. proposed a general decentralized MARL framework [11] and Iqbal et al. proposed an actor-attention-critic approach for multi-agent cooperation [12]. However, these methods either rely on explicit information sharing or use end-to-end RL models, which are not adaptive to complex and complicated tasks. Moreover, these end-to-end RL models lack interpretability on the interaction among agents.

B. Hierarchical Reinforcement Learning

Hierarchical reinforcement learning (HRL) is a type of reinforcement learning (RL) that leverages the hierarchical structure of a given task learns a hierarchical policy [29] [30]. Dayan et al. developed the first HRL work and proposed feudal reinforcement learning (FRL), where the high-level *managers* learn how to set the tasks to the *sub-managers* while the sub-managers learn how to perform the sub-tasks [31]. In this way, a complex task can be divided into small sub-tasks that are easier to be solved [31]. Kulkarni et al. introduced *intrinsic reward* design for low-level tasks and used a deep neural network to approximate the value function [22]. The proposed intrinsic reward mechanism motivates the agents to explore new behavior for its own sake.

Some previous works have applied hierarchical reinforcement learning for multi-agent cooperation. For example, Dietterich proposed the MAXQ framework to learn the hierarchical policy of each agent [32]. Besides, Ghavamzadeh et al. proposed a COM-Cooperative HRL for multi-agent communication [33]. Recently, Ahilan et al. proposed a deep MAHRL approach, in which a central manager policy chooses subtasks for other workers simultaneously [34]. Similar deep MAHRL approaches can also be found in [35]. However, all of these works either only consider the discrete action space of each agent or rely on centralized training.

VII. Conclusion and Future Directions

This paper studies how distributed multiple agents learn to cooperate in continuous action space and the non-stationarity issue. Unlike the traditional end-to-end RL methods, we propose a hierarchical reinforcement learning approach for distributed multi-agent cooperation. The agent cooperation is efficiently learned in high-level discrete action space, while the low-level individual control is handled by independent reinforcement learning. We incorporate *opponent modeling* in

the high-level layer to encourage cooperation and stabilize Q-learning. Then, we present a case study of cooperative lane change and conduct extensive experiments via simulation and a real-world testbed. In the future, we will investigate deeper hierarchy discovery and the theoretical guarantee. Currently, the design of task decomposition and state-action space in different layers still requires human knowledge. It is still an open question how to discover the hierarchical architecture automatically. Besides, most existing MARL approaches are trained and evaluated in simulation. The gap between simulation and reality also deserves more investigation.

VIII. Acknowledgment

The research is supported by Hong Kong RGC TRS (project T41-603), RIF (projects R5009-21 and R5060-19), CRF (projects C5026-18G and C5018-20GF), and GRF (project 15204921).

References

[1] M. A. L. Silva, S. R. de Souza, M. J. F. Souza, and A. L. C. Bazzan, "A reinforcement learning-based multi-agent framework applied for solving routing and scheduling problems," *Expert Systems with Applications*, vol. 131, pp. 148–171, 2019.

[2] P. Zhao, S. Suryanarayanan, and M. G. Simoes, "An energy management system for building structures using a multi-agent decision-making control methodology," *IEEE Transactions on Industry Applications*, vol. 49, no. 1, pp. 322–330, 2012.

[3] S. Jiang, J. Liang, J. Cao, and R. Liu, "An ensemble-level programming model with real-time support for multi-robot systems," in *IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, 2016, pp. 1–3.

[4] L. Ma, Z. Wang, and H.-K. Lam, "Event-triggered mean-square consensus control for time-varying stochastic multi-agent system with sensor saturations," *IEEE Transactions on Automatic Control*, vol. 62, no. 7, pp. 3524–3531, 2016.

[5] Y. Sahni, J. Cao, and S. Jiang, "Middleware for multi-robot systems," in *Mission-oriented Sensor Networks and Systems: Art and Science*. Springer, 2019, pp. 633–673.

[6] J. Wang, J. Cao, M. Stojmenovic, M. Zhao, J. Chen, and S. Jiang, "Pattern-rl: Multi-robot cooperative pattern formation via deep reinforcement learning," in *18th IEEE International Conference on Machine Learning And Applications (ICMLA)*, 2019, pp. 210–215.

[7] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "A survey and critique of multiagent deep reinforcement learning," *Autonomous Agents and Multi-Agent Systems*, vol. 33, no. 6, pp. 750–797, 2019.

[8] P. Zhu, W. Dai, W. Yao, J. Ma, Z. Zeng, and H. Lu, "Multi-robot flocking control based on deep reinforcement learning," *IEEE Access*, vol. 8, pp. 150 397–150 406, 2020.

[9] M. Damani, Z. Luo, E. Wenzel, and G. Sartoretti, "Primal _2: Pathfinding via reinforcement and imitation multi-agent learning-lifelong," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2666–2673, 2021.

[10] X. Lyu, Y. Xiao, B. Daley, and C. Amato, "Contrasting centralized and decentralized critics in multi-agent reinforcement learning," in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. ACM, 2021, pp. 844–852.

[11] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Basar, "Fully decentralized multi-agent reinforcement learning with networked agents," in *International Conference on Machine Learning (ICML)*, 2018, pp. 5872–5881.

[12] S. Iqbal and F. Sha, "Actor-attention-critic for multi-agent reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2019, pp. 2961–2970.

[13] C. Qu, S. Mannor, H. Xu, Y. Qi, L. Song, and J. Xiong, "Value propagation for decentralized networked deep multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 1182–1191.

[14] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *International Conference on Learning Representations (ICLR) Posters*, 2016.

[15] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2000, pp. 1057–1063.

[16] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, 2008.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[18] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International Conference on Machine Learning (ICML)*, 2014, pp. 387–395.

[19] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations (ICLR) Posters*, 2016.

[20] S. Pateria, B. Subagdja, A.-h. Tan, and C. Quek, "Hierarchical reinforcement learning: A comprehensive survey," *ACM Computing Surveys*, vol. 54, no. 5, pp. 1–35, 2021.

[21] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[22] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 29, 2016, pp. 3675–3683.

[23] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine Learning Proceedings (ICML)*. Elsevier, 1994, pp. 157–163.

[24] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning (ICML)*, 2018, pp. 1861–1870.

[25] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, 2004, pp. 2149–2154.

[26] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 6379–6390.

[27] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2018, pp. 2974–2982.

[28] K. Sivanathan, B. Vinayagam, T. Samak, and C. Samak, "Decentralized motion planning for multi-robot navigation using deep reinforcement learning," in *International Conference on Intelligent Sustainable Systems (ICISS)*. IEEE, 2020, pp. 709–716.

[29] T. G. Dietterich, "The maxq method for hierarchical reinforcement learning," in *International Conference on Machine Learning (ICML)*, vol. 98. Citeseer, 1998, pp. 118–126.

[30] N. K. Jong and P. Stone, "Hierarchical model-based reinforcement learning: R-max+ maxq," in *International Conference on Machine Learning (ICML)*, 2008, pp. 432–439.

[31] P. Dayan and G. E. Hinton, "Feudal reinforcement learning," in *Advances in Neural Information Processing Systems (NeurIPS)*. Morgan Kaufmann, 1992, pp. 271–278.

[32] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *Journal of Artificial Intelligence Research*, vol. 13, pp. 227–303, 2000.

[33] M. Ghavamzadeh and S. Mahadevan, "Learning to communicate and act using hierarchical reinforcement learning," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2004, pp. 1114–1121.

[34] S. Ahilan and P. Dayan, "Feudal multi-agent hierarchies for cooperative reinforcement learning," in *Multidisciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, 2019, p. 57.

[35] J. Chakravorty, P. N. Ward, J. Roy, M. Chevalier-Boisvert, S. Basu, A. Lupu, and D. Precup, "Option-critic in cooperative multi-agent systems," in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2020, pp. 1792–1794.