Autonomous Platoon Control With Integrated Deep Reinforcement Learning and Dynamic Programming

Tong Liu, Lei Lei, Senior Member, IEEE, Kan Zheng, Senior Member, IEEE, and Kuan Zhang, Member, IEEE

Abstract—Autonomous vehicles in a platoon determine the control inputs based on the system state information collected and shared by the Internet of Things (IoT) devices. Deep reinforcement learning (DRL) is regarded as a potential method for car-following control and has been mostly studied to support a single following vehicle. However, it is more challenging to learn an efficient car-following policy with convergence stability when there are multiple following vehicles in a platoon, especially with unpredictable leading vehicle behavior. In this context, we adopt an integrated DRL and dynamic programming (DP) approach to learn autonomous platoon control policies, which embeds the deep deterministic policy gradient (DDPG) algorithm into a finite-horizon value iteration framework. Although the DP framework can improve the stability and performance of DDPG, it has the limitations of lower sampling and training efficiency. In this article, we propose an algorithm, namely, finite-horizon-DDPG with sweeping through reduced state space using stationary approximation (FH-DDPG-SS), which uses three key ideas to overcome the above limitations, i.e., transferring network weights backward in time, stationary policy approximation for earlier time steps, and sweeping through reduced state space. In order to verify the effectiveness of FH-DDPG-SS, simulation using real driving data is performed, where the performance of FH-DDPG-SS is compared with those of the benchmark algorithms. Finally, platoon safety and string stability for FH-DDPG-SS are demonstrated.

Index Terms—Deep reinforcement learning (DRL), dynamic programming (DP), platoon control.

I. INTRODUCTION

A key application scenario of Internet of Vehicles (IoV) [1], vehicle platooning has progressed dramatically thanks to the development of cooperative adaptive cruise control (CACC) and advanced Internet of Things (IoT) technologies in various fields [2], [3], [4], [5], [6], [7]. Autonomous platoon control aims to determine the control inputs for the following autonomous vehicles so that all the vehicles move at the same speed while maintaining the desired distances between each pair of preceding and following vehicles. The autonomous vehicles are considered to have L3–L5 driving automation as defined by the society of automotive engineers (SAEs) classification standard [8]. A well-designed platoon controller is able to increase road capacity, reduce fuel consumption, as well as enhance driving safety and comfort [9], [10].

Platoon controllers have been proposed based on classical control theory, such as linear controller, $H_{\infty}$ controller, and sliding-mode controller (SMC) [9], [11], [12]. Platoon control is essentially a sequential stochastic decision problem (SSDP), where a sequence of decisions has to be made over a specific time horizon for a dynamic system whose states evolve in the face of uncertainty. To solve such an SSDP problem, a few existing works relied on the model predictive control (MPC) method [10], [13], [14], [15], [16], [17], where the trajectories of the leading vehicles are predicted by a model.

Although the MPC controller provides some safety guarantees to the control policy, the control performance is still restricted by the accuracy of the model itself. As another promising method, reinforcement learning (RL) can learn an optimal control policy directly from experience data by trial and error without requiring the stochastic properties of the underlying SSDP model [18], [19], [20]. Moreover, the more powerful deep RL (DRL) methods can deal with the curse-of-dimensionality problem of RL by approximating the value functions as well as policy functions using deep neural networks [21], [22]. In recent years, research on DRL has made significant progress and many popular DRL algorithms have been proposed, including value-based methods, such as deep Q network (DQN) [23] and double DQN [24]; and actor–critic methods such as deep deterministic policy gradient (DDPG) [25], asynchronous advantage actor–critic (A3C) [26], and trust region policy optimization (TRPO) [27]. The RL/DRL algorithms have been applied to solve the platoon control problem in a few recent literature [28], [29], [30], [31], [32], [33], [34], [35], [36], [37].

To elaborate, most of the contributions have addressed the car-following control problem of supporting a single following vehicle. An adaptive proportional–integral (PI) controller is presented in [29] whose parameters are tuned based on the state of the vehicle according to the control policy learned by actor–critic with kernel machines. However, the PI controller...
needs to predefine the candidate set of parameters before learning. In order to avoid this problem, an adaptive controller with parameterized batch actor–critic is proposed in [30]. A few works improve the RL/DRL-based control policy by modeling/predicting the leading vehicle (leader)’s behavior [31], [32]. In [31], a predictive controller based on the classical DRL algorithm DDPG [25] is presented as an alternative to the MPC controller, which uses advanced information about future speed reference values and road grade changes. The human driving data has been used in [33] and [34] to help RL/DRL achieve improved performance. In [35], the DDPG is applied for car-following control problem taking into account the acceleration-related delay, where the leader is assumed to drive at a constant speed. The proposed algorithm is used for comparing the performance of DRL and MPC for adaptive cruise control (ACC)-based car-following control problems in [36]. It is shown that DDPG has more advantages over MPC in the presence of uncertainties. In [37], a deterministic promotion RL method is proposed to improve training efficiency, where the direction of action exploration is evaluated by a normalization-based function and the evaluated direction works as a model-free search guide in return.

Meanwhile, DRL-based platoon control with multiple following vehicles has only been studied in a few recent works [38], [39], [40]. Based on predecessor-leader following (PLF) topology, a CACC-based control algorithm using DDPG is proposed in [38]. While DDPG is the most widely used algorithm in the existing DRL-based car-following controllers [31], [34], [35], [36], [38], it is shown that although DDPG performs well in the single following vehicle system, it is more difficult to learn an efficient control policy with convergence stability in a platoon system where there are multiple following vehicles and unpredictable leading vehicle behavior [39], [40]. To address this problem, the DDPG-based technique is invoked in [39] for determining the parameters of the optimal velocity model (OVM) instead of directly determining the accelerations. Meanwhile, Yan et al. [40] proposed a hybrid car-following strategy (HCFS) that selects the best actions derived from the DDPG controller and the linear controller, which is used to determine vehicle acceleration in the platoon. By combining with the classical control solutions, the performances are improved in [39] and [40]. However, the classical controllers also limit the performance of the above solutions, especially in the complex driving environment with random disturbance and nonlinear system dynamics.

In this context, we adopt an integrated DRL and dynamic programming (DP) approach to improve the convergence stability and performance of the DDPG-based platoon control policy without resorting to the help of the classical controllers. Specifically, we propose an algorithm that builds upon the finite-horizon DDPG (FH-DDPG) algorithm that was applied for the energy management of microgrids [41]. FH-DDPG addresses the unstable training problem of DDPG in a finite-horizon setting by using two key ideas: 1) backward induction and 2) time-dependent actors/critics. The DDPG algorithm is embedded into a finite-horizon value iteration framework, and a pair of actor and critic networks are trained for each time step by backward induction. It has been demonstrated in [41] that compared with DDPG, FH-DDPG is much more stable and achieves better performance.

However, FH-DDPG also suffers from some limitations that can be considered as the “side-effects” of its DP framework, i.e., low sampling efficiency and training efficiency. First, since FH-DDPG has to train $K$ actor and critic networks for a finite-horizon problem with $K$ time steps, the sampling efficiency of FH-DDPG is $1/K$ that of DDPG. Specifically, for $E$ episodes of training experience, the actor and critic networks of DDPG are trained with $E K$ data entries, while each pair of the $K$ actor and critic networks in FH-DDPG is only trained with $E$ data entries at the corresponding time step. Second, FH-DDPG has to sweep through the entire state space when training the actor and critic networks at each time step. As a result, the exhaustive sweeps approach considers a large portion of the inconsequential states, resulting in many wasted training updates.

To address the above two limitations in FH-DDPG and improve the sampling and training efficiency, we use three key ideas in our proposed DRL algorithm for platoon control, namely, FH-DDPG with sweeping through reduced state space using stationary policy approximation (FH-DDPG-SS). The contributions of this article are itemized next and an approach summary of related works on DRL-based platoon controller design is given in Table I to describe the characteristics of existing methods and highlight the contributions of our proposed algorithm.

1) To overcome the first limitation of FH-DDPG, i.e., the low sampling efficiency, we propose two key ideas, namely, transferring network weights backward in time and stationary policy approximation for earlier time steps. The first key idea is to transfer the trained actor and critic network weights at time step $k + 1$ to the initial network weights at time step $k$. The second key idea is using FH-DDPG to train the actors and critics from time steps $K$ to $m + 1$, and then train a single pair of actor and critic networks using DDPG from time steps $1$ to $m$, where the initial target network weights are set to the trained actor and critic network weights at time step $m + 1$.

2) To address the second limitation of FH-DDPG, i.e., the wasteful updates due to exhaustive sweeps, we propose the third key idea, namely, sweeping through reduced state space. Specifically, we train and test a “kick-off” policy to obtain a more refined state space. By sweeping through the reduced state space, the training efficiency and performance are enhanced as agents can focus on learning the states that good policies frequently visit.

3) To implement the above three key ideas, the FH-DDPG-SS algorithm is proposed to combine and integrate the three improvements for FH-DDPG.

The remainder of this article is organized as follows. The system model is introduced in Section II. Section III formulates the SSDP model for platoon control. The proposed DRL algorithms to solve the SSDP model are presented in Section IV. In Section V, the performance of FH-DDPG-SS is compared with those of the benchmark algorithms by
simulation. Moreover, platoon safety and string stability are demonstrated. Section VI concludes this article.

II. SYSTEM MODEL

We consider a platoon control problem with a number of \( N > 2 \) vehicles, i.e., \( \mathcal{V} = \{0, 1, \ldots, N-1\} \), wherein the position, velocity, and acceleration of a following vehicle (follower) \( i \in \mathcal{V} \backslash \{0\} \) at time \( t \) are denoted by \( p_i(t) \), \( v_i(t) \), and \( \text{acc}_i(t) \), respectively. Here, \( p_i(t) \) represents the 1-D position of the center of the front bumper of vehicle \( i \). Each follower \( i \) is manipulated by a distributed car-following policy of a DRL controller with Vehicle-to-Everything (V2X) communications.

Each vehicle \( i \in \mathcal{V} \) obeys the dynamics model described by a first-order system

\[
\begin{align*}
\dot{p}_i(t) &= v_i(t) \quad (1) \\
\dot{v}_i(t) &= \text{acc}_i(t) \quad (2) \\
\ddot{\text{acc}}_i(t) &= -\frac{1}{\tau_i} \text{acc}_i(t) + \frac{1}{\tau_i} u_i(t) \quad (3)
\end{align*}
\]

where \( \tau_i \) is a time constant representing driveline dynamics and \( u_i(t) \) is the vehicle control input (commanded acceleration) at time \( t \). In order to ensure driving safety and comfort, the following constraints are applied:

\[
\begin{align*}
\text{acc}_{\text{min}} \leq \text{acc}_i(t) &\leq \text{acc}_{\text{max}} \quad (4) \\
\text{u}_{\text{min}} \leq u_i(t) &\leq \text{u}_{\text{max}} \quad (5)
\end{align*}
\]

where \( \text{acc}_{\text{min}} \) and \( \text{acc}_{\text{max}} \) are the acceleration limits, while \( \text{u}_{\text{min}} \) and \( \text{u}_{\text{max}} \) are the control input limits.

The headway of follower \( i \) at time \( t \), i.e., bumper-to-bumper distance between follower \( i \) and its preceding vehicle (predecessor) \( i-1 \), is denoted by \( d_i(t) \) with

\[
d_i(t) = p_{i-1}(t) - p_i(t) - L_{i-1} \quad (6)
\]

where \( L_{i-1} \) is the body length of vehicle \( i-1 \).

We adopt the constant time-headway policy (CTHP) in this article, i.e., follower \( i \) aims to maintain a desired headway \( d_{e,i}(t) \), which satisfies

\[
d_{e,i}(t) = r_i + h_i v_i(t) \quad (7)
\]

where \( r_i \) is a standstill distance for safety of follower \( i \) and \( h_i \) is a constant time gap of follower \( i \) which represents the time that it takes for follower \( i \) to bridge the distance in between the vehicles \( i \) and \( i-1 \) when continuing to drive with a constant velocity.

The control errors, i.e., gap-keeping error \( e_p(t) \) and velocity error \( e_v(t) \) of follower \( i \), are defined as

\[
\begin{align*}
e_p(t) &= d_i(t) - d_{e,i}(t) \quad (8) \\
e_v(t) &= v_{i-1}(t) - v_i(t) \quad (9)
\end{align*}
\]

III. SSDP MODEL FOR PLATOON CONTROL

An SSDP can be formulated to determine the vehicle’s control action. The time horizon is discretized into time intervals of length \( T \) seconds (s), and a time interval \( \{kT, (k+1)T\} \) amounts to a time step \( k = 1, 2, \ldots, K \), where \( K \) is the total number of time steps. In the remainder of this article, we will use \( x_k := x((k-1)T) \) to represent any variable \( x \) at time \( (k-1)T \).

In the following, the state space, action space, system dynamics model, and reward function of the SSDP model are presented, respectively.

A. State Space

At each time step \( k \in \{1, 2, \ldots, K\} \), the controller of follower \( i \) determines the vehicle control input \( u_{i,k} \), based on the observations of the system state. \( v_{i,k} \) and \( \text{acc}_{i,k} \) can be measured locally, while \( e_{p,i,k} \) and \( e_{v,i,k} \) can be measured through a radar unit mounted at the front of the vehicle. Thus, the state that the follower \( i \) can obtain locally is denoted by \( x_{i,k} = [e_{p,i,k}, e_{v,i,k}, \text{acc}_{i,k}]^T \).

Additionally, the follower \( i \) can obtain the driving status \( x_{j,k} \) and control input \( u_{j,k} \) of the other vehicles \( j \in \mathcal{V} \backslash \{i\} \) via V2X communications. In this article, we adopt the predecessors following (PF) information topology, in which \( \text{acc}_{j-1,k} \) and \( u_{j-1,k} \) are transmitted to the follower \( i \in \mathcal{V} \backslash \{0\} \). It has been proved in our previous work that the optimal policy of the SSDP model with the preceding vehicle’s acceleration \( \text{acc}_{j-1,k} \) and control input \( u_{j-1,k} \) in the state performs at least as well as the optimal policy for the SSDP model which does not include this information [5]. Thus, the system state for the follower \( i \) is denoted as \( S_{i,k} = [v_{i,k}, \text{acc}_{i-1,k}, u_{i-1,k}]^T \). The state space is \( \mathcal{S} = \{S_{i,k} | e_{p,i,k}, e_{v,i,k} \in [-\infty, \infty], \text{acc}_{i,k}, \text{acc}_{i-1,k} \in \text{[acc}_{\text{min}}, \text{acc}_{\text{max}}], u_{i-1,k} \in [\text{u}_{\text{min}}, \text{u}_{\text{max}}] \} \).

### TABLE I

**APPROACH SUMMARY OF RELATED WORKS ON RL/DRL-BASED PLATOON CONTROLLER DESIGN**

| Scenario                  | Approach | References | General description |
|---------------------------|----------|------------|---------------------|
| Single following vehicle  | RL/DRL   | [29]–[37] | DDPG is the most widely used DRL algorithm and performs well in the single following vehicle system. However, it is more difficult to learn an efficient control policy with convergence stability in a platoon system where there are multiple following vehicles and unpredictable leading vehicle behavior. |
| Multiple following vehicles| DDPG     | [38]       | By combining with the classical control solutions, the performance of DDPG is improved. However, the classical controllers also limit the performance of these solutions, especially in the complex driving environment with random disturbance and non-linear system dynamics. |
|                           | DDPG-OVM | [39]       | HCTS                                                             |
|                           | F1-DDPG-SS| Proposed   | We adopt an integrated DRL and DP approach to improve the convergence stability and performance of DDPG-based platoon control policy without resorting to the help of the classical controllers. We also use three key ideas to overcome the limitations of the DP framework and improve the sampling efficiency and training efficiency. |
B. Action Space

The control input $u_{i,k}$ of the follower $i \in \mathcal{V}\setminus\{0\}$ is regarded as the action at the time step $k$.

The action space is $\mathcal{A} = \{u_{i,k}, |u_{i,k}| \in [u_{\text{min}}, u_{\text{max}}]\}$.

C. System Dynamics Model

The system dynamics are derived in discrete time on the basis of forward Euler discretization. Note that for the leader 0, $e_{p,0,k} = e_{v,0,k} = 0$, thus the system dynamics model evolves in discrete time according to

$$x_{0,k+1} = A_0 x_{0,k} + B_0 u_{0,k}$$

where

$$A_0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 - \frac{T}{\tau_0} & 1 - \frac{T}{\tau_0} \end{bmatrix}, \quad B_0 = \begin{bmatrix} 0 \\ 0 \\ \frac{T}{\tau_0} \end{bmatrix}.$$  \hfill (11)

For the follower $i \in \mathcal{V}\setminus\{0\}$ in the platoon, we have

$$x_{i,k+1} = A_i x_{i,k} + B_i u_{i,k} + C_i \text{acc}_{i-1,k}$$

where

$$A_i = \begin{bmatrix} 1 & T & -h_i T \\ 0 & 1 & -T \\ 0 & 1 - \frac{T}{\tau_i} & 1 - \frac{T}{\tau_i} \end{bmatrix}, \quad B_i = \begin{bmatrix} 0 \\ 0 \\ \frac{T}{\tau_i} \end{bmatrix}, \quad C_i = \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix}.$$  \hfill (13)

D. Reward Function

The reward function can guide the optimization objectives and has an impact on the convergence of the DRL algorithm. Our objective is to minimize gap-keeping error $e_{p,i,k}$ and velocity error $e_{v,i,k}$ to achieve the platoon control target while penalizing control input $u_{i,k}$ to reduce the fuel consumption and the jerk to improve the driving comfort. Note that the jerk is the change rate in acceleration, which is given by

$$j_{i,k} = \frac{\text{acc}_{i,k+1} - \text{acc}_{i,k}}{T} = -\frac{1}{\tau_i} \text{acc}_{i,k} + \frac{1}{\tau_i} u_{i,k}$$

where the second equality is due to the forward Euler discretization of (3).

Although the quadratic cost function is normally adopted in optimal control problems, it is found that it does not work well for the DDPG algorithm as the sudden large changes of reward values often decrease its training stability. Therefore, an absolute-value cost function is adopted for DDPG in [35], [36], and [40] to improve its performance. However, we found that the absolute-value cost function could hinder the further performance improvement when the control errors are relatively small. Therefore, we design a Huber loss function [42] as the reward function for each follower $i \in \mathcal{V}\setminus\{0\}$, which is given by

$$R(S_{i,k}, u_{i,k}) = \begin{cases} r_{\text{abs}}, & \text{if } r_{\text{abs}} < \varepsilon \\ r_{\text{qua}}, & \text{if } r_{\text{abs}} \geq \varepsilon \end{cases}$$

where

$$r_{\text{abs}} = -\left\{ \frac{e_{p,i,k}}{\hat{e}_{\text{p, max}}} + a \frac{e_{v,i,k}}{\hat{e}_{v, \text{max}}} + b \frac{u_{i,k}}{u_{\text{max}}} + c \frac{j_{i,k}}{2 \text{acc}_{\text{max}}/T} \right\}$$

$$r_{\text{qua}} = -\lambda \left\{ (e_{p,i,k})^2 + a (e_{v,i,k})^2 + b (u_{i,k})^2 + c (j_{i,k}T)^2 \right\}$$

where $\varepsilon$ is the reward threshold, and $\hat{e}_{\text{p, max}}$ and $\hat{e}_{v, \text{max}}$ are the nominal maximum control errors such that it is larger than most possible control errors. $\lambda$ is the reward scale. $a$, $b$, and $c$ are the positive weights and can be adjusted to determine the relative importance of minimizing the gap-keeping error, the velocity error, the control input, and the jerk.

Thus, the expected cumulative reward $J_{\pi_i}$ of the follower $i$ over the finite-time horizon $K$ under a policy $\pi_i$ can be expressed as

$$J_{\pi_i} = \mathbb{E}_{\pi_i} \left[ \sum_{t=1}^{K} \gamma^{t-1} R(S_{i,t}, u_{i,t}) \right]$$

where $\gamma$ is the reward discount factor.

The ultimate objective is to find the optimal policy $\pi_i^*$ that maximizes the expected cumulative reward $J_{\pi_i}$, i.e.,

$$\pi_i^* = \arg \max_{\pi_i} J_{\pi_i}. \hfill (17)$$

IV. DRL Algorithms

To solve the above SSDP, we propose a DRL algorithm which improves on the FH-DDPG algorithm [41]. In the following, we will first provide a brief introduction to the FH-DDPG algorithm and then elaborate on the proposed improvements.

A. FH-DDPG

FH-DDPG is a combination of DRL and DP, where the DDPG algorithm is embedded into a finite-horizon value iteration framework. It is designed to solve finite-horizon SSDP and improve the stability of the DDPG algorithm.

DDPG is a well-known DRL algorithm widely applied to continuous control. It trains both a pair of actor and critic networks, i.e., $\mu(S_k|\theta^\mu)$ and $Q(S_k, u_k|\theta^Q)$, to derive the optimal policy $\pi^*(\mu|\theta^\mu)$ and the corresponding action-value (Q-value) $Q^*(S_k, u_k|\theta^Q)$, respectively [25]. The action-value is defined as the expected cumulative discounted reward from time step $k$: $Q(S_k, u_k|\theta^Q) = \mathbb{E}_\pi \left[ \sum_{\tau=k}^{\infty} \gamma^{\tau-k} R(S_{\tau}, u_{\tau}) \right]$. Experience replay is adopted in DDPG to enable stable and robust learning. When the replay buffer is full, the oldest sample will be discarded before a new sample is stored in the buffer. A minibatch from the buffer is sampled at each time step in order to update the actor and critic networks. DDPG creates a copy of the actor and critic networks as target networks, i.e., $\mu(S_{k+1}|\theta^\mu)$ and $Q(S_{k+1}, u_{k+1}|\theta^Q)$, to calculate the target values. The critic network is updated by minimizing the root-mean-square error (RMSE) $L_k = R(S_k, u_k) + \gamma Q(S_{k+1}, \mu^\prime(S_{k+1}|\theta^\mu)|\theta^Q) - Q(S_k, u_k|\theta^Q)$ using the sampled gradient descent with respect to $\theta^Q$. The actor network is updated by using the sampled deterministic policy gradient ascent on $Q(S_k, \mu(S_k|\theta^\mu)|\theta^Q)$ with respect to $\theta^\mu$. In order to improve the stability of learning, the weights of
these target networks are updated by soft target update, where the target networks are constrained to slowly track the learned networks.

DDPG is designed to solve the infinite-horizon SSDPs, where the actors and critics are the same for every time step. On the other hand, the optimal policies and the corresponding action values are normally time dependent in a finite-horizon setting [43]. Therefore, there are \( K \) actors and critics in FH-DDPG with \( K \) time steps. As shown in Fig. 1, FH-DDPG starts by having the myopic policy \( \mu_k^*(S_k) = \mu^\text{mo}(S_k) \) as the optimal policy with the terminal reward \( R_k \) for the final time step \( K \). And then, the finite horizon value iteration starts from time step \( K - 1 \) and uses backward induction to iteratively derive the optimal policy \( \mu_k^*(S_k|\theta_k) \) and the action-value \( Q_k^*(S_k, u_k|\theta_k) \) for each time step \( k \), until it reaches the first time step \( k = 1 \). In each time step, an algorithm similar to DDPG is adopted to solve a one-period MDP in which an episode only consists of two time steps. However, different from DDPG, the target actor network \( \mu_k^*(S_{k+1}|\theta_k) \) and critic network \( Q_k^*(S_{k+1}, u_{k+1}|\theta_k) \) of the current time step \( k \) are fixed and set as the trained actor network \( \mu_k(S_{k+1}|\theta_k) \) and critic network \( Q_k(S_{k+1}, u_{k+1}|\theta_k) \) of the next time step \( k + 1 \). This greatly increases the stability and performance of the algorithm. The pseudocode of the FH-DDPG algorithm is given in Appendix-A. Note that in each time step, the DDPG-FT function is used to train the respective actor and critic networks, where DDPG-FT is the abbreviation for DDPG with fixed targets.

B. Improving Sampling Efficiency

For a finite-time horizon consisting of \( K \) time steps, when both DDPG and FH-DDPG are trained for \( E \) episodes, a total of \( EK \) data entries are generated from the experience data as a data entry is generated per time step. Since DDPG has a single pair of actor and critic networks, all the \( EK \) data entries are used to train this pair of networks. Meanwhile, there are \( K \) pairs of actor and critic networks in FH-DDPG. For each time step \( k \), the corresponding pair of actor and critic networks is trained only with the \( E \) data entries of time step \( k \). Therefore, the sampling efficiency of FH-DDPG is only \( 1/K \) of that of DDPG. To improve sampling efficiency, two key ideas are proposed in the following.

1) Transferring Network Weights Backward in Time: In FH-DDPG, the actor \( \mu(S_k|\theta_k) \) and critic \( Q(S_k, u_k|\theta_k) \) at each time step \( k \) are trained with random initial parameters. Inspired by the parameter-transfer approach in transfer learning [44], we transfer the trained actor and critic network weights at time step \( k + 1 \), i.e., \( \theta^\mu_{k+1} \) and \( \theta^Q_{k+1} \) to the initial weights at time step \( k \), i.e., \( \theta^\mu_k \) and \( \theta^Q_k \), respectively. Thus, although \( \mu(S_k|\theta_k) \) and \( Q(S_k, u_k|\theta_k) \) are trained based on the \( E \) data entries of the \( E \) episodes at time step \( k \), the trainings are built upon the initial weights \( \theta^\mu_{k+1} \) and \( \theta^Q_{k+1} \), which were in turn trained based on the \( E \) data entries of the \( E \) episodes at time step \( k + 1 \) and built upon the initial weights \( \theta^\mu_{k+2} \) and \( \theta^Q_{k+2} \). In this way, the actor and critic networks at time step \( k \) are actually trained from the experiences of the \( E(K - k) \) data entries of the \( E \) episodes from time steps \( k \) to \( K \), instead of only the \( E \) data entries as in FH-DDPG. The proposed algorithm is given in Algorithm 1, namely, FH-DDPG with network weights transferring backward in time (FH-DDPG-NB).

![Fig. 1. FH-DDPG framework.](https://example.com/fig1.png)

**Algorithm 1** FH-DDPG-NB Algorithm

1: Randomly initialize actor and critic network weights as \( \theta^\mu_0 \) and \( \theta^Q_0 \)
2: Set \( \mu_0^*(S_k) = \mu_0^\text{mo}(S_k) \) for the final time step \( K \)
3: for \( k = K - 1, \ldots, 1 \) do
4: if \( k = K - 1 \) then
5: \( \theta^\mu_k = \theta^\mu_0 \) and \( \theta^Q_k = \theta^Q_0 \)
6: else
7: \( \theta^\mu_k = \theta^\mu_{k+1} \) and \( \theta^Q_k = \theta^Q_{k+1} \)
8: end if
9: \( \theta^\mu_{k+1} \), \( \theta^Q_{k+1} \leftarrow \text{DDPG - FT}(\theta^\mu_k, \theta^Q_k, \theta^\mu_k, \theta^Q_k, k) \)
10: Update the target network:
11: \( \theta^\mu_k \leftarrow \theta^\mu_{k+1} \), \( \theta^Q_k \leftarrow \theta^Q_{k+1} \)
12: end for
13: return \( \{\theta^\mu_{k+1}, \theta^Q_{k+1}\}^{K-1}_{k=1} \)
Algorithm 2 FH-DDPG-SA-(NB) Algorithm

1: Set the time horizon as \([m + 1, \ldots, K]\)
2: \([\theta_{\mu,i,k}, \theta_{Q,i,k}]_k=0^{K-1} \leftarrow \text{FH} - \text{DDPG}(-\text{NB})\)
3: Set the time horizon as \([1, \ldots, m]\)
4: Set the initial target networks weights with \(\theta_{\mu,i} = \theta_{\mu,i,m+1}\)
   and \(\theta_{Q,i} = \theta_{Q,i,m+1}\)
5: \(\theta_{\mu,i}, \theta_{Q,i} \leftarrow \text{DDPG}\)

2) Stationary Policy Approximation for Earlier Time Steps: Although the action values are generally time-dependent for the finite-horizon control problems, there is a vanishing difference in action-values when the horizon length is sufficiently large [45]. Taking the finite-horizon linear quadratic regulator (LQR) as an example, the action-value gradually converges to the steady-state value as the time step \(k\) decreases from the horizon \(K\). Moreover, for \(k\) not close to horizon \(K\), the LQR optimal policy is approximately stationary [46], [47]. Even though the platoon control problem is not an LQR problem in the strict sense, since both the system state \(S_{i,k}\) and action \(u_{i,k}\) have constraints and there are non-Gaussian random disturbances, we can observe similar trends in the learned policy by FH-DDPG. This allows us to improve sampling efficiency by first obtaining the time step threshold \(m\) such that the action-values and optimal policies are approximately constant and stationary when \(k \leq m\). And then, we adopt a single pair of actor and critic networks from time steps 1 to \(m\).

To elaborate, we first obtain \(m\) by solving the LQR problem for platoon control and analyzing the corresponding results, ignoring the state/action constraints and random disturbances. This enables us to determine the value of \(m\) in an efficient manner. Then, FH-DDPG is trained from time steps \(K\) to \(m + 1\). Next, instead of training a separate pair of actor and critic networks for each time step from 1 to \(m\), we train a single actor network \(\mu(S_{i,k}|\theta_{\mu})\) and critic network \(Q_i(S_{i,k}, u_{i,k}|\theta_{Q,i})\) for all the time steps \(k \in \{1, 2, \ldots, m\}\). Specifically, the actor and critic networks are trained using DDPG, where the initial values of the target networks are set to those of the trained actor and critic networks at time step \(m + 1\), i.e., \(\theta_{\mu,i,m+1}\) and \(\theta_{Q,i,m+1}\). The well-trained initial values for the target networks can significantly increase the stability and performance of the DDPG algorithm. In this way, the actor and critic networks are trained from the experiences of the \(E_m\) data entries of the \(E\) episodes from time steps 1 to \(m\). The proposed algorithm is given in Algorithm 2, namely, FH-DDPG with stationary policy approximation for earlier time steps (FH-DDPG-SA). The function FH-DDPG is realized by the FH-DDPG algorithm given in Appendix-A. Note that FH-DDPG-SA can be combined with FH-DDPG-NB, by adopting the latter algorithm instead of FH-DDPG to train the actor and critic networks from time steps \(K\) to \(m + 1\) in line 2 of the pseudocode. This will result in the FH-DDPG-SA-NB algorithm.

C. Sweeping Through Reduced State Space

FH-DDPG embeds DRL under the framework of DP, while the classical approach of DP is to sweep through the entire state space at each time step \(k\). This exhaustive sweeps approach leads to many wasteful updates during training, since many of the states are inconsequential as they are visited only under poor policies or with very low probability. An alternative approach is trajectory sampling which sweeps according to on-policy distribution [18]. Although trajectory sampling is more appealing, it is impossible to be adopted by FH-DDPG due to the latter’s backward induction framework.

Inspired by trajectory sampling, we improve FH-DDPG by sweeping through a reduced state space. Specifically, we first learn a relatively good kick-off policy by exhaustive sweeps, and then obtain a reduced state space by testing the kick-off policy, and finally continue to train the policy by sweeping through the reduced state space to further improve the performance. This approach can help agents to focus on learning the states that good policies frequently visit, which improves training efficiency. Note that a good policy in RL should achieve a large expected cumulative reward \(J_x\), as given in (18). For example, in platoon control, the control errors are normally small under good policies as the ends of the training episodes are approached, and it is not necessary to sweep through large control error states.

For platoon control, although the theoretical bounds of gap-keeping error \(e_{p,i,k}\) and velocity error \(e_{v,i,k}\) are infinity, it is impossible to sweep through an infinite range when training. Therefore, we need to restrict sweeping to a finite range at first. In practice, there are some empirical limits for \(e_{p,i,k}\) and \(e_{v,i,k}\) for a reasonable platoon control policy. Since we consider that FH-DDPG is trained from scratch, some relatively large control error states could be visited during training due to the random initial policy and exploration. Therefore, we first sweep through a relatively large state space, i.e.,

\[
S^{\text{ls}}_{i,k} = \left\{ (e_{p,i,k}, e_{v,i,k}, acc_{i,k})^T \mid e_{p,i,k} \in [-e_{p,\text{max}}, e_{p,\text{max}}], e_{v,i,k} \in [-e_{v,\text{max}}, e_{v,\text{max}}], \right. \\
\left. acc_{i,k} \in [acc_{\text{min}}, acc_{\text{max}}] \right\}
\]

(18)

where \(e_{p,\text{max}}\) and \(e_{v,\text{max}}\) are the same for each time step and are larger than most control errors during the training of FH-DDPG. Thus, we first train FH-DDPG in the state space \(S^{\text{ls}}_{i,k}\) to learn a kick-off policy \(\hat{\mu}_{i,k}(S_{i,k}|\theta_{\mu,i,k})\) for the follower \(i\) at time step \(k\), and then obtain the upper and lower bounds of a more refined state space, i.e.,

\[
S^{\text{rs}}_{i,k} = \left\{ (e_{p,i,k}, e_{v,i,k}, acc_{i,k})^T \mid e_{p,i,k} \in [\bar{e}_{p,i,k,\text{min}}, \bar{e}_{p,i,k,\text{max}}], e_{v,i,k} \in [\bar{e}_{v,i,k,\text{min}}, \bar{e}_{v,i,k,\text{max}}], \right. \\
\left. acc_{i,k} \in [\bar{acc}_{i,k,\text{min}}, \bar{acc}_{i,k,\text{max}}] \right\}
\]

(19)

for the follower \(i\) at time step \(k\) by testing \(\hat{\mu}_{i,k}(S_{i,k}|\theta_{\mu,i,k})\). Next, the actor network \(\hat{\mu}_{i,k}(S_{i,k}|\theta_{\mu,i,k})\) and critic network \(\hat{Q}_{i,k}(S_{i,k}, u_{i,k}|\theta_{Q,i,k})\) are further trained by FH-DDPG, which only sweeps through \(S^{\text{rs}}_{i,k}\).

Combining the above three improvements for FH-DDPG, we propose a novel DRL algorithm, namely, FH-DDPG with
Algorithm 3 FH-DDPG-SS Algorithm

1: Set $S_{ls}$ according to (18)
2: $\{\hat{\theta}_i^{h, k}, \hat{Q}_i^{h, k}\}_{k=1}^{K-1} \leftarrow \text{FH} - \text{DDPG} - \text{SA} - \text{NB}(S_{ls})$
3: for $g = 1, \ldots, G$ do
4: Test $\{\hat{\mu}_i^{h, k}(S_{ls}^{h, k})\}_{k=1}^{K-1}$
5: Store $\{\hat{\phi}_i^{h, k}(S_{ls}^{h, k})\}_{k=1}^{K-1} = [\hat{\phi}_i^{h, k}, \hat{\phi}_i^{h, k}, \hat{\phi}_i^{h, k}]$ in $E_{i,k}^{h}$
6: end for
7: for $k = 1, \ldots, K - 1$ do
8: Find the upper and lower bounds $\bar{\tau}_i^{h, k}, \bar{\tau}_i^{h, k}, \bar{\tau}_i^{h, k}, \bar{\tau}_i^{h, k}, \bar{\tau}_i^{h, k}$ in $E_{i,k}^{h}$
9: Set $S_{ls}^{h, k}$ according to (19)
10: end for
11: Set the initial actor and critic network weights as $\{\hat{\theta}_i^{h, k}, \hat{Q}_i^{h, k}\}_{k=1}^{K-1}$
12: $\{\hat{\theta}_i^{h, k}, \hat{Q}_i^{h, k}\}_{k=1}^{K-1} \leftarrow \text{FH} - \text{DDPG} - \text{SA}(S_{ls}^{h, k})$

V. Experimental Results

In this section, we present the simulation results of the proposed FH-DDPG-SS algorithm as well as benchmark DRL algorithms, i.e., DDPG, FH-DDPG, and HCFS [40].

A. Experimental Setup

All the algorithms are trained/tested using the open-source data given in [34]. Specifically, the data was extracted from the Next Generation Simulation (NGSIM) data set [48], which was retrieved from the eastbound I-80 in the San Francisco Bay area in Emeryville, CA, USA, on 13 April 2005, at a sampling rate of 10 Hz, with 45 min of precise location data available in the full data set. Then, car-following events were extracted by applying a car-following filter as described in [49], where the leading and following vehicle pairs of each event stay in the same lane. In our experiment, we used 1000 car-following events. Moreover, although there are data for both the leading vehicle and the following vehicle in the data set, we only used the velocity data of the leading vehicle to simulate the real-world environment with uncertainty, so that the DRL algorithms can be trained and evaluated. 80% of the data (i.e., 800 car-following events) is used for training and 20% (i.e., 200 car-following events) is used for testing. The platoon control environment and the DRL algorithms are implemented in Tensorflow 1.14 using Python [50]. We compare the performance of the proposed FH-DDPG-SS algorithm with the benchmark algorithms in terms of the average cumulative reward. Moreover, the platoon safety and string stability performance for FH-DDPG-SS are also demonstrated. To ensure a fair comparison, all the algorithms are trained and tested in the same environment with the same reward function.

The technical constraints and operational parameters of the platoon control environment are given in Table II. In general, the parameters of the platoon environment and state/action space in Table I are determined using the values reported in [35] and [36] as a reference. The interval for each time step is set to $T = 0.1$ s, and each episode is composed of 100 time steps (i.e., $K = 100$) with a duration of 10 s. As the number of vehicles set in the existing literature on platoon control ranges from 3 to 8 [11], [13], [16], [17], [38], [39], [40], we set the number of vehicles to $N = 5$. We initialize the states for each of four followers with $x_{i,0} = [1.5, -1.0], \forall i \in \{1, 2, 3, 4\}$. We experiment with several sets of reward coefficients $a, b$, and $c$ to balance the minimization of gap-keeping error $e_{p,i,k}$, velocity error $e_{v,i,k}$, control input $u_{i,k}$, and jerk $j_{i,k}$. In addition, the absolute-value cost function can improve the algorithm’s convergence stability when control errors are large, and the square-value cost function is conducive to further performance improvement when control errors are small. Thus, we determine the reward threshold $e$ with the best performance for all algorithms by a simple grid search. The nominal maximum control errors in the reward function (15) are set to $\hat{e}_{p,i,k} = 15$ m and $\hat{e}_{v,i,k} = 10$ m/s so that it is larger than most possible control errors during training for all DRL algorithms. For the FH-DDPG-SS algorithm, the time step threshold $m = 11$. Moreover, the maximum gap-keeping error $e_{p,\text{max}}$ and maximum velocity error $e_{v,\text{max}}$ in (18) are set to

| Table II | Technical Constraints and Operational Parameters of the Platoon Control Environment |
|----------------------------------------|------------------------------------------|
| Notations | Description | Values |
| Platoon environment | $T$ | Interval for each time step | 0.1 s |
| | $K$ | Total time steps in each episode | 100 |
| | $m$ | Time step threshold | 11 |
| | $N$ | Number of vehicles | 5 |
| | $\tau_i$ | Driveline dynamics time constant | 0.1 s |
| | $h_i$ | Time gap | 1 s |
| State & action | $e_{p,\text{max}}$ | Maximum gap-keeping error | 2 m |
| | $e_{v,\text{max}}$ | Maximum velocity error | 1.5 m/s |
| | $g_{\text{acc}}^{\text{min}}$ | Minimum acceleration | $-2.6$ m/s$^2$ |
| | $g_{\text{acc}}^{\text{max}}$ | Maximum acceleration | 2.6 m/s$^2$ |
| | $u_{\text{min}}$ | Minimum control input | $-2.6$ m/s$^2$ |
| | $u_{\text{max}}$ | Maximum control input | 2.6 m/s$^2$ |
| Reward function | $a$ | Reward coefficient | 0.1 |
| | $b$ | Reward coefficient | 0.1 |
| | $c$ | Reward coefficient | 0.2 |
| | $\hat{e}_{p,\text{max}}$ | Nominal maximum gap-keeping error | 15 m |
| | $\hat{e}_{v,\text{max}}$ | Nominal maximum velocity error | 10 m/s |
| | $e$ | Reward threshold | $-0.4483$ |

sweeping through reduced state space using stationary policy approximation (FH-DDPG-SS), which is given in Algorithm 3. Note that the overall procedure of FH-DDPG-SS is the same as that described in Section IV-C, except that in lines 2 and 12, the FH-DDPG-SA-NB and FH-DDPG-SA algorithms are adopted instead of the FH-DDPG algorithm to incorporate the improvements in Algorithms 1 and 2. The reason why we use FH-DDPG-SA instead of FH-DDPG-SA-NB in line 12 is that the initial actor and critic networks weights for all the time steps are carried over from the previous training, so we no longer need to transfer network weights backward in time.
TABLE III
HYPERPARAMETERS OF THE DRL ALGORITHMS FOR TRAINING

| Parameter                          | Value             |
|------------------------------------|-------------------|
| Actor network size                 | DDPG: 256, 128    |
|                                    | FH-DDPG: 400, 300, 100 |
|                                    | FH-DDPG-SS: 400, 300, 100 |
| Critic network size                | DDPG: 256, 128    |
|                                    | FH-DDPG: 400, 300, 100 |
|                                    | FH-DDPG-SS: 400, 300, 100 |
| Actor activation function          | DDPG: relu, relu, tanh |
|                                    | FH-DDPG: relu, relu, tanh |
|                                    | FH-DDPG-SS: relu, relu, relu, tanh |
| Critic activation function         | DDPG: relu, relu, linear |
|                                    | FH-DDPG: relu, relu, linear |
|                                    | FH-DDPG-SS: relu, relu, relu, linear |
| Actor learning rate $\alpha$       | DDPG: $1e^{-4}$   |
|                                    | FH-DDPG: $1e^{-4}$ |
|                                    | FH-DDPG-SS: $1e^{-4}$ |
| Critic learning rate $\beta$       | DDPG: $1e^{-3}$   |
|                                    | FH-DDPG: $1e^{-3}$ |
|                                    | FH-DDPG-SS: $1e^{-3}$ |
| Total training episodes $E$        | DDPG: 5000        |
|                                    | FH-DDPG: 5000     |
|                                    | FH-DDPG-SS: 3000, 2000 |
| Batch size $N_b$                   | DDPG: 250000      |
|                                    | FH-DDPG: 25000    |
|                                    | FH-DDPG-SS: 2500, 2000 |
| Reward scale $\lambda$            | DDPG: 5e-3        |
| Reward discount factor $\gamma$    | DDPG: 1           |
| Soft target update $\eta$          | DDPG: 0.001      |
|                                    | FH-DDPG: /        |
|                                    | FH-DDPG-SS: 0.01  |
| Noise type                         | DDPG: Ornstein-Uhlenbeck Process with $\theta = 0.15$ and $\sigma = 0.5$ |
| Final layer weights/biases         | DDPG: Random uniform distribution $[-3 \times 10^{-3}, 3 \times 10^{-3}]$ |
| initialization                     | FH-DDPG-SS: Random uniform distribution $[-\frac{1}{\sqrt{f}}, \frac{1}{\sqrt{f}}]$ (f is the fan-in of the layer) |

2 m and 1.5 m/s, respectively. Additionally, to reduce large oscillations in $u_{i,k}$ and $\ddot{c}_{i,k}$, we set $j_{i,k}$ in FH-DDPG and FH-DDPG-SS in the testing phase within $[-0.3, 0.6]$ when $k > 11$ by clipping $u_{i,k}$.

The hyperparameters for training are summarized in Table III. The values of all the hyperparameters were selected by performing a grid search as in [23], using the values reported in [25] as a reference. DDPG has two hidden layers with 256 and 128 nodes, respectively; while FH-DDPG and FH-DDPG-SS have three hidden layers with 400, 300, and 100 nodes, respectively. The sizes of input layers for all DRL algorithms are the same and decided by the PF information topology. Moreover, an additional 1-D action input is fed to the second hidden layer for each critic network. The total number of training episodes $E$ for all DRL algorithms is set to 5000. For FH-DDPG-SS, we first train the algorithm for 3000 episodes to learn the kick-off policy in the first phase, and then continue to train 2000 episodes within the reduced state space in the second phase. The replay buffer sizes for DDPG and FH-DDPG are 250000 and 2500, respectively. This is because the replay buffer for FH-DDPG only stores the data entries for the corresponding time step. Since FH-DDPG-SS is trained in two phases, the replay buffer sizes for the first and second phases are 25000 and 2000, respectively. Moreover, FH-DDPG-SS leverages the FH-DDPG-SA-(NB) algorithm, which trains the $K-m-1$ actors and critics for time steps $K$ to $m+1$ using FH-DDPG, and a single pair of actor and critic for time steps 1 to $m$ using DDPG. The soft target update is implemented with a parameter of $\eta = 0.001$ for DDPG. As FH-DDPG uses a fixed target network, there is no soft target update.

B. Comparison of FH-DDPG-SS With the Benchmark Algorithms

1) Performance for Testing Data: The individual performance of each follower $i \in \{1, 2, 3, 4\}$ as well as the sum performance of the four followers are reported in Table IV for DDPG, FH-DDPG, HCFS, and FH-DDPG-SS, respectively. The individual performance of each follower and the sum performance of all followers are obtained by averaging the returns of the corresponding followers and the sum returns of all followers, respectively, over 200 test episodes after training is completed. Note that in RL terminology, a return is the cumulative rewards of one episode. It can be observed that the individual performance of the preceding vehicles is attenuated by following vehicles upstream of the platoon for each algorithm. Compared with the other algorithms, FH-DDPG-SS consistently shows the best individual performance for each follower $i \in \{1, 2, 3, 4\}$.

We can observe that the ranking in terms of the sum performance of all followers for the different algorithms is FH-DDPG-SS > HCFS > FH-DDPG > DDPG, where FH-DDPG-SS outperforms DDPG, FH-DDPG, and HCFS algorithms by 46.62%, 13.73%, and 10.60%, respectively. Note that HCFS outperforms DDPG because two actions are obtained from DDPG and the classical linear controller at each time step, and the one that achieves the maximum reward is selected. On the other hand, FH-DDPG performs better than DDPG as it applies backward induction and time-dependent actors/critics. Moreover, FH-DDPG-SS further improves the performance of FH-DDPG by implementing the three key ideas proposed in Section IV.

In Table V, we present the maximum, minimum, and standard deviation of the sum returns of the four followers across the 200 test episodes for DDPG, FH-DDPG, HCFS, and FH-DDPG-SS, respectively. It can be observed that although the objective of DRL algorithms is to maximize the expected return as given in (16) and (17), FH-DDPG-SS achieves the best performance in terms of the maximum, minimum, and standard deviation of the returns among all the DRL algorithms.
TABLE IV
Performance after training with the NGSIM data set. Each episode has 100 time steps in total. We present the individual performance of each follower as well as the sum performance of the four followers for DDPG, FH-DDPG, HCFS, and FH-DDPG-SS, respectively.

| Algorithm    | Individual performance | Sum performance |
|--------------|------------------------|-----------------|
|              | Follower 1 | Follower 2 | Follower 3 | Follower 4 |                  |
| DDPG         | -0.0680   | -0.0876   | -0.0899   | -0.2980   | -0.5435          |
| FH-DDPG      | -0.0736   | -0.0845   | -0.0856   | -0.0927   | -0.3364          |
| HCFS         | -0.0673   | -0.0740   | -0.0828   | -0.1005   | -0.3246          |
| FH-DDPG-SS   | -0.0600   | -0.0691   | -0.0776   | -0.0835   | -0.2902          |

(2) Convergence Properties: The performance of DRL algorithms is evaluated periodically during training by testing without exploration noise. Specifically, we run ten test episodes after every 100 training episodes, and average the returns over the ten test episodes as the performance for the latest 100 training episodes. The performance as a function of the number of training episodes for each follower with DDPG, FH-DDPG, and FH-DDPG-SS is plotted in Fig. 2. The convergence curve of HCFS is not plotted here since HCFS combines the trained DDPG controller with the linear controller, and thus the convergence property of HCFS is the same as that of DDPG. It can be observed from Fig. 2 that the initial performances of FH-DDPG and FH-DDPG-SS are much better than that of DDPG. This is due to the different ways how DDPG and FH-DDPG/FH-DDPG-SS work. For DDPG, the first point in Fig. 2(a) is obtained by running ten test episodes of the initial policy and calculating the average return. Since the initial weights of the actor network are random, the initial policy has poor performance. In FH-DDPG, there is a pair of actor/critic networks associated with each time step, and the actors/critics are trained backward in time. Therefore, when we train the actor/critic of the first time step, the actors/critics of the rest of the time steps are already trained. Similarly, the first point in Fig. 2(c) for FH-DDPG-SS is obtained by running ten test episodes where only the policies (actors) for time steps 1 to \( m \) are not trained, while the policies (actors) of the time steps \( m + 1 \) to \( K \) are already trained.

Fig. 2(a) shows that although the performances of all the followers in DDPG improve quickly in the first 200 episodes, the performance curves exhibit significantly larger oscillation during the following training episodes compared to those of FH-DDPG and FH-DDPG-SS. This is especially true for follower 4, whose performance drops below \(-20\) at around 1500 episodes. The above results demonstrate that the convergence of DDPG is relatively unstable, especially for the followers upstream the platoon. As shown in Fig. 2(b), the convergence stability of FH-DDPG is significantly improved over DDPG, as there are only small fluctuations in the performance curve. This is due to the fact that FH-DDPG trains backward in time, where DDPG with a fixed target is adopted in each time step to solve a one-period MDP. However, as FH-DDPG is under the framework of DP, it sacrifices the sampling and training efficiency to achieve the convergence stability. As shown in Fig. 2(b), although the performance of FH-DDPG quickly increases in the first 200 episodes, there is hardly any further improvement during the remaining training episodes. Fig. 2(c) shows that the convergence stability of FH-DDPG-SS is similar to that of FH-DDPG and much better than that of DDPG. Moreover, the performance of FH-DDPG-SS is consistently better than that of FH-DDPG for each follower during the whole training episode. For example, focusing on follower 1, the performance of FH-DDPG is around \(-0.0735\) at the beginning of training and finally reached \(-0.0715\) toward the end.
On the other hand, the performance of FH-DDPG-SS is around $-0.0665$ initially and finally reached $-0.0600$ at the end of training. The superior performance of FH-DDPG-SS before 3000 episodes is due to the two proposed key ideas of transferring network weights backward in time and stationary policy approximation for earlier time steps, which improve the sampling efficiency. In other words, from the same number of data entries generated by the 3000 training episodes, FH-DDPG-SS can make more efficient use of the data entries to learn a better policy. Moreover, it can be observed that there is a sudden performance improvement for all the vehicles of FH-DDPG-SS at around 3000 episodes. This salient performance gain is due to the third key idea of sweeping through the reduced state space. Since FH-DDPG-SS is trained in the refined state space for the last 2000 episodes, the training efficiency is improved as agents focus on learning the states that good policies frequently visit.

3) Testing Results of One Episode: We focus our attention on a specific test episode having 100 time steps, and plot-driving status $\epsilon_{\text{evi},k}$, $\delta_{\text{evi},k}$, $\alpha_{\text{evi},k}$ and control input $u_{\text{evi},k}$ along with jerk $j_{\text{evi},k}$ of each follower $k$ for all the time steps $k \in \{1, 2, \ldots, 100\}$. Fig. 3(a)–(d) shows the results of a specific test episode for DDPG, FH-DDPG, HCFS, and FH-DDPG-SS, respectively. It can be observed that the overall shapes of the corresponding curves of all the algorithms look very similar except that the performance curves for follower 4 using DDPG have large oscillations. This observation is aligned with the results in Table III, where follower 4 has significantly worse performance when using DDPG compared with using other algorithms. Fig. 3 shows that in general for each follower $i \in \{1, 2, 3, 4\}$, $\epsilon_{\text{pi},i}$ has an initial value of 1.5 m and is reduced over time to approximately 0 m; $\epsilon_{\text{vi},i}$ has an initial value of $-1$ m/s and is increased to approximately 0 m/s. $\alpha_{\text{evi},i}$ is relatively large at the beginning of the episode to increase acc_{evi,ik} as fast as possible, so that $\epsilon_{\text{pi},i}$ and $\epsilon_{\text{vi},i}$ can promptly converge to 0. Correspondingly, acc_{evi,ik} of each follower $i$ has an initial value of 0 m/s$^2$ and is suddenly increased to a relatively large value. Then both $\alpha_{\text{evi},i}$ and acc_{evi,ik} are quickly reduced to a negative value, and finally are increased over time to approximately 0 m/s$^2$. After the driving status and control input converge to near 0, the values fluctuate around 0 with $u_{\text{evi},i}$ trying to maximize the expected cumulative reward in (17) without knowing the future control inputs $u_{\text{evi},i-1}$, $k < K < K$, of the predecessor $i-1$. Additionally, $j_{\text{evi},i}$ of each follower $i$ starts with a large positive value and is then reduced to a negative value. After converging to near 0 m/s$^3$, the value of $j_{\text{evi},i}$ fluctuates around 0 m/s$^3$.

A closer examination of Fig. 3 reveals that the performance differences of the algorithms are reflected in convergence speed to steady-state and the oscillations of the driving status and control input. Focusing on $\epsilon_{\text{pi},i}$, it can be observed that there are still positive gap-keeping errors for followers 2–4 in DDPG up to the end of the episode. Moreover, $\epsilon_{\text{pi},i}$ of follower 4 in DDPG has the slowest convergence speed to 0 m and the largest oscillations among all the algorithms. Meanwhile, $\epsilon_{\text{pi},i}$ in FH-DDPG is reduced to 0 m for each follower, but there are relatively large oscillations after convergence. $\epsilon_{\text{pi},i}$ in HCFS has smaller oscillations than that in FH-DDPG and also converges to 0 m, but there are also large oscillations near the end of the episode for follower 4. $\epsilon_{\text{pi},k}$ in FH-DDPG-SS has the fastest convergence speed to 0 m, and then remains around 0 m with small oscillations.

C. Platoon Safety

In order to demonstrate that the platoon safety is ensured in the proposed FH-DDPG-SS algorithm, Table VI summarizes the average, maximum, and minimum return as well as the standard deviation across the 200 test episodes for each follower $i$ of FH-DDPG-SS. It can be observed from Table VI that the standard deviation of each follower $i$ is small ranging from 0.0015 to 0.0017. Moreover, the differences between the maximum and minimum returns are small for all followers. Specifically, the minimum return is worse than the maximum return by 16.28%, 15.22%, 13.95%, and 12.96% for the four followers, respectively.

To demonstrate that platoon safety is ensured even in the worst test episode, it can be observed in Fig. 4 that the followers have an initial gap-keeping error $\epsilon_{\text{pi},0}$ of 1.5 m and the gap-keeping error is reduced over time to approximately 0 m. The most negative $\epsilon_{\text{pi},k}$ is $-0.1014$ m at $k = 35$, which will not result in vehicle collision since the absolute value of the position error is much smaller than the desired headway.

D. String Stability

The string stability of a platoon indicates whether oscillations are amplified upstream of the traffic flow. The platoon is called string stable if sudden changes in the velocity of a preceding vehicle are attenuated by following vehicles upstream of the platoon [51]. To show the string stability of
Fig. 3. Results of a specific test episode. The driving status $e_{pi,k}$, $e_{vi,k}$, and $acc_{i,k}$ along with the control input $u_{i,k}$ and jerk $j_{i,k}$ of each follower $i$ are represented as different curves, respectively. (a) DDPG. (b) FH-DDPG. (c) HCFS. (d) FH-DDPG-SS.

As shown in Fig. 5, the amplitude of the oscillations in $e_{pi,k}$ and $e_{vi,k}$ of each follower $i \in \{2, 3, 4\}$ are both smaller than those of its respective predecessor $i - 1$, demonstrating the string stability of the platoon.

the proposed FH-DDPG-SS algorithm, we simulate the platoon where the leader acceleration is set to $2 \text{ m/s}^2$ when $20 < k \leq 30$, and $0 \text{ m/s}^2$ otherwise. The followers’ initial gap-keeping and velocity errors are all set to 0.
VI. Conclusion

This article has studied how to solve the platoon control problem using an integrated DRL and DP method. First, the SSDP model for platoon control has been formulated with a Huber loss function for the reward. Then, the FH-DDPG-SS algorithm has been proposed to improve sampling and training efficiency over the baseline FH-DDPG algorithm with three key ideas. Finally, the performance of FH-DDPG-SS has been compared with DDPG, FH-DDPG, and HCFS based on real driving data extracted from the NGSIM. The results have shown that FH-DDPG-SS has learned better policies for platoon control, with significantly improved performance and better convergence stability. Moreover, the platoon safety and string stability for FH-DDPG-SS have been demonstrated.

Appendix

A. Pseudocode of FH-DDPG Algorithm

The pseudocode of the FH-DDPG algorithm [41] is given in Function 1.

References

[1] R. Silva and R. Iqbal, “Ethical implications of social Internet of Vehicles systems,” IEEE Internet Things J., vol. 6, no. 1, pp. 517–531, Feb. 2019.
[2] Y. Chen, C. Lu, and W. Chu, “A cooperative driving strategy based on velocity prediction for connected vehicles with robust path-following control,” IEEE Internet Things J., vol. 7, no. 5, pp. 3822–3832, May 2020.
[50] M. Abadi et al., “Tensorflow: A system for large-scale machine learning,” in Proc. 12th USENIX Symp. Oper. Syst. Design Implement., 2016, pp. 265–283.

[51] G. J. L. Naus, R. P. A. Vugts, J. Ploeg, M. J. G. van de Molengraft, and M. Steinbuch, “String-stable CACC design and experimental validation: A frequency-domain approach,” IEEE Trans. Veh. Technol., vol. 59, no. 9, pp. 4268–4279, Nov. 2010.

Tong Liu received the B.S. degree from Nanjing University of Posts and Telecommunications, Nanjing, China, in 2018. He is currently pursuing the Ph.D. degree with Beijing University of Posts and Telecommunications, Beijing, China.

His current research interests include deep reinforcement learning, modern control theory, and their application in Internet of Vehicles.

Lei Lei (Senior Member, IEEE) received the B.S. and Ph.D. degrees in telecommunications engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2001 and 2006, respectively.

She is currently an Associate Professor with the College of Engineering and Physical Sciences, University of Guelph, Guelph, ON, Canada. Her research interests mainly lie in machine learning/deep reinforcement learning, Internet of Things/Internet of Vehicles, mobile-edge computing, and smart grid.

Kuan Zhang (Member, IEEE) received the B.Sc. degree in communication engineering and the M.Sc. degree in computer applied technology from Northeastern University, Shenyang, China, in 2009 and 2011, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2016.

He has been an Assistant Professor with the Department of Electrical and Computer Engineering, University of Nebraska–Lincoln, Omaha, NE, USA, since 2017. He was also a Postdoctoral Fellow with the Broadband Communications Research Group, Department of Electrical and Computer Engineering, University of Waterloo from 2016 to 2017. His current research interests include security and privacy for mobile social networks, e-healthcare systems, cloud/edge computing, and cyber–physical systems.

Dr. Zhang was the recipient of the Best Paper Award at IEEE WCNC 2013, Securecomm 2016, and BigDataSE 2019.

Kan Zheng (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees from Beijing University of Posts and Telecommunications, Beijing, China, in 1996, 2000, and 2005, respectively.

He is currently a Full Professor with Ningbo University, Ningbo, Zhejiang, China. He has rich experiences in research and standardization of new emerging technologies. He has authored over 200 journal articles and conference papers in the field of wireless communications, vehicular networks, IoT, and security.

Prof. Zheng holds editorial board positions with several journals. He has also served in the organizing/TPC committees for more than ten conferences.