Transportation Research Part C 147 (2023) 103987

Contents lists available at ScienceDirect

Transportation Research Part C

journal homepage: www.elsevier.com/locate/trc

# Modified DDPG car-following model with a real-world human driving experience with CARLA simulator

Dianzhao Li *, Ostap Okhrin

*Chair of Econometrics and Statistics, esp. in the Transport Sector, Technische Universität Dresden, Dresden, Germany*

ARTICLE INFO

*Keywords:*
DRL
Car-following model
Real driving dataset
CARLA
ROS

ABSTRACT

In the autonomous driving field, fusion of human knowledge into Deep Reinforcement Learning (DRL) is often based on the human demonstration recorded in a simulated environment. This limits the generalization and the feasibility of application in real-world traffic. We propose a two-stage DRL method to train a car-following agent, that modifies the policy by leveraging the real-world human driving experience and achieves performance superior to the pure DRL agent. Training a DRL agent is done within CARLA framework with Robot Operating System (ROS). For evaluation, we designed different driving scenarios to compare the proposed two-stage DRL car-following agent with other agents. After extracting the "good" behavior from the human driver, the agent becomes more efficient and reasonable, which makes this autonomous agent more suitable to Human–Robot Interaction (HRI) traffic.

## 1. Introduction

With the vigorous development of new energy vehicles, autonomous driving has become more and more attractive to the academia. Due to the complex real-world traffic environment, vehicles are making continuously decisions that are building cornerstones in achieving fully autonomous driving. A list of solutions for complex decision-making problems has been proposed in the literature over the last decades.

The first family of autonomous techniques is the so-called *rule-based* control strategies, which use lists of complex control rules to define the behavior of vehicles in the traffic flow. For the car-following task in autonomous driving, models worth mentioning are the Gaxis–Herman–Rothery (GHR) model by Gazis et al. (1961), the Wiedemann (1974) car-following model, Intelligent Driver Model (IDM) by Treiber et al. (2000) or stochastic car-following models by Treiber and Kesting (2017). Kikuchi and Chakroborty (1992) proposed a fuzzy inference system-based car-following model, which consists of many direct natural language-based driving rules. Obviously, it is impossible to consider all the situations that may occur and formulate corresponding control strategies accordingly. Moreover, rule-based control strategies are not suitable for time-varying and non-stationary traffic conditions.

Due to the rapid development of deep learning in recent years, second approach to solve control problems involves the use of *imitation learning* (IL) or *behavior cloning* (BC). With BC, the perception and control parts in autonomous driving are learned from human demonstrations using deep neural networks (DNN). The DNN learns, from an RGB image from the camera or corresponding features obtained from other sensors like LiDAR, Laser, Ultrasonic, and GPS as input and outputs the desired reaction to the vehicle control in certain circumstances. End-to-end imitation systems can be learned offline in a safe way. Bojarski et al. (2016) trained a convolutional neural network (CNN) to map raw images from a single camera directly to steering commands. Codevilla et al. (2018) proposed command-conditional IL, in which low-level controls, as well as high-level commands, are learned from expert demonstrations with vision-based inputs.

* Corresponding author.
*E-mail addresses:* dianzhao.li@tu-dresden.de (D. Li), ostap.okhrin@tu-dresden.de (O. Okhrin).

https://doi.org/10.1016/j.trc.2022.103987
Received 1 April 2022; Received in revised form 8 September 2022; Accepted 8 December 2022
Available online 21 December 2022

Although BC is widely used, even in real cars, that are allowed for driving on the streets, it still has many shortcomings. Codevilla et al. (2019) pointed out four main limitations:

*Generalization*: As a type of supervised learning, BC is also limited by the size of the dataset and for scenarios, not appearing in the dataset, the performance will be unsatisfactory.
*Driving dataset biases*: Supervised learning is limited by the type or the source of the dataset, and the trained control strategy will be biased. Diversity is an important criterion for a dataset, as every driver has his own driving style, and even different vehicles have different operation performances, which are reflected in the datasets.
*Causal confusion*: Some spurious correlations cannot be distinguished from true causes in human demonstration patterns if we do not use an explicit causal model or on-policy demonstrations, see de Haan et al. (2019).
*High variance*: Since off-policy IL uses a static dataset, initialization and data sampling will induce high variance. As the cost function in BC is optimized via Stochastic Gradient Descent, which assumes the data is independent and identically distributed. However, the human demonstration dataset is usually interrelated over a long period of time. Therefore the model can be very sensitive to the initialization, see Hanin and Rolnick (2018).

A feasible solution to the challenge above that has attracted widespread attention in the academic community over the last decades is *Deep Reinforcement Learning* (DRL). DRL that combines reinforcement learning and deep learning, has provided solutions for many complex decision-making tasks on playing chess, Go, Atari games, uses in robotics, drones, and vehicle behavior (Mnih et al., 2013; Silver et al., 2016, 2017; Gu et al., 2017; Akhloufi et al., 2019; He et al., 2017; Isele et al., 2018). Within this field, agents interact with the environments to learn the optimal behaviors, improving over time through trial and error, and in general, do not base on the datasets.

Although DRL methods are present, unfortunately, are not prominent in transportation scenarios. For example, Sallab et al. (2016) used Deep Q learning (DQN) and Deep Deterministic Actor Critic Algorithm (DDAC) as control algorithms for lane-keeping tasks in TORCS simulator. Later, Wang et al. (2018) proposed Q-learning based training process to train a control policy for lane-changing tasks with a smooth and efficient behavior. The study by Ngai and Yung (2011) offered a multiple-objective RL framework, which is used to solve the overtaking problem in traffic flow with correct action as well as collision avoidance. Recently, Nosrati et al. (2018) proposed an RL-based hierarchical framework with DQN and Proximal Policy Optimization (PPO) to solve the multi-lane cruising problem, showing that this design enables better learning performance. Hart et al. (2021) proposed a DRL car-following model which decomposes the multi-goal problem into two subtasks: free-driving and car-following with a Modularized Reinforcement Learning approach, therefore the car-following agent can follow the leader with the desired speed and also keep a reasonable gap to the leader.

In addition to pure DRL, researchers are combining DRL with human prior knowledge to improve exploration efficiency. Hester et al. (2018) proposed the Deep Q-learning from Demonstration (DQfD) algorithm, which maintained two separate replay buffers to store demonstration data and self-generated data respectively. They showed that DQfD performs better than other related algorithms for incorporating demonstration data into DQN. Vecerik et al. (2017) utilized human demonstrations with Deep Deterministic Policy Gradient (DDPGfD) algorithm. The human demonstrations and actual interactions are filled into the replay buffer and sampled with prioritized replay mechanism. The results on simulated tasks showed that DDPG with demonstrations outperforms pure DDPG. Huang et al. (2021) proposed a framework that first uses BC with the expert demonstration to derive an imitative expert policy and then further improve it with DRL. A different approach has been done by Liu et al. (2021) who modified the update of the policy network in RL to leverage human prior knowledge, which can adaptively sample experience from the agent's self-exploration and expert demonstration for the policy update.

All aforementioned methods combining human demonstrations with DRL has four limitations: First, application of supervised learning to human demonstrations and obtaining a decent initialization for the DRL policy network does not guarantee a more comprehensive utilization of the demonstrations, a pure DRL agent is often unable to safely interact with human drivers in real-world traffic (Litman, 2017). Second, training a human-like agent with DRL or BC usually requires fairly large datasets, otherwise, the performance is poor. The real-world datasets are not enough to cover all the possible traffic scenarios. Moreover, in the real-world driving datasets, for instance, NGSIM and HighD datasets, the drivers are influenced by the complex real traffic environment and cannot focus on only one certain driving task. Since we want our agent to focus on the car-following behavior only, the commonly used datasets need to be re-extracted. Therefore the size of the datasets will be reduced greatly. In addition, recording data in the simulation usually requires dozens of hours of operations by human experts, which is very resource-intensive. Third, most of the papers combine human demonstrations with DRL, which are generated and recorded in a simulated environment. This leads to restrictions on generalization and dataset biases issues when using the data (Huang et al., 2021; Liu et al., 2021). Fourth, while training a DRL agent with human demonstrations, an important issue for this off-policy learning is *extrapolation error* (Fujimoto et al., 2019), which occurs when a mismatch happens between the distribution of data induced by the current policy and the distribution of data contained in the batch. Consequently, it will be impossible to learn a value function for a policy that selects actions not contained in the batch. In the next, we briefly outline our contribution.

### 1.1. Contribution

All these limitations above motivated us to propose the two-stage DRL algorithm that tackles all of these issues. The two-stage DRL algorithm trains car-following agents that balance the policies learned by the agent itself through DRL in the simulator and the

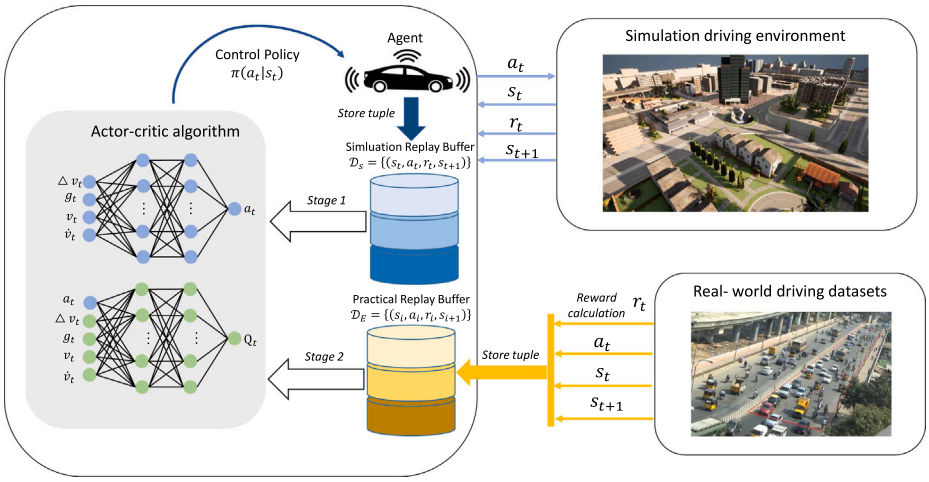



**Fig. 1.** The conceptual framework of the two-stage DDPG agent. Stage 1: the agent updates networks with the simulation replay buffer. Stage 2: the agent also learns from the modified practical replay buffer based on real driving datasets.

policies implied in real-world human demonstration datasets. Therefore we obtain an agent that is superior to pure DRL agents and drivers in the datasets and more suitable for real-world traffic. The overall system architecture is depicted in Fig. 1. First, we use DRL with experience replay (Lin, 1992) to train an agent that can follow the leading vehicle and maintain the desired distance and speed. Second, we store the real driving datasets into the practical replay buffer and resume training the agent with both replay buffers. We itemized our contributions as follows:

1. We mix the real-world database with the self-generated data in the simulation, to encourage the DRL agent to learn expert-like behaviors by leveraging the mixed data (Hester et al., 2018; Vecerik et al., 2017). We prove that our two-stage DRL algorithm can obtain the expert-like behavior policy which is superior to pure DRL agents and real divers, and this will also tackle the performance drop due to dataset with limited size in common DRL or BC algorithms.
2. Since DRL agents need to continuously interact with the environment to obtain rewards and optimize their policy, which means training a DRL agent in a real-world traffic environment is often impractical and expensive. Therefore, we propose a novel framework that uses CARLA (Car Learning to Act), an open-source simulator for autonomous driving research, and Robot operating system (ROS) for co-simulation to train our DRL agent in a real-world like environment with real dynamics and can be applied seamlessly to the real world with greatly improved efficiency (Dosovitskiy et al., 2017; Quigley et al., 2009).
3. We compare different utilization of real-world driving datasets for DRL agents and tackle the *extrapolation error* when learning from real data.

To sum up, within this paper we propose the first car-following model that is trained via real datasets and DRL on the simulating environment CARLA and ROS with hyper-realistic dynamics. The remainder of this paper is organized as follows. In Section 2, the preliminary knowledge related to this work is reviewed. Then, we introduce our approach in detail in Section 3 and the experimental setups for the training and evaluation process are discussed in Section 4. Afterwards, we evaluate the trained agents with different driving scenarios and discuss the results in Section 5. Section 6 summarizes this paper.

## 2. Related works

### 2.1. Reinforcement learning

In RL, an agent interacts with an environment under the objective to maximize the received reward. Consider it as a Markov Decision Process (MDP), an autonomous agent at time step $t$ observes a state $s_t$ of the environment and then interacts with the environment by executing an action $a_t$ according to the policy $\pi(a_t|s_t)$. Afterwards, the environment and the agent transition to a new state $s_{t+1}$ with the probability $P(s_{t+1}|s_t, a_t)$ and meanwhile a reward $r_{t+1}$ is provided to the agent as the feedback. The objective is to find an optimal policy $\pi^*$ that maximizes the expected discounted cumulative rewards $\mathbb{E}_{\pi^*}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}\right\}$, where $\gamma \in [0, 1]$ is the discount factor with lower values making the agent to prefer immediate to distant rewards. Furthermore, in RL we use a state-value function $V^{\pi}(s_t) = \mathbb{E}_{\pi}\left\{R_t|s_t\right\}$ to measure how good a certain state $s_t$ is, in terms of expected cumulative reward by following a certain policy $\pi$. Similarly, the action-value function $Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi}\left\{R_t|s_t, a_t\right\}$ is defined as the expected return of the agent, that starts from state $s_t$, executes an action $a_t$, and follows policy $\pi$ afterwards. Although RL can come up with some optimal policies

after training, for the large state and action space in high dimensions, such a tabular format becomes computationally infeasible. With the establishment of deep learning, DRL regained interest with DNN being used to approximate the optimal Q values (Mnih et al., 2015; Silver et al., 2016).

### 2.2. Deep deterministic policy gradient (DDPG)

To tackle the high-dimensional, continuous action spaces, Lillicrap et al. (2015) developed a Deep deterministic policy gradient (DDPG) method that can learn policies in high-dimensional, continuous action spaces. DDPG is an actor-critic algorithm with two networks: actor network $\mu(s|\theta^{\mu})$ with network parameter $\theta^{\mu}$ and critic network $Q(s,a|\theta^{Q})$ with network parameter $\theta^{Q}$. The actor network will output an action $a_t$ based on the given states $s_t$, the critic network gets the action $a_t$ from the actor network, and the given states $s_t$, predict the goodness of the action $a_t$. DDPG uses the target networks method with the target actor network $\mu'$ and target critic network $Q'$. Lillicrap et al. (2015) found that for more stable learning, it is better to make the target networks slowly track the trained networks. The parameters of the target networks after each update of the trained network are updated using a sliding average for both the actor and the critic:

$$\theta^{\mu'} = \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}, \tag{1}$$

$$\theta^{Q'} = \tau\theta^{Q} + (1-\tau)\theta^{Q'}, \tag{2}$$

where $\tau \ll 1$ is the soft target update rate.

To balance between exploration and exploitation, an additive noise is usually added to the deterministic action to explore the action space:

$$a_t = \mu(s_t|\theta^{\mu}) + \xi_t. \tag{3}$$

This additive noise $\xi_t$ could be the realization of the Gaussian distribution or the zero-reverting Ornstein and Uhlenbeck (1930) process suggested by Lillicrap et al. (2015). The pseudo-code of DDPG is shown in Algorithm 1. In this work, as the action and state space for the car-following task are continuous, we use DDPG as our DRL algorithm. Since the main contribution is the proposal of the fusion of RL and small real datasets with simulation environment CARLA, we do not make any comparison with other DRL algorithms.

**Algorithm 1:** DDPG algorithm

Initialize actor network $\mu$ and critic network $Q$ with random weights $\theta^{\mu}$ and $\theta^{Q}$
Initialize the target networks $\mu'$ and $Q'$ with weights $\theta^{\mu'} \leftarrow \theta^{\mu}$, $\theta^{Q'} \leftarrow \theta^{Q}$
Initialize replay buffer $D$
**for** $episode \in [1, M]$ **do**
  Initialize a random process noise $\xi$ for action exploration
  Receive initial observation state $s_1$
  **for** $t \in [1, T]$ **do**
    Select action $a_t = \mu(s_t|\theta^{\mu}) + \xi_t$ according to the current policy and exploration noise
    Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $D$
    Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $D$
    Set $y_i = r_i + \gamma Q'\left\{s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}\right\}$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i \left\{y_i - Q(s_i, a_i|\theta^{Q})\right\}^2$
    Update the actor policy using the sampled policy gradient: $\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^{Q})|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s_i}$
    Update the target networks:
    $\theta^{\mu'} = \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$
    $\theta^{Q'} = \tau\theta^{Q} + (1-\tau)\theta^{Q'}$
  **end**
**end**

### 2.3. Experience replay

If the networks only learned the highly correlated consecutive samples from the experience as they occurred sequentially in the environment, this would therefore lead to inefficient learning. One possible approach to tackle this problem involves the use of *experience replay* proposed by Lin (1992). Following it we store the experiences of an agent at each time step in a dataset called the *replay memory* $D_t$. Let the experience of an agent at time $t$ can be denoted as:

$$d_t = (s_t, a_t, r_t, s_{t+1}), \tag{4}$$

containing the state of the environment $s_t$, the action $a_t$ from agent at given state, the reward $r_t$ the agent received as a result of the previous state–action pair $(s_t, a_t)$, and the next state of the environment $s_{t+1}$. With this replay buffer, we randomly sample some tuples that break the temporal correlations by mixing less recent experiences for the updates, and rare experiences will be used for more than just a single update. To utilize real driving datasets in DRL, *experience replay* mechanism is essential, since we need to store the dataset in the *replay memory* and let the agent learn the driving policy implicit in the datasets.

### 2.4. Extrapolation error

Extrapolation error is the distributional shift between the dataset and true state–action visitation of the current policy in off-policy value learning (Fujimoto et al., 2019). Since in RL, the accuracy of Q-function depends on the estimate of the Q-value, when the current policy $\pi(a|s)$ differs substantially from the policy $\pi_\beta(a|s)$ in the replay buffer, which means the target policy selects an out-of-distribution (OOD) action $a'$ at the state $s'$, such that $(s', a')$ is unlikely, or not contained in the replay buffer. The estimate of Q-value may be arbitrarily bad without sufficient data near the current $(s', a')$ and these estimation errors will accumulate over each iteration, resulting in arbitrarily poor performance (Levine et al., 2020).

As stated by Fujimoto et al. (2019), the off-policy RL algorithms are ineffective when learning *truly off-policy* without any online interaction, not only the distributional shift of trained policy but also the distributional shift of different initialization will affect the performance of off-policy learning. One way to overcome this problem is to add active data collection and compensate the estimated error (Vecerik et al., 2017; Hester et al., 2018; Dadashi et al., 2021). For this reason, we combine the real-world driving dataset and the online interactive experience in the replay buffer to tackle *extrapolation error* and improve the performance of our DRL agent.

### 2.5. CARLA and ROS

Since training a DRL agent in the real world is impossible and expensive, we usually use a simulated environment instead. There are numerous well-known autonomous driving simulators used in academic research such as CARLA, CarSim, LGSVL, Gazebo, etc. In this work, we use CARLA and ROS as our training platforms. CARLA is an open-source simulator grounded on Unreal Engine with hyper-realistic physics that uses the OpenDRIVE standard to define roads and urban settings (Dosovitskiy et al., 2017; Dupuis et al., 2010). It is based on a scalable client–server architecture, with the server being responsible for simulation tasks, e.g. sensor rendering, computation of physics, etc. The client is usually defined by the user, which via API can easily specify the sensor setup of the vehicle. Afterwards, a controlled amount of camera images and vehicle-related information will be provided for further usage. Because of the high fidelity of CARLA, it is often used in the field of autonomous driving (Ravi Kiran et al., 2018; Dworak et al., 2019; Gómez-Huélamo et al., 2020; Tran and Le, 2019; Niranjan et al., 2021). Furthermore, an important reason why we choose CARLA as our training environment is that it provides the CARLA-ROS bridge that enables two-way communication between Robot Operating System (ROS) (Quigley et al., 2009) and CARLA.

Connection to ROS is vital for our work since ROS is an open-source robotics middleware suite, which enables the communication between various sensors on a single robot and multiple robots without affecting the independence of each part. As our vehicles are equipped with several sensors to perceive the surroundings, exposing nearby vehicles, communication middleware is required between vehicles as well as between sensors and controllers within one vehicle. Since automated vehicles are complex systems with a high degree of interdependencies between their components, ROS is suitable for us to carry out autonomous driving tasks such as car-following and makes our agent vehicle access the information of the leading vehicle such as position and velocity in a leader–follower pair (Hellmund et al., 2016). Furthermore, ROS is a robot system that is well developed and used for autonomous robots and vehicles. The use of ROS in the simulation would help us to transfer the trained car-following agent to the real vehicle seamlessly in our future work.

## 3. Our approaches

### 3.1. System overview

As shown in Fig. 1, we train and evaluate our control policies directly using the CARLA simulator. The agent interacts with the simulation environment to obtain rewards, observe the states, and store them in the replay buffer. Therefore we update the DNN with the small batches sampled from the replay buffer. Here, we assume that the states observed by the agent are already known, including the position of the leader and follower, velocity, acceleration, and other information. Since this only involves simple sensor information processing issues which are out of the scope of the current research. After training the DRL agent, we store the processed real driving dataset into another replay buffer called the *practical replay buffer* and continue training the agent with both replay buffers.

### 3.2. DDPG agent for longitudinal control

In this paper, we use DDPG as our working DRL algorithm to obtain the longitudinal control policy, with the setup introduced in this section.

### 3.2.1. Action and state space

In the case of the end-to-end DRL algorithm, the action space is the throttle and brake. Although this can potentially solve the internal control problem of the vehicle, it is accompanied by a decrease in generalization. Different vehicles have different dynamic characteristics, therefore keeping our algorithm useful for fixed vehicles and cannot be applied to other types. For this reason, the action space $A$ of our DDPG agent uses continuous variable acceleration $\dot{v}_t$. Moreover, the real human driving dataset of the platoon driving experiments of Punzo et al. (2005) does not contain direct information about throttles and brakes, but the speed and acceleration. If we want to use real driving data, it is also more conducive to using acceleration as action space.

To follow the leader, the agent needs to be able to communicate with it, so as the state space $S$, we use such features as the velocity of follower $v_t$, the acceleration of follower $\dot{v}_t$, the leader's velocity $v_{t,l}$, and the bumper-to-bumper gap $g_t$. As suggested by Kim (1999), we normalize the data generally for speeding up the learning process and faster convergence. The state $s_t$ at time step $t$ is thus defined as

$$s_t = \begin{pmatrix} \frac{v_t}{v_{des}} \\ \frac{\dot{v}_t - \dot{v}_{min}}{\dot{v}_{max} - \dot{v}_{min}} \\ \frac{v_{t,l} - v_t}{v_{des}} \\ \frac{g_t}{g_{max}} \end{pmatrix}, \tag{5}$$

where $v_{des}$ denotes the desired speed, $\dot{v}_{max}$ and $\dot{v}_{min}$ define the feasible range of accelerations, $g_{max}$ indicates the maximum space gap for normalization of the input states.

### 3.2.2. Reward function

As the most critical part of the RL algorithm, the formulation of the reward function is highly related to the performance of the agent. We choose the reward function proposed by Hart et al. (2021), with which, the trained agent performs greatly in the car-following task. The reward function focuses on safety factor and also makes a balance between the comfort for the driver and the driving efficiency. It can be divided into three weighted sub-rewards:

$$r_t = w_{safe} r_{t,\,safe} + w_{gap} r_{t,\,gap} + w_{jerk} r_{t,\,jerk}. \tag{6}$$

The first term $r_{t,\,safe}$ compares the kinematically needed deceleration with the comfortable deceleration $b_{comf}$ and focuses on the response of driver to safety–critical situations:

$$r_{t,\,safe} = -\tanh\left(\frac{b_{kin} - b_{comf}}{-\dot{v}_{min}}\right) \mathbb{I}\{b_{kin} > b_{comf}\}, \tag{7}$$

where $b_{kin} = \frac{v_t - v_{t,l}}{g_t} \mathbb{I}\{v_t > v_{t,l}\}$, and $\mathbb{I}\{\cdot\}$ is the indicator function.

The second term $r_{t,\,gap}$ devises for driving efficiency, to prevent the follower agent from keeping a long distance with the leader to avoid the occurrence of dangerous situations:

$$r_{t,\,gap} = \begin{cases} \frac{\varphi\{(g_t - g_{opt})/g_{var}\}}{\varphi(0)}, & \text{if } g_t < g^*, \\ \frac{\varphi\{(g_t - g_{opt})/g_{var}\}}{\varphi(0)} \left(1 - \frac{g_t - g^*}{g_{lim} - g^*}\right), & \text{otherwise}, \end{cases} \tag{8}$$

where $g_{opt} = v_t T + g_{min}$, $g_{var} = 0.5 g_{opt}$, $g_{lim} = v_t T_{lim} + 2 g_{min}$, and $\varphi\{x\}$ describing the density function of the standard normal distribution.

The third term $r_{t,\,jerk}$ addresses the control of jerk for a comfortable driving:

$$r_{t,\,jerk} = -\left(\frac{1}{j_{comf}} \frac{\mathrm{d}\dot{v}_t}{\mathrm{d}t}\right)^2. \tag{9}$$

As the weights measure the trade-off among different sub-rewards, we conducted a series of experiments, tested different combinations of weights, and selected the set of weights that has the best performance and at most interpretable. We choose the weights for safety factor $w_{safe} = 1.0$, efficiency factor $w_{gap} = 0.5$ and the comfortable factor weights $w_{jerk} = 0.004$ with the rest of hyperparameters in Table 1. As seen from the weights, safety, being the most important, has the highest weight, while jerk, being just a comfort parameter the lowest.

### 3.2.3. Neural network structure

The structure of DDPG neural networks used in this study is depicted graphically in Fig. 2. Both actor and critic are using fully connected neural networks with two hidden layers, both hidden layers containing 32 neurons. In two hidden layers, we use ReLU activation function. For the output layer in critic network, we use linear activation function, and for the output layer in actor network, we use Tanh as activation function to scale the output to [−1, 1] for better performance, afterwards, this bounded output will be mapped to the acceleration range.

**Table 1**
Hyperparameters used for the input states and reward function setup.

| Symbol | Description | Value |
|---|---|---|
| $\dot{v}_{min}$ | Minimum acceleration | $-9\,\mathrm{m/s^2}$ |
| $\dot{v}_{max}$ | Maximum acceleration | $5\,\mathrm{m/s^2}$ |
| $v_{des}$ | Desired velocity | $20\,\mathrm{m/s}$ |
| $g_{max}$ | Maximum distance between leader and follower | 200 m |
| $b_{comf}$ | Comfortable deceleration | $2\,\mathrm{m/s^2}$ |
| $T$ | Desired time gap to the leading vehicle | 1.5 s |
| $g_{min}$ | Desired minimum space gap | 2 m |
| $T_{lim}$ | Upper time gap limit for zero reward | 15 s |

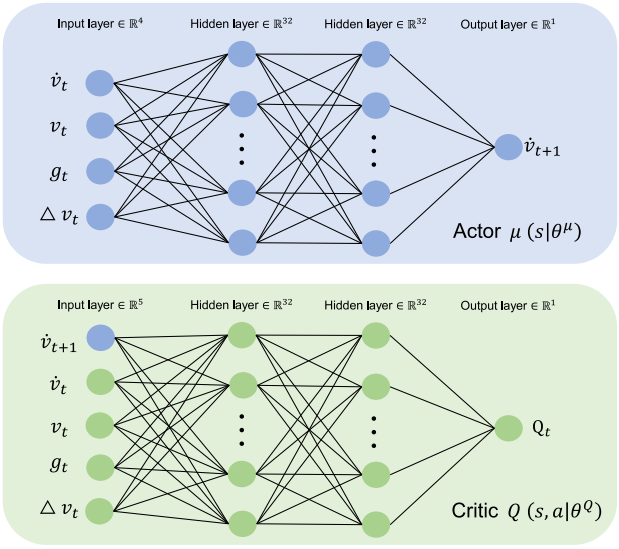



**Fig. 2.** Neural network architecture of Actor and Critic in DDPG, each of them has two hidden layers with 32 neurons in each layer.

### 3.3. Lateral control policy

In the car-following task, we were usually only concerned about longitudinal control, but because of the high fidelity of physic parameters in the CARLA simulator, during the training, there were frictions between the tires and the road, which cause very small heading drift error, usually $0.5°$ for 1000 training steps. To correct this heading drift error and eliminate effects on longitudinal control, we apply Stanley controller proposed by Thrun et al. (2006) as the lateral control policy (steering angle) to keep the vehicle in the middle of the road. The considered controller is one type of geometrical path-tracking controller, which helped Stanford to win the DARPA challenge in 2006. Moreover, since it considers both the lateral and the heading error, it shows a very good performance. In a Stanley controller, the steering angle $\phi(t)$ of the vehicle is given by:

$$\phi_t = \theta_p(t) + \tan^{-1}\left\{\frac{k_v\, d_f(t)}{v(t)}\right\}.$$

where $\theta_p(t)$ compensates the angular error $\theta_p$ and the second term compensates the front lateral distance error $d_f$ measured from the center of the front axle to the nearest point on the path. Furthermore, $\frac{v}{k_v}$ is a headway distance and $k_v$ is a normalizing constant.

### 3.4. Real-world human driving datasets

We use the real-world car-following trajectory from Punzo et al. (2005) as our real-world human demonstration, which we later on call "Napoli datasets". It is created and processed after a platoon driving experiment, in which several vehicles follow each other on urban roads and highways. The Napoli datasets were collected on two days along the same route on two urban roads and one rural road. It consists of five sub-dataset with an overall amount of 24 min driving data. The first urban road is a 2 km long straight road with congested traffic (i.e., stop-and-go traffic conditions). It has four intersections and an estimated capacity of 900 vehicles per hour (veh/h). The second urban road is also approximately 2 km long but with an estimated capacity of 1200 veh/h. The rural road is a two-lane 3 km long highway with an estimated capacity of 1500 veh/h. The traffic flow during collecting datasets was approximately 400 veh/h. With a non-stationary Kalman filter (Kalman, 1960) applied to position data, high-quality car-following

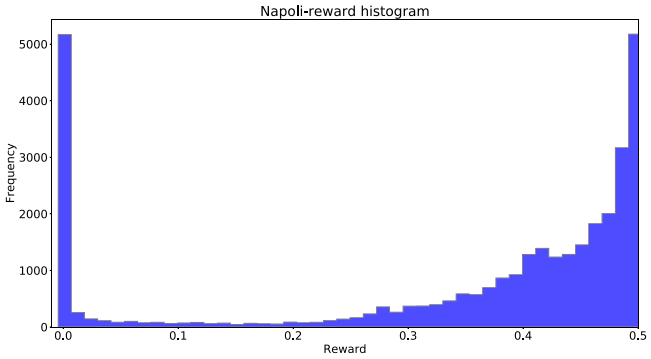



**Fig. 3.** Reward distribution in Napoli datasets according to the reward function in Section 3.2.2.

data was provided, including the velocity of each vehicle and bumper-to-bumper distance between leader–follower pair with the frequency of 10 Hz.

To store the driving datasets into replay buffer, we need to construct lists of tuples $d_i = (s_i, a_i, r_i, s_{i+1})$. Since the states and actions at each time step are already known from the datasets, only rewards are missing. To compute the reward for each action the human driver made, we use the same reward function (6) as for the DDPG agent. The histogram of the resulted rewards for the human drivers is shown in Fig. 3. As the maximum reward $r_t$ agent can obtain is 0.5, we see that under this reward function setting, half of the actions performed by the human driver can be considered as good ($r_i \geq 0.4$). This means that we can expect that the agent will learn some good behaviors from human drivers. Nevertheless, we still observe a large fraction of actions with zero reward, which indicates that these actions executed by human drivers result in a higher time gap above the upper time gap limit in (8) and Table 1. We separate the Napoli dataset into two parts, 95% of the data for training, and 5% of the data for evaluation. For the second stage of training, we utilize training data and let the agent improve its driving policy by leverage between real driving data and online experience from interaction with the simulator. Furthermore, the real driving actions with lower rewards work as negative incentives which can exclude some undesirable behaviors. Moreover, the Next Generation Simulation (NGSIM) trajectory dataset is used as another real-world driving dataset for evaluation in Section 5.2.

## 4. Experimental setup

### 4.1. Simulation environment

As discussed in Section 2.5, we use CARLA being our simulation environment. Since CARLA is grounded on Unreal engine, the static objects like infrastructure, buildings, or vegetation, and dynamic objects such as vehicles, cyclists, or pedestrians are made by 3D models. For the car-following task, we just need leading and following vehicles running on the road. To save computing resources and speed up the algorithms, we choose the OpenDRIVE standalone mode of CARLA for simulation. The standalone mode runs the full simulation by only using the OpenDRIVE file which describes every detail on the road. Other additional geometries or assets such as buildings or vegetation will not be created, as shown in Fig. 4. The simulator will take the OpenDRIVE *.xodr* file and procedurally create temporal 3D meshes which describe the road definition in a minimalistic manner. Moreover, to prevent vehicles from falling off the road, visible walls are created at the boundaries of the road, to act as a safety measure.

### 4.2. CARLA-ROS co-simulation

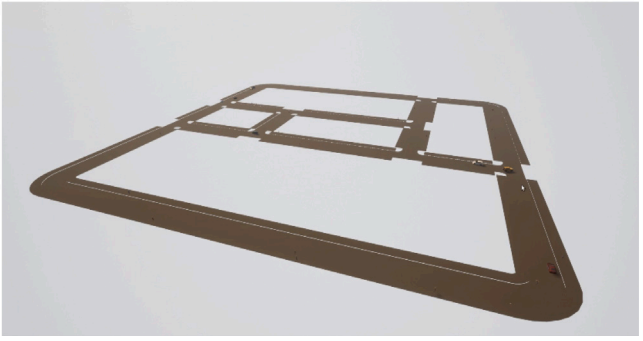
Although the hyper-realistic simulated environment of CARLA provides API to modify all aspects related to the simulation, to process the sensor information, and enable interoperability with control and perception modules, we should rely on ROS. In this paper, we use CARLA-ROS co-simulation to implement and evaluate our approach. For the car-following task, we need to ensure the communication between leader and follower, which ROS has strength at Hellmund et al. (2016). Furthermore, ROS provides GUI tools for diagnostics and monitoring during the development and run-time of the system. This allows us to have an immediate overview of the status of the system.

The transmission of information between two vehicles as well as the control node in ROS is realized via a transport system with *publish / subscribe* semantics. The control node can obtain required information from each vehicle, and publish control information to the agent, for instance, throttle and brake.

Here, *hero1* indicates the leader, and *hero2* means the follower in a leader–follower pair. Four different topics provide different information about the vehicles in CARLA:

1. */carla/hero/vehicle_status*: provides the current speed, acceleration, orientation of the vehicle, as well as the control values reported by CARLA.

(a) The common driving environment



(b) The OpenDRIVE standalone mode

**Fig. 4.** Two different driving simulation modes of CARLA: (a) the standard driving mode in CARLA with full rendering; (b) the OpenDRIVE standalone mode with minimalistic rendering for better simulation efficiency.

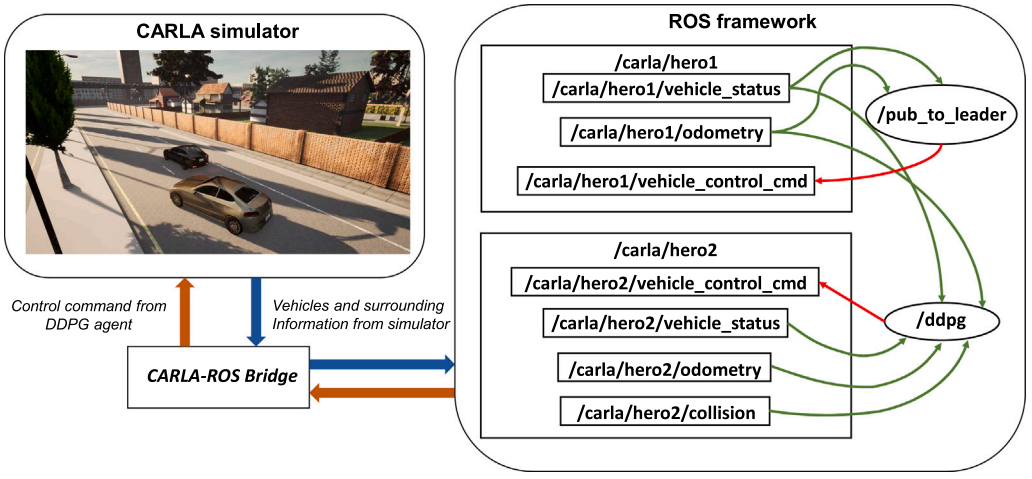



**Fig. 5.** The structure of our proposed CARLA-ROS framework, the CARLA-ROS bridge transport the information of vehicles and surrounding from CARLA into ROS, afterward, transfer the control command from DDPG agent in ROS to the vehicle in CARLA.

2. */carla/hero/odometry*: provides the current position of the vehicle in the specified coordinate frame which usually is Cartesian coordinates (UTM).
3. */carla/hero/vehicle_control_cmd*: messages sent to CARLA apply a control signal to the vehicle, which includes throttle: [0.0, 1.0], brake: [0.0, 1.0], steering: [−1.0, 1.0].

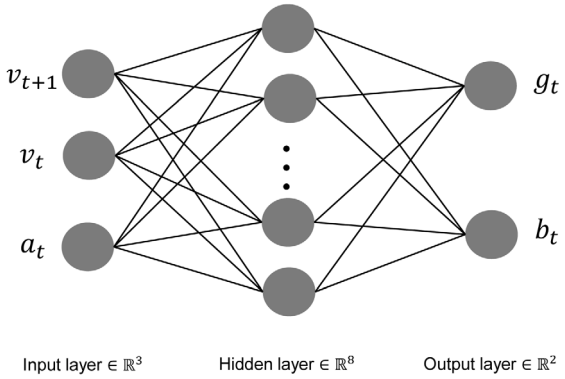



**Fig. 6.** Control neural network with inputs of $v_{t+1}$, $v_t$, $a_t$, and output $g_t$ and $b_t$.

4. */carla/hero/collision*: retrieves collision data detected by the collision sensor. It is used to reset the environment when a collision between two vehicles occurs.

As mentioned in Section 3.2 and shown in Fig. 5, neural networks that were updated and learned will output a control signal, such as acceleration of the agent, and will be transported back to CARLA to control the agent. However, in CARLA, there is no controller to directly use acceleration or speed to control vehicles which are our actions in DDPG. This can be performed only indirectly via throttle and brake. For this reason, we use the reverse data method to obtain a controller that controls vehicles via acceleration and speed, as discussed in details in the next section.

### 4.3. Controller for CARLA

For CARLA simulation, */carla/hero/vehicle_control_cmd* topic is used to control the vehicle via throttle and brake. But as discussed above, we should use acceleration as the output of our agent to ensure generalization. To overcome this problem, there are two feasible solutions: First, as vehicle models in CARLA are wrappers for the PhysX Vehicle by NVIDIA GameWorks, we can map the throttle and brake to velocity and acceleration with lists of equations according to the vehicle dynamic models. Nevertheless, in many cases, we cannot use this method, when the dynamic models of the object are unknown, for instance, with real vehicles connected via ROS. Second, we can use the reverse data method to train a simple *control neural network* via supervised fashion that links acceleration and velocity to throttle and brake. In this paper to allow for generalization, we go for the second approach.

At first, we run four hours of driving test with automatic control mode in CARLA. In this case, the vehicle is controlled automatically by CARLA with throttle and brake. During the driving, we collect datasets that include the velocity and acceleration of the vehicle at each time step, as well as the throttle and brake at the same time step that result in the corresponding acceleration and velocity. Second, we reorganize the collected dataset with the input states:

$$x_t = \begin{pmatrix} v_{t+1} \\ v_t \\ a_t \end{pmatrix},$$

where $v_{t+1}$ is the target velocity of next time step, $v_t$ is the current velocity, $a_t$ is the current acceleration. The output states of the neural network are:

$$y_t = \begin{pmatrix} g_t \\ b_t \end{pmatrix},$$

where $g_t$ and $b_t$ are throttle and brake at time step $t$ respectively. This procedure can be performed with any other simulation environment.

We use a simple fully connected neural network as shown in Fig. 6, and train it with the new reorganized dataset. Afterwards, if the proposed DDPG algorithm is planned to be applied to different models of vehicles, only this control neural network should be retrained. As an example, if we have a robot car in the real world, but the actual physic models are unknown. We can first collect the controlling data similarly as in CARLA, and then train the control network with the input of acceleration. Afterwards, this extra control network can be connected to our pretrained DRL policy network and resume training for some small episodes to achieve sufficient performance. Compared with the end-to-end DRL, this method greatly improves generalization.

### 4.4. Training process

To train car-following agents, leader–follower pairs are required. Thus, the Ornstein–Uhlenbeck process is used to generate random leader trajectories, an example leader velocity trajectory is illustrated in Fig. 7. Before each episode of the RL training

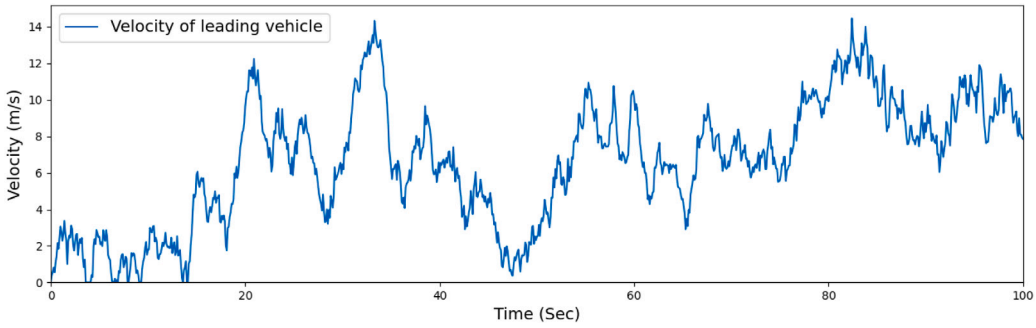



**Fig. 7.** An example leading vehicle velocity trajectory based Ornstein–Uhlenbeck process for the RL training process.

**Table 2**
Hyperparameters used in the simulation.

| Symbol | Description | Value |
|---|---|---|
| $l_r$ | Learning rate | 0.001 |
| $\gamma$ | Reward discount factor | 0.95 |
| $N_D$ | Size of the replay buffer | 2000 |
| $B$ | Training batch size | 32 |
| $\tau$ | Soft target update rate | 0.001 |
| $\theta_1$ | Parameter in Ornstein–Uhlenbeck process | $0.15\,\mathrm{s}^{-1}$ |
| $\sigma_1$ | Parameter in Ornstein–Uhlenbeck process | $0.20\,\mathrm{s}^{-0.5}$ |

1. $\theta_1$ and $\sigma_1$ were determined according to the suggestion in Lillicrap et al. (2015).

**Table 3**
Hyperparameters used for IDM model.

| Symbol | Description | Value |
|---|---|---|
| $v_{des}$ | Desired velocity | 20 m/s |
| $T$ | Safe time gap | 1 s |
| $a$ | Maximum acceleration | $2\,\mathrm{m/s}^2$ |
| $b_{comf}$ | Comfortable deceleration | $2\,\mathrm{m/s}^2$ |
| $g_{min}$ | Minimum gap | 2.5 m |

process, such random velocity trajectories will be generated, afterwards, the ROS node *pub_to_leader* receives the velocity commands from the generated trajectories and controls the leading vehicle to run with the desired velocity in CARLA. The agent aims to follow the leading vehicle with the obtainment of maximum rewards according to the reward function defined in Section 3.2.2. Furthermore, the initial speeds of the leader and the follower are set to 0 m/s, and the initial bumper-to-bumper gap is set randomly in the range [0 m, 100 m].

To balance exploration and exploitation, we add exploration noise following another zero-reverting Ornstein–Uhlenbeck process to the output of the DDPG policy network as suggested by Lillicrap et al. (2015), and we use clipping to ensure that the processed output does not exceed the boundary. The structure of actor and critic neural networks is discussed in Section 3.2.3. Before training our DDPG agent, we deliberately choose a list of hyperparameters for the learning as shown in Table 2.

We compare several approaches with different utilization rates for the real human driving dataset, to cover the whole spectrum between DRL and BC, and also include the IDM model (Treiber et al., 2000):

- BC: train a neural network same as the actor network in DDPG with real-world human driving datasets by supervised learning.
- IDM: the IDM parameters shown in Table 3 are calibrated according to the evaluation datasets.
- Fully off-policy DDPG: fill the replay buffer only with real-world human driving datasets and train the DDPG agent from scratch.
- Pure DDPG: only trained by interacting with the simulator, without any real-world human driving datasets.
- Two-stage DDPG: after training a pure DDPG, ratio $r$ of the replay memory is sampled from *practical replay buffer* which consists of real-world human driving datasets, and other $(1-r)$ from *simulation replay buffer* with self-generated experience by interacting with the environment. Afterwards, training is resumed. In this work, we choose the ratio $r$ from 0.1 to 1.0 with 0.1 step to cover the whole spectrum.

The training process runs on an NVIDIA GeForce RTX 3080 GPU and takes roughly two hours to finish 10000 steps of interaction with the simulated environment. The simulation interval is 0.1 s, which means the DRL agent makes decisions every 0.1 s corresponding to 10 HZ of real data observation. Our DDPG agents and fully off-policy DDPG are trained for one million timesteps

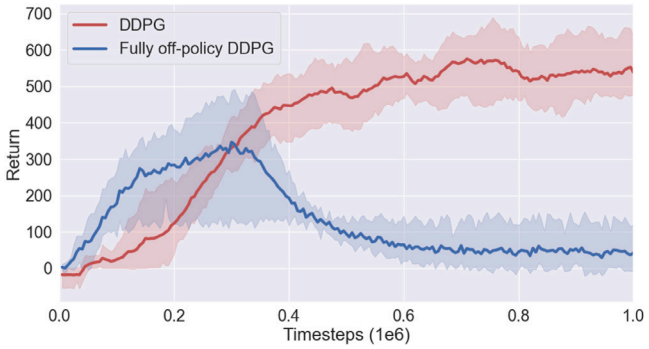



**Fig. 8.** Rewards obtained by the pure DDPG agent and fully off-policy DDPG agent during the training process.

multiple times with different initial seeds, with the reward obtained over time shown in Fig. 8. We see that the DDPG agent is converged during the training, with even more training steps, the performance will not be improved. However, the fully off-policy DDPG agent fails to learn from the real data. This result suggests that differences in the state distribution under the initial policies cause extrapolation error, which can drastically offset the performance of the fully off-policy agent. Furthermore, the BC agent is trained for 20 epochs with the real-world human driving dataset.

## 5. Evaluation and discussion

We apply the proposed methods and train an autonomous driving agent in the designed driving scenarios. Afterwards, methods are evaluated on the other set of scenarios.

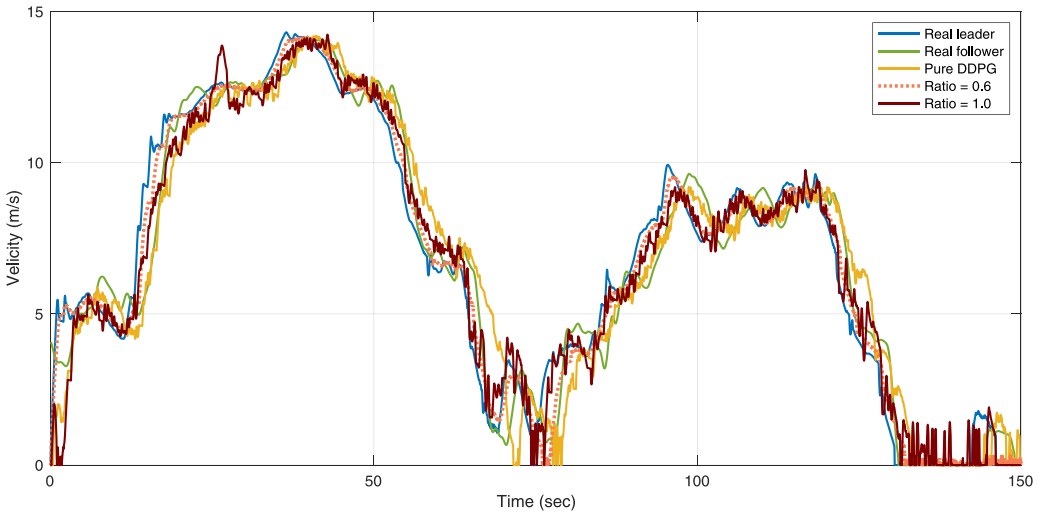
### *5.1. Evaluation with a leader trajectory from real-world driving dataset*

First, after the training process discussed in Section 4.4, we get several two-stage DDPG car-following agents. To verify that the trained two-stage DDPG agents learned human driving behaviors from real data, we compare the trained two-stage DDPG followers with pure DDPG follower and the real follower by following the evaluation leader trajectory from the Napoli dataset which has a car-following period of 150 s. The results are depicted in Fig. 9 and Appendix, where the velocity trajectory and the bumper-to-bumper distance between each follower and their leader are used. We see that the trained two-stage agents can follow the trajectory of the leading vehicle, whether it is in acceleration, deceleration, or standstill phases. The DDPG follower with 100% real data i.e. $Ratio = 1.0$, performs worse than the pure DDPG follower and keeps too much distance to the leading vehicle, moreover, the velocity profile of this follower is unstable with too much unnecessary acceleration and deceleration. The DDPG agent with the ratio of 1.0 real data starts from the pure DDPG, after training a pure DDPG, fill the replay buffer only with real-world human driving datasets, and resume training. Even with pretty good initial performance, the discrepancy between the trained policy and the policy in the datasets can offset the performance. On contrary, other two-stage DDPG followers also start from the pure DDPG agent but learn from real-world human demonstrations and interactive experiences. Even with different proportions of the real dataset i.e. from $Ratio \in \{0.1, \dots, 0.9\}$, with the help of the interactive experience from the training process, the estimation error from OOD will be corrected. Therefore, the two-stage DDPG improves its driving behavior from pure DDPG follower by learning from the real dataset. Nonetheless, different proportions of the real dataset in the replay buffer still cause different performances. The two-stage DDPG with $Ratio = 0.6$ has the best performance compare with other followers. In the high-speed driving phase, for driving efficiency, the vehicle maintains an appropriate distance while ensuring safety, and in the standstill phase keep a more safe distance to the leading vehicle. The two-stage DDPG agents modified its driving policy with the real driving dataset and tends to drive much closer to the leader compared with the pure DDPG agent. Noticing that during the standstill phases such as (T∈[75 s, 80 s]) and (T∈[130 s, 140 s]), when the leader is waiting for the traffic light in the signalized intersection, the DDPG with human experience agent still keeps small velocity to approach the leader with safe distance, and wait for the traffic light turning green. For further evaluation, we use the two-stage DDPG agent with $Ratio = 0.6$ as our working two-stage DDPG agent.

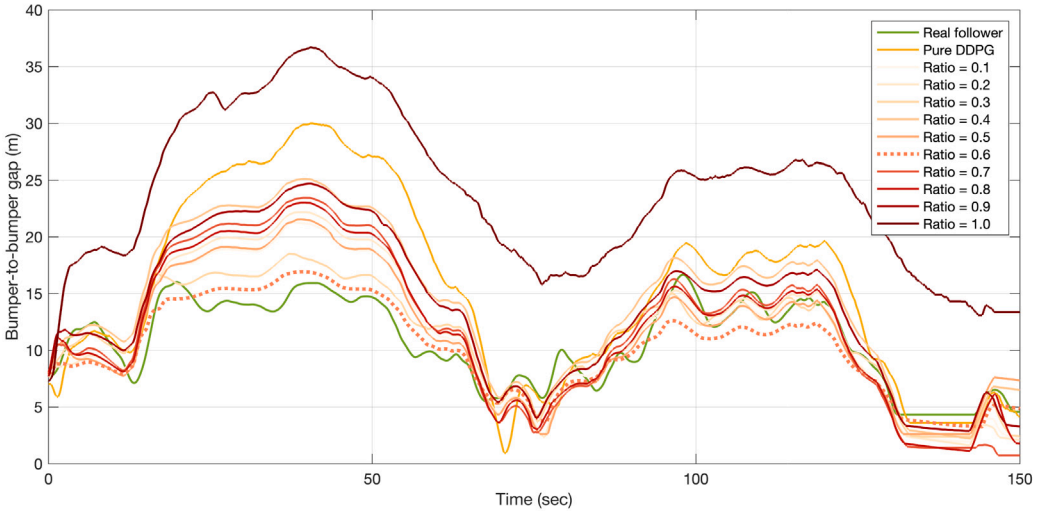
#### *5.1.1. Compare with other approaches*

Since we have compared different two-stage DDPG agents with different utilization ratios of the real dataset, a horizontal comparison with other approaches is also necessary. Here we compare the Two-stage DDPG agent ($Ratio = 0.6$) with BC agent, fully off-policy agent, IDM follower, and real follower. As illustrated in Fig. 10, we see that the BC agent and fully off-policy agent fail at the beginning of the evaluation. Fully off-policy DDPG agent suffer from extrapolation error that we introduced in Section 2.4, where the agent learns only with the real-world datasets from scratch, which means the action distribution generated by the initial policy has enormous difference to the action distribution of the real-world datasets, leading to a no-learning result.

The agent trained through the reward function shows a more passive driving style during the high-speed driving phase (T∈[15 s, 65 s] and T∈[90 s, 125 s]) and tends to be safe, therefore keeping a larger distance from the leader. However, in the low-speed




(a) Speed comparison of different followers



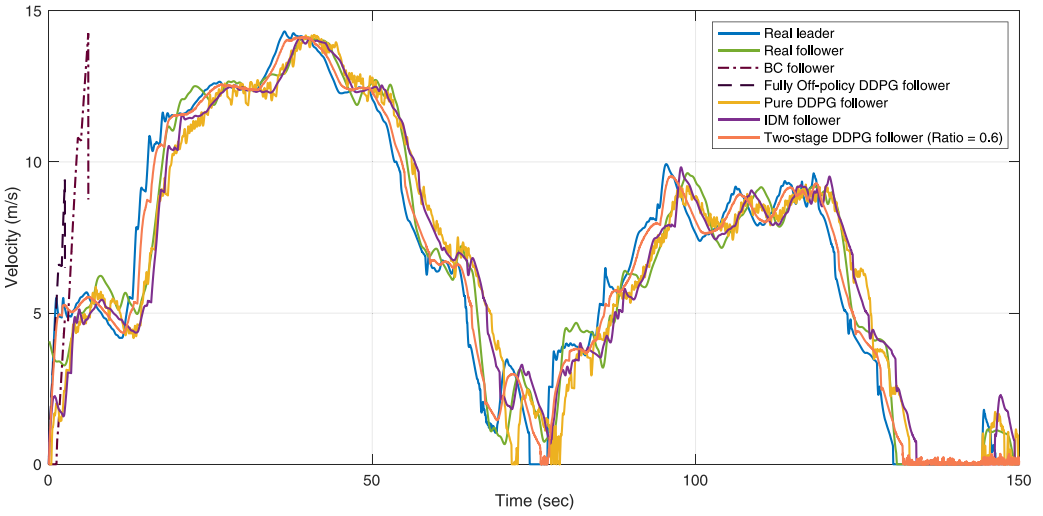

(b) Bumper-to-bumper gap comparison of different followers

**Fig. 9.** The evaluation results by following a real leader trajectory (blue) from Napoli datasets for two-stage DDPG followers with different utilization ratios of the real dataset. For better visualization, the velocity trajectories in (a) just show the $Ratio = 0.6$ and $Ratio = 1.0$ compared with pure DDPG and real follower, with the rest of the velocity trajectories shown in Appendix. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

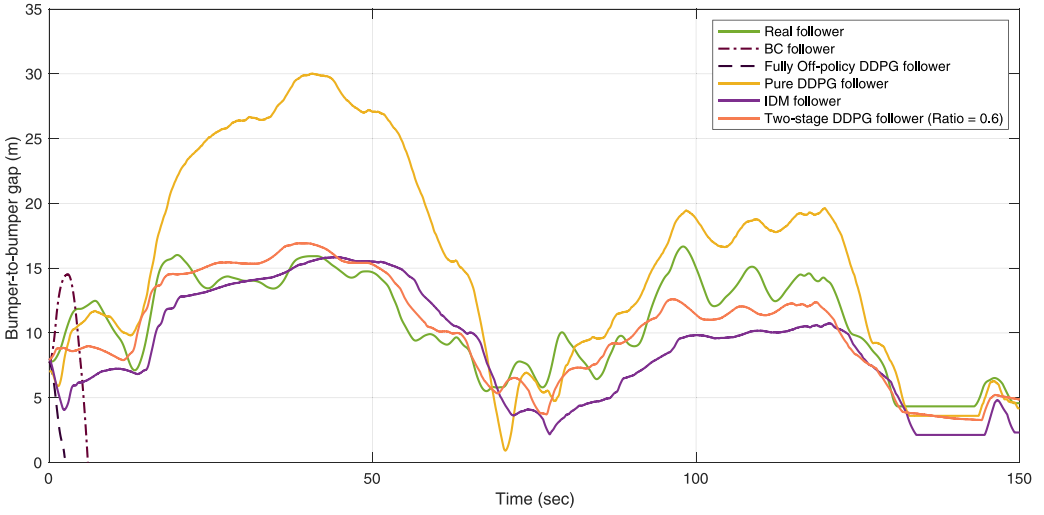
or standstill phase (T∈[130 s, 140 s]), the IDM model accustom to keep a short distance to the leader, which is a potential safety risk for not only the passenger of the ego vehicle but also for the leading vehicle driver. These immature driving behaviors are not suitable for real-world traffic. The two-stage DDPG agent modified its driving policy and tends to drive much closer to the leader compared to the pure DDPG agent during the high-speed phase, during the low-speed phase or standstill phase, the two-stage DDPG agent tries to stand farther to the leader than IDM follower. Moreover, from Fig. 10 we see that the two-stage DDPG follower has a shorter reaction time than other car-following agents.

### *5.1.2. Time-to-collision (TTC) analysis*

TTC is a widely used proximal safety indicator initially introduced by Hayward (1972). Fig. 11 shows the TTC below 100 s of every time step for real follower from dataset, pure DDPG agent, two-stage DDPG agent, and the IDM model. Table 4 shows the TTC distribution under 10 s for different followers. Since the fully off-policy DDPG agent and BC agent either has worse performance than pure DDPG or fails in the evaluation, these agents are excluded in TTC analysis.




(a) Speed comparison of different followers



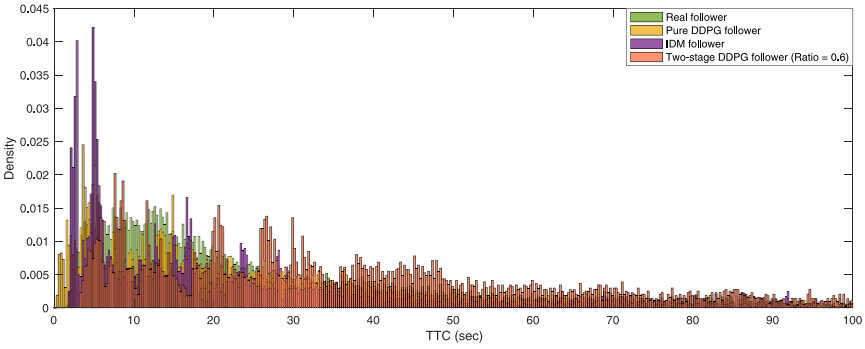

(b) Bumper-to-bumper gap comparison of different followers

**Fig. 10.** The evaluation results by following a real leader trajectory (blue) from Napoli datasets for different followers. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

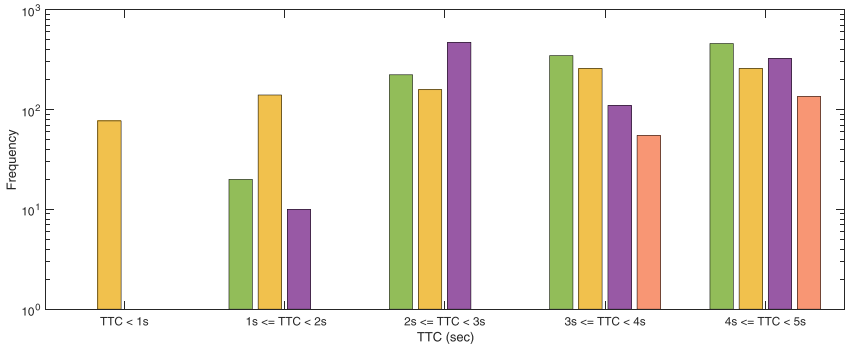
We consider TTC under 2 s as the safety–critical situation (Minderhoud and Bovy, 2001). As illustrated in Fig. 11, the pure DDPG follower, IDM model as well as the real follower suffer from TTC lower than 2 s, which indicate dangerous situations, but our two-stage DDPG follower has a higher lower bound than the other three agents. The lower TTC bound of the pure DDPG agent, real follower and IDM model is 0.48 s, 1.73 s and 1.98 s respectively, while the one for the two-stage DDPG agent is 3.37 s. Moreover, the TTC distribution of the two-stage DDPG is mainly concentrated around 7 s with smaller variance considering TTC under 10 s. The two-stage DDPG follower not only improves its driving policy with the real dataset but also learns to refrain from risky driving behavior implied in the real dataset. Thus, we see that the two-stage agent maintains a larger TTC than the pure DDPG agent, real follower, and IDM model while driving, which indicates better safety.

### *5.2. Evaluation with NGSIM dataset*

The NGSIM trajectory dataset is a well-known and de facto standard database, providing longitudinal and lateral positional information for about 3366 vehicle trajectories in certain spatiotemporal regions. In this work, we use NGSIM I-80 trajectory dataset as another evaluation database to compare the performance of the baseline algorithms and our two-stage DDPG follower. For the car-following task, since the drivers on the road are affected by the complex traffic environment, we need re-extract leader–follower




(a) Distribution of TTC under 100s




(b) Comparison for three agents with TTC less than 5s

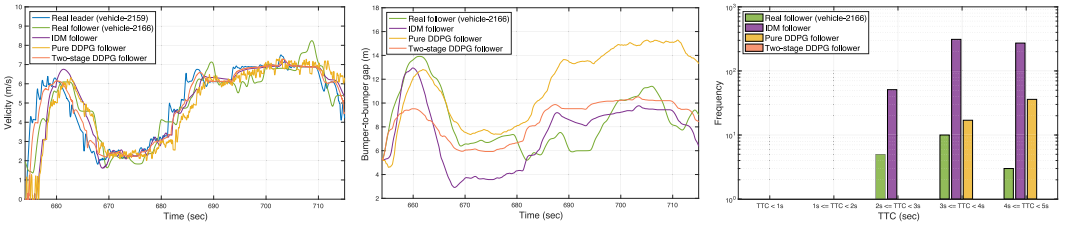
**Fig. 11.** Distribution of Time-to-collision (TTC) for real follower in dataset (green), pure DDPG agent (yellow), IDM (purple), two-stage DDPG agent (orange). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

**Table 4**
Computational results of different agents with TTC under 10 s.

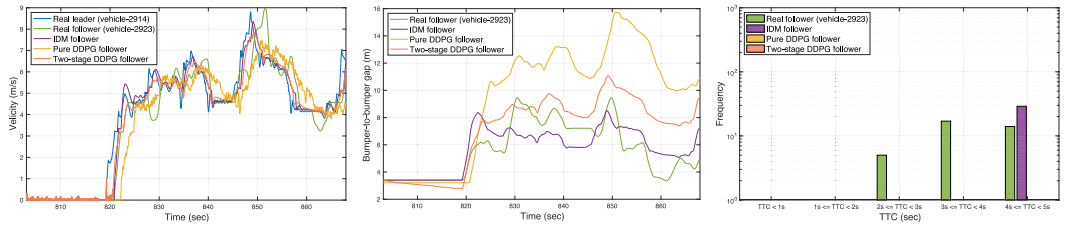
| | TTC (Sec) | | | |
|---|---|---|---|---|
| Agents | Minimum | Mean | Median | Std. dev |
| Real driver | 1.73 | 6.27 | 6.33 | 2.21 |
| IDM | 1.98 | 4.90 | 4.88 | 2.02 |
| Pure DDPG | 0.48 | 4.87 | 4.73 | 2.42 |
| Two-stage DDPG | 3.37 | 6.94 | 7.44 | 1.77 |

pairs from the original NGSIM dataset (Thiemann et al., 2008; Kesting and Treiber, 2008; Montanino and Punzo, 2013; Treiber and Kesting, 2013). After the re-extraction of the NGSIM I-80 datasets, we use five car-following pairs with car-following period greater than 60 s shown in Table 5 for evaluation. Same as the evaluation in Section 5.1.1, the proposed two-stage DDPG follower is compared with the real follower in the dataset, pure DDPG follower, and IDM follower, where each IDM follower is calibrated according to the evaluation database.

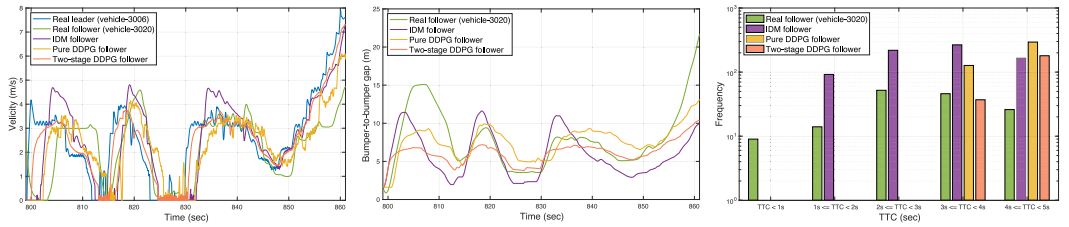
The evaluation results illustrated in Fig. 12 and Table 6 are similar to the results in Section 5.1.1. From the velocity trajectories and bumper-to-bumper gap figures, we observe that the two-stage DDPG followers have a shorter reaction time than other followers, and follow closer to the leading vehicle for travel efficiency compared with the passive pure DDPG follower. The real follower and IDM follower show more aggressive risky driving behavior and tend to drive too close to the leader. The hazards of this kind of reckless driving are reflected in the TTC analysis. As shown in the TTC figures, the real followers and IDM followers have smaller TTC, in cases 3, 4, and 5 which have stop-and-go traffic situations, even with TTC smaller than 2 s, which can be seen as dangerous situations. While the two-stage DDPG followers never have TTC smaller than 3.5 s with smaller variances and drive safer than pure DDPG follower with higher driving efficiency. It is important to mention that the NGSIM I-80 database is not used during the training process, but only for evaluation.



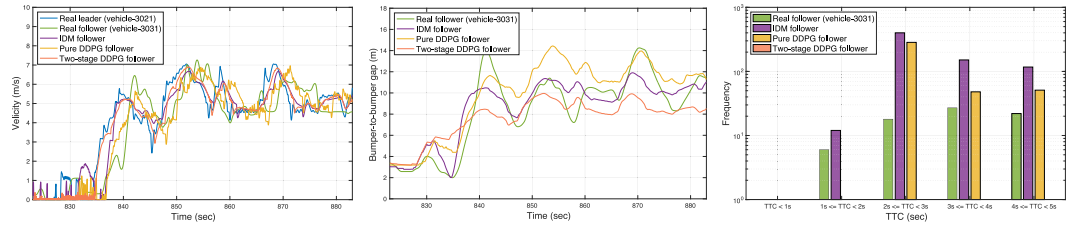

(a) Case 1: vehicle 2159 and vehicle 2166 as leader-follower pair (T = 654.0s∼715.0s)



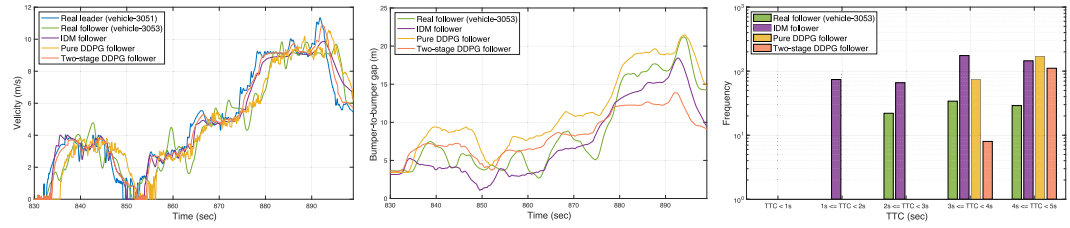

(b) Case 2: vehicle 2914 and vehicle 2923 as leader-follower pair (T = 803.0s∼868.0s)




(c) Case 3: vehicle 3006 and vehicle 3020 as leader-follower pair (T = 799.0s∼861.0s)




(d) Case 4: vehicle 3021 and vehicle 3031 as leader-follower pair (T = 823.0s∼883.0s)




(e) Case 5: vehicle 3051 and vehicle 3053 as leader-follower pair (T = 830.0s∼899.0s)

**Fig. 12.** We compare two-stage DDPG follower (red) with IDM follower (purple), pure DDPG follower (yellow), and real follower (green) following the real leader (blue) in the NGSIM I-80 dataset in CARLA. (a)∼(e) are the results following five different leader trajectories, from left to right are the velocity trajectories, bumper-to-bumper gap, and TTC distribution for different followers respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

**Table 5**
Five different car-following pairs are selected from the re-extracted NGSIM I-80 dataset.

| Case | Leader ID | Follower ID | Start time | End time |
|---|---|---|---|---|
| 1 | Vehicle 2159 | Vehicle 2159 | 654.0s | 715.0s |
| 2 | Vehicle 2914 | Vehicle 2923 | 803.0s | 868.0s |
| 3 | Vehicle 3006 | Vehicle 3020 | 799.0s | 861.0s |
| 4 | Vehicle 3021 | Vehicle 3031 | 823.0s | 883.0s |
| 5 | Vehicle 3051 | Vehicle 3053 | 830.0s | 899.0s |

**Table 6**
Evaluation results of TTC under 10 s for different agents in different evaluation cases from NGSIM dataset.

| | Agents | TTC (Sec) | | | |
|---|---|---|---|---|---|
| | | Minimum | Mean | Median | Std. dev |
| Case 1 | Real driver | 2.86 | 6.92 | 7.56 | 2.16 |
| | IDM | 2.36 | 5.64 | 5.18 | 2.00 |
| | Pure DDPG | 3.61 | 7.55 | 7.88 | 1.74 |
| | Two-stage DDPG | 5.89 | 8.41 | 8.62 | 1.29 |
| Case 2 | Real driver | 2.77 | 5.68 | 5.29 | 1.96 |
| | IDM | 4.51 | 7.69 | 7.94 | 1.56 |
| | Pure DDPG | 7.89 | 8.98 | 8.97 | 0.70 |
| | Two-stage DDPG | 6.41 | 7.91 | 7.79 | 1.05 |
| Case 3 | Real driver | 0.75 | 5.28 | 5.50 | 2.41 |
| | IDM | 1.80 | 6.43 | 7.12 | 2.40 |
| | Pure DDPG | 3.11 | 6.13 | 5.79 | 1.89 |
| | Two-stage DDPG | 3.51 | 6.81 | 6.25 | 1.94 |
| Case 4 | Real driver | 1.92 | 5.46 | 5.01 | 2.30 |
| | IDM | 1.98 | 4.65 | 3.65 | 2.62 |
| | Pure DDPG | 2.00 | 5.19 | 4.16 | 2.77 |
| | Two-stage DDPG | 5.35 | 7.98 | 8.07 | 1.23 |
| Case 5 | Real driver | 2.33 | 5.71 | 5.24 | 2.32 |
| | IDM | 1.19 | 5.91 | 6.41 | 2.56 |
| | Pure DDPG | 3.06 | 6.48 | 6.09 | 2.25 |
| | Two-stage DDPG | 3.90 | 7.31 | 7.36 | 2.25 |

### *5.3. Evaluation with self-defined leading vehicle speed profile*

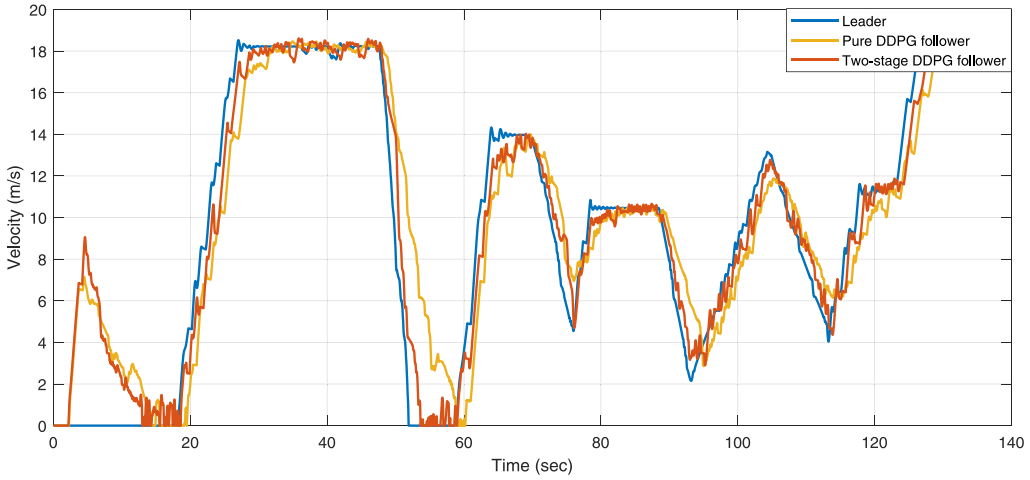
Next, we designed a leading vehicle speed profile that accounts for specific behavior not seen in the Napoli dataset and NGSIM dataset. First, we define a velocity trajectory for the leading vehicle with some safety–critical and common driving behavior. Fig. 13 illustrates the comparison of the performance of pure DDPG agent and two-stage DDPG agent.

The initial gap between followers and leader is 50 m, and the initial velocity of each follower is $0\,\mathrm{m/s}$. Both followers accelerate from a standstill state as they realize the big distance to the leader. By approaching the leader, both followers reduce their velocity to obtain a safe distance. As the leading vehicle starts to accelerate from $T = 18$ s and reaches the velocity of $18\,\mathrm{m/s}$, both followers follow this change. The pure DDPG agent is clearly much passive, needs more reaction time, and keeps a bigger distance during the constant velocity driving stage. Due to learning from the human driving behavior, the two-stage DDPG follower has a shorter reaction time and keeps much closer to the leader, which improves the driving efficiency.

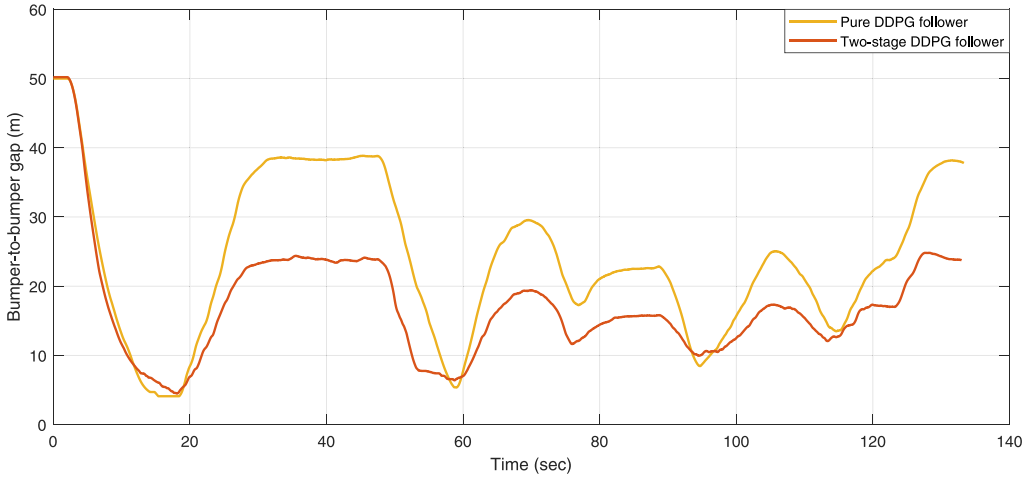
From $T = 48$ s, the leader performs safety–critical braking with the deceleration of $-5\,\mathrm{m/s^2}$ until it stops. Our two-stage DDPG follower notices this braking timely and starts to decelerate accordingly while keeping a safe distance all the time. Same as the starting point, the pure DDPG follower cannot sense this braking opportunely and leads to a smaller distance to the leader. Afterward, the leader drives with some common accelerate and decelerate scenarios, and the two-stage DDPG follower is able to keep a more appropriate distance than the pure DDPG follower.

## 6. Conclusion and outlook

In this study, we proposed a so-called two-stage DDPG method to combine the real-world human driving dataset with DRL method to obtain a car-following agent which has human experience. With this approach, the agent can modify its driving policy by extracting the good behaviors from real-world human demonstrations. This will help the autonomous agent adapt to the real-world traffic environment. Moreover, we also proposed a framework that uses ROS and hyper-realistic autonomous driving simulator CARLA together to overcome the communication problem for car-following tasks. The agent can obtain all the information from




(a) Speed comparison of different follower




(b) Bumper-to-bumper gap comparison of different follower

**Fig. 13.** The evaluation results by following an external leader trajectory for pure DDPG agent (yellow) and DDPG agent with human experience (red). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

other vehicles, for instance, position, velocity, acceleration, etc. With this framework. We used a small control network to connect to the policy network from DDPG for generalization. This control network can be retrained repeatedly for different types of vehicles whether in reality or simulation. Therefore, the trained DDPG networks can be reused many times. To the best of our knowledge, this is the first time that the CARLA-ROS communication method and the general control network are used for DRL in CARLA.

To evaluate the results, we designed different driving scenarios, analyzed the improvement of the two-stage DDPG agent qualitatively and quantitatively. The results showed that the two-stage DDPG agent was superior to the pure DDPG agent in terms of safety and driving efficiency. The proposed framework with CARLA-ROS communication made the car-following task with DRL in CARLA very easy to train and evaluate.

Although our research showed that combining the real-world human demonstrations improved the driving ability of the DDPG agent, there are still some suboptimal issues. One of them is that we calculate the reward for the actions in the real driving dataset based on the expert experience reward function. This cannot fully represent the behavior of the real driver, since this reward function is not the internal reward function when human drives in real-world traffic. A better solution is to use Inverse Reinforcement Learning (IRL) to obtain the internal reward function from the human driver. This will be discussed in the forthcoming paper.

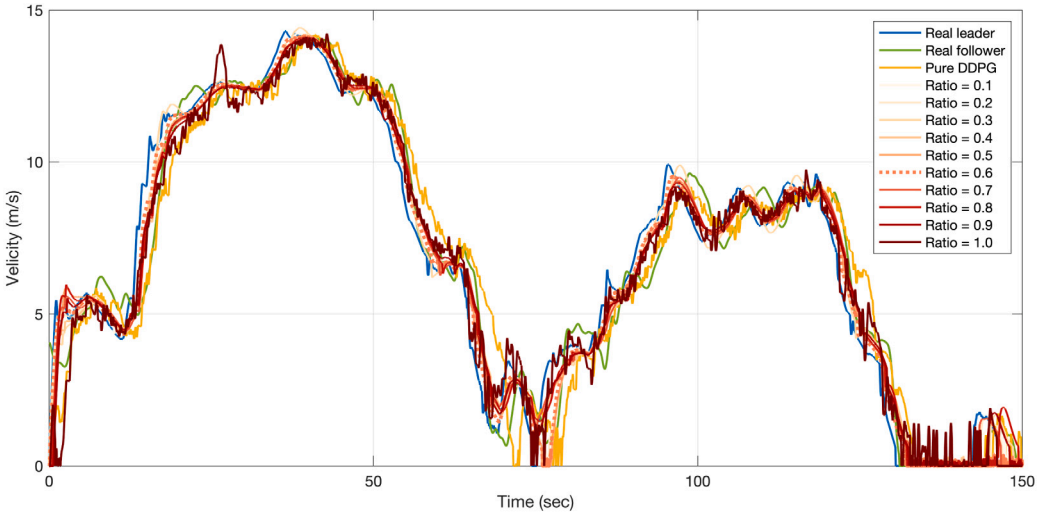



**Fig. A.14.** The evaluation results by following a real leader trajectory (blue) from Napoli datasets for two-stage DDPG followers with different utilization ratios of the real dataset. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

## CRediT authorship contribution statement

**Dianzhao Li:** Conceptualization, Methodology, Software, Formal analysis, Data curation, Writing – original draft. **Ostap Okhrin:** Conceptualization, Methodology, Formal analysis, Writing – reviewing and editing, Supervision.

## Data availability

The authors do not have permission to share data.

## Acknowledgments

This work was funded by ScaDS.AI (Center for Scalable Data Analytics and Artificial Intelligence) Dresden/Leipzig. We would like to thank Martin Treiber, Martin Waltz, and Fabian Hart for constructive criticism of the manuscript, and also thank Vincenzo Punzo and Martin Treiber for providing the Napoli and re-extracted NGSIM I-80 dataset.

## Appendix. Evaluation results of two-stage DDPG followers with different ratios

See Fig. A.14.

## References

Akhloufi, M.A., Arola, S., Bonnet, A., 2019. Drones chasing drones: Reinforcement learning and deep search area proposal. Drones 3, 58.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., et al., 2016. End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316.

Codevilla, F., Müller, M., López, A., Koltun, V., Dosovitskiy, A., 2018. End-to-end driving via conditional imitation learning. In: 2018 IEEE International Conference on Robotics and Automation. ICRA, IEEE, pp. 4693–4700.

Codevilla, F., Santana, E., López, A.M., Gaidon, A., 2019. Exploring the limitations of behavior cloning for autonomous driving. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 9329–9338.

Dadashi, R., Rezaeifar, S., Vieillard, N., Hussenot, L., Pietquin, O., Geist, M., 2021. Offline reinforcement learning with pseudometric learning. In: International Conference on Machine Learning. PMLR, pp. 2307–2318.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V., 2017. CARLA: An open urban driving simulator. In: Proceedings of the 1st Annual Conference on Robot Learning. pp. 1–16.

Dupuis, M., Strobl, M., Grezlikowski, H., 2010. OpenDRIVE 2010 and beyond–Status and future of the de facto standard for the description of road networks. In: Proc. of the Driving Simulation Conference Europe. pp. 231–242.

Dworak, D., Ciepiela, F., Derbisz, J., Izzat, I., Komorkiewicz, M., Wójcik, M., 2019. Performance of LiDAR object detection deep learning architectures based on artificially generated point cloud data from CARLA simulator. In: 2019 24th International Conference on Methods and Models in Automation and Robotics. MMAR, IEEE, pp. 600–605.

Fujimoto, S., Meger, D., Precup, D., 2019. Off-policy deep reinforcement learning without exploration. In: International Conference on Machine Learning. PMLR, pp. 2052–2062.

Gazis, D.C., Herman, R., Rothery, R.W., 1961. Nonlinear follow-the-leader models of traffic flow. Oper. Res. 9, 545–567.

Gómez-Huélamo, C., Del Egido, J., Bergasa, L.M., Barea, R., López-Guillén, E., Arango, F., Araluce, J., López, J., 2020. Train Here, Drive There: Simulating real-world use cases with fully-autonomous driving architecture in CARLA simulator. In: Workshop of Physical Agents. Springer, pp. 44–59.

Gu, S., Holly, E., Lillicrap, T., Levine, S., 2017. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In: 2017 IEEE International Conference on Robotics and Automation. ICRA, IEEE, pp. 3389–3396.
de Haan, P., Jayaraman, D., Levine, S., 2019. Causal confusion in imitation learning. Adv. Neural Inf. Process. Syst. 32, 11698–11709.
Hanin, B., Rolnick, D., 2018. How to start training: The effect of initialization and architecture. arXiv preprint arXiv:1803.01719.
Hart, F., Okhrin, O., Treiber, M., 2021. Formulation and validation of a car-following model based on deep reinforcement learning. arXiv preprint arXiv:2109.14268.
Hayward, J.C., 1972. Near-miss determination through use of a scale of danger. Highw. Res. Rec..
He, Y., Zhao, N., Yin, H., 2017. Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach. IEEE Trans. Veh. Technol. 67, 44–55.
Hellmund, A.-M., Wirges, S., Taş, Ö.Ş., Bandera, C., Salscheider, N.O., 2016. Robot operating system: A modular software framework for automated driving. In: 2016 IEEE 19th International Conference on Intelligent Transportation Systems. ITSC, IEEE, pp. 1564–1570.
Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., et al., 2018. Deep q-learning from demonstrations. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, pp. 3223–3230.
Huang, Z., Wu, J., Lv, C., 2021. Efficient deep reinforcement learning with imitative expert priors for autonomous driving. arXiv preprint arXiv:2103.10690.
Isele, D., Rahimi, R., Cosgun, A., Subramanian, K., Fujimura, K., 2018. Navigating occluded intersections with autonomous vehicles using deep reinforcement learning. In: 2018 IEEE International Conference on Robotics and Automation. ICRA, IEEE, pp. 2034–2039.
Kalman, R.E., 1960. A new approach to linear filtering and prediction problems. Trans. ASME D 35–45.
Kesting, A., Treiber, M., 2008. Calibrating car-following models by using trajectory data: Methodological study. Transp. Res. Rec. 2088, 148–156.
Kikuchi, S., Chakroborty, P., 1992. Car-following model based on fuzzy inference system. Transp. Res. Rec. 82–91.
Kim, D., 1999. Normalization methods for input and output vectors in backpropagation neural networks. Int. J. Comput. Math. 71, 161–171.
Levine, S., Kumar, A., Tucker, G., Fu, J., 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. arXiv preprint arXiv:2005.01643.
Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D., 2015. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.
Lin, L.-J., 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. Mach. Learn. 8, 293–321.
Litman, T., 2017. Autonomous Vehicle Implementation Predictions. Victoria Transport Policy Institute Victoria, Canada.
Liu, H., Huang, Z., Lv, C., 2021. Improved deep reinforcement learning with expert demonstrations for urban autonomous driving. arXiv preprint arXiv:2102.09243.
Minderhoud, M.M., Bovy, P.H.L., 2001. Extended time-to-collision measures for road traffic safety assessment. Accid. Anal. Prev. 33, 89–97.
Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.
Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al., 2015. Human-level control through deep reinforcement learning. Nature 518, 529–533.
Montanino, M., Punzo, V., 2013. Making NGSIM data usable for studies on traffic flow theory: Multistep method for vehicle trajectory reconstruction. Transp. Res. Rec. 2390, 99–111.
Ngai, D.C.K., Yung, N.H.C., 2011. A multiple-goal reinforcement learning method for complex vehicle overtaking maneuvers. IEEE Trans. Intell. Transp. Syst. 12, 509–522.
Niranjan, D.R., VinayKarthik, B.C., et al., 2021. Deep learning based object detection model for autonomous driving research using CARLA simulator. In: 2021 2nd International Conference on Smart Electronics and Communication. ICOSEC, IEEE, pp. 1251–1258.
Nosrati, M.S., Abolfathi, E.A., Elmahgiubi, M., Yadmellat, P., Luo, J., Zhang, Y., Yao, H., Zhang, H., Jamil, A., 2018. Towards practical hierarchical reinforcement learning for multi-lane autonomous driving. In: 2018 NIPS MLITS Workshop.
Ornstein, L.S., Uhlenbeck, G.E., 1930. On the theory of the Brownian motion. Phys. Rev. 36, 823.
Punzo, V., Formisano, D.J., Torrieri, V., 2005. Nonstationary Kalman filter for estimation of accurate and consistent car-following data. Transp. Res. Rec. 1934, 2–12.
Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y., et al., 2009. ROS: An open-source robot operating system. In: ICRA Workshop on Open Source Software, vol. 3. Kobe, Japan, p. 5.
Ravi Kiran, B., Roldao, L., Irastorza, B., Verastegui, R., Suss, S., Yogamani, S., Talpaert, V., Lepoutre, A., Trehard, G., 2018. Real-time dynamic object detection for autonomous driving using prior 3d-maps. In: Proceedings of the European Conference on Computer Vision, ECCV Workshops.
Sallab, A.E., Abdou, M., Perot, E., Yogamani, S., 2016. End-to-end deep reinforcement learning for lane keeping assist. In: Proceedings of MLITS, NIPS Workshop, vol. 2, pp. 1–9..
Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al., 2016. Mastering the game of go with deep neural networks and tree search. Nature 529, 484–489.
Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al., 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv preprint arXiv:1712.01815.
Thiemann, C., Treiber, M., Kesting, A., 2008. Estimating acceleration and lane-changing dynamics from next generation simulation trajectory data. Transp. Res. Rec. 2088, 90–101.
Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., et al., 2006. Stanley: The robot that won the DARPA grand challenge. J. Field Robotics 23, 661–692.
Tran, L.-A., Le, M.-H., 2019. Robust U-Net-based road lane markings detection for autonomous driving. In: 2019 International Conference on System Science and Engineering. ICSSE, IEEE, pp. 62–66.
Treiber, M., Hennecke, A., Helbing, D., 2000. Congested traffic states in empirical observations and microscopic simulations. Phys. Rev. E 62, 1805.
Treiber, M., Kesting, A., 2013. Traffic flow dynamics. In: Traffic Flow Dynamics: Data, Models and Simulation. Springer-Verlag Berlin Heidelberg, pp. 983–1000.
Treiber, M., Kesting, A., 2017. The intelligent driver model with stochasticity-new insights into traffic flow oscillations. Transp. Res. Procedia 23, 174–187.
Vecerik, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., Riedmiller, M., 2017. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. arXiv preprint arXiv:1707.08817.
Wang, P., Chan, C.-Y., de La Fortelle, A., 2018. A reinforcement learning based approach for automated lane change maneuvers. In: 2018 IEEE Intelligent Vehicles Symposium. IV, IEEE, pp. 1379–1384.
Wiedemann, R., 1974. Simulation des StraßEnverkehrsflusses. In: Schriftenreihe des IfV, vol. 8, Institut für Verkehrswesen, University of Karlsruhe, Germany.