Transportation Research Part C 117 (2020) 102662

Contents lists available at ScienceDirect

Transportation Research Part C

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

Safe, efficient, and comfortable velocity control based on reinforcement learning for autonomous driving

Meixin Zhu[a,b], Yinhai Wang[b,*], Ziyuan Pu[b], Jingyun Hu[b], Xuesong Wang[a,c,*], Ruimin Ke[b]

[a] School of Transportation Engineering, Tongji University, Shanghai 201804, China
[b] Department of Civil and Environmental Engineering, University of Washington, Seattle, 98195 Washington, United States
[c] Key Laboratory of Road and Traffic Engineering, Ministry of Education, Tongji University, Shanghai, China

ARTICLE INFO

Keywords:
Car following
Autonomous driving
Velocity control
Reinforcement learning
NGSIM
Deep Deterministic Policy Gradient (DDPG)

ABSTRACT

A model used for velocity control during car following is proposed based on reinforcement learning (RL). To optimize driving performance, a reward function is developed by referencing human driving data and combining driving features related to safety, efficiency, and comfort. With the developed reward function, the RL agent learns to control vehicle speed in a fashion that maximizes cumulative rewards, through trials and errors in the simulation environment. To avoid potential unsafe actions, the proposed RL model is incorporated with a collision avoidance strategy for safety checks. The safety check strategy is used during both model training and testing phases, which results in faster convergence and zero collisions. A total of 1,341 car-following events extracted from the Next Generation Simulation (NGSIM) dataset are used to train and test the proposed model. The performance of the proposed model is evaluated by the comparison with empirical NGSIM data and with adaptive cruise control (ACC) algorithm implemented through model predictive control (MPC). The experimental results show that the proposed model demonstrates the capability of safe, efficient, and comfortable velocity control and outperforms human drivers in that it 1) has larger TTC values than those of human drivers, 2) can maintain efficient and safe headways around 1.2s, and 3) can follow the lead vehicle comfortably with smooth acceleration (jerk value is only a third of that of human drivers). Compared with the MPC-based ACC algorithm, the proposed model has better performance in terms of safety, comfort, and especially running speed during testing (more than 200 times faster). The results indicate that the proposed approach could contribute to the development of better autonomous driving systems. Source code of this paper can be found at https://github.com/MeixinZhu/Velocity_control.

1. Introduction

Car following is the most frequent driving scenario. The main task of car following is controlling vehicle velocity to keep safe and comfortable following gaps. Autonomous car-following velocity control has the promise of mitigating drivers' workload, improving traffic safety, and increasing road capacity (Wang et al., 2013; Zhu et al., 2018a; Zhu et al., 2020).

* Corresponding authors at: Department of Civil and Environmental Engineering, University of Washington, Seattle 98195, Washington, United States (Y. Wang). Key Laboratory of Road and Traffic Engineering, Ministry of Education, Tongji University, Shanghai, China (X. Wang).
E-mail addresses: meixin92@uw.edu (M. Zhu), yinhai@uw.edu (Y. Wang), ziyuanpu@uw.edu (Z. Pu), jingyun@uw.edu (J. Hu), wangxs@tongji.edu.cn (X. Wang), ker27@uw.edu (R. Ke).

https://doi.org/10.1016/j.trc.2020.102662
Received 5 March 2019; Received in revised form 28 April 2020; Accepted 3 May 2020
Available online 12 June 2020

Driver models are critical elements of velocity control systems (Wang et al., 2016a; Wang et al., 2016b; Cheng et al., 2020). In general, the driver models related to car-following behavior are established based on two approaches: the rule-based approach and the supervised learning approach (Kuefler et al., 2017; Zhang et al., 2011). The rule-based approach mainly refers to traditional car-following models, such as the Gazis-Herman-Rothery model (Gazis et al., 1961) and the intelligent driver model (Treiber et al., 2000). The supervised learning approach relies on the data provided through human demonstrations to approximate the relationship between car-following states and vehicle acceleration actions.

These two approaches all intend to emulate human drivers' car-following behavior. However, solely imitating human driving behaviors may not be the best solution for autonomous driving. Firstly, users may not want autonomous vehicles driving in a way like them (Basu et al., 2017). Secondly, driving should be optimized with respect to safety, efficiency, and comfort, besides imitating human drivers because human drivers may not drive optimally (Chai and Wong, 2015).

To resolve these problems, this study proposes a car-following model for autonomous velocity control based on reinforcement learning (RL). This model directly optimizes driving safety, efficiency, and comfort, by learning from interactions with a simulation environment. Specifically, the deep deterministic policy gradient (DDPG) algorithm (Lillicrap et al., 2015) that performs well in the continuous control field is utilized to learn an actor-network together with a critic-network. The actor-network is responsible for policy generation: outputting following vehicle accelerations based on speed, relative speed, and clearance distance. The critic-network is responsible for policy improvement: update the actor's policy parameters in the direction of performance improvement. A reward function is developed by referencing human driving data and combining driving features related to safety, efficiency, and comfort. To avoid potential unsafe actions, the proposed RL model is incorporated with a collision avoidance strategy for safety checks. The safety check strategy is used during both model training and testing phases, which results in faster convergence and zero collisions.

To evaluate the proposed model, real-world driving data collected in the Next Generation Simulation (NGSIM) project (U.D. of Transportation, 2009) are used to train and test the model. The model is compared with empirical NGSIM data and also an adaptive cruise control (ACC) algorithm implemented through model predictive control (MPC), to demonstrate the model's ability to follow a leading vehicle safely, efficiently, and comfortably.

The *major contributions* of this paper are:

- Applied RL to real-world driving data for autonomous driving velocity control and developed a framework for multi-objective autonomous driving planning based on RL.
- Designed a new reward function that incorporates driving safety, efficiency, and comfort, which can lead to stable convergence.
- Incorporated RL with a kinematic collision avoidance strategy for a safety check, which resulted in faster convergence and zero collisions.

2. Background

2.1. Car following

Car-following models describe the movements of a following vehicle (FV) in response to the actions of the lead vehicle (LV) (Zhu et al., 2018a). They are essential components of microscopic traffic simulation (Brackstone and McDonald, 1999) and serve as theoretical references for autonomous car-following systems (Wei et al., 2010). The first car-following model (Pipes, 1953) was proposed in the middle 1950s, and a number of models have been developed since then, e.g., the Gazis-Herman-Rothery (GHR) model (Gazis et al., 1961), the intelligent driver model (IDM) (Treiber et al., 2000), the optimal velocity model (Bando et al., 1995), and the models proposed by Helly (1959), Gipps (1981), and Wiedemann (1974). A detailed review and historical development of this subject can be found in Brackstone and McDonald (1999) and Saifuzzaman and Zheng (2014).

2.2. Adaptive cruise control

Adaptive Cruise Control (ACC) system is an extension of the conventional Cruise Control (CC) system that helps maintain a constant vehicle velocity and is closely related to velocity control in the current study. Since constant velocity control becomes less useful under congested traffic situations, ACC extends CC functionality by dynamically adjusting the ACC host vehicle's velocity to maintain a proper clearance distance (a constant time headway policy is usually adopted to determine the distance) between the lead vehicle and the host vehicle. This is achieved by utilizing various sensors such as radar to measure the relative distance and relative velocity between the vehicles.

Model Predictive Control (MPC) tends to be the most frequently used control methods for designing ACC algorithms. MPC is a type of control that at each time step, a sequence of control inputs is obtained by solving a finite-horizon optimization problem and only the first element of the solved control sequence is applied (Camacho and Alba, 2013). This process repeats at the next time step with new measurements. One major advantage of MPC is its ability to handle constraints on control inputs (e.g., vehicle acceleration) and system states (e.g., following distance).

By using a linear and continuous model of car following, Luo et al. (2010) proposed an MPC controller for real-world car-following situations. The MPC tries to control the following vehicle's acceleration so that the relative distance between the two vehicles is in a safe region. Simulation results showed that the MPC controller demonstrated a safer behavior than that of real drivers. Takahama and Akasaka (2018) developed a practical MPC-based ACC algorithm with a low computational cost that can run on

embedded microprocessors. Specifically, a low-order prediction model was utilized to decrease the computation load. Results showed that their algorithm is with high responsiveness and less discomfort. Li et al. (2010) proposed an MPC-based ACC system to address the issues of tracking capability, fuel economy and driver desired response. A quadratic cost function was used to indicate tracking errors, fuel consumption, and accordance with driver characteristics. Simulations showed that the developed ACC system has significant benefits in terms of fuel economy, tracking capability, and satisfying driver desired car following characteristics. Lefevre et al. (2015) a learning-based method for autonomous car-following velocity control. A driver model was first developed to reproduce drivers' car-following behavior. The driver model's outputs (vehicle acceleration values) were then served as a reference by the MPC controller. By solving a constrained optimization problem, the MPC controller can ensure that the vehicle follows the model's behavior and also satisfies some safety criteria.

With the above-mentioned studies showing the power of MPC, this study chose RL for autonomous velocity control for two reasons: (1) RL is much faster than MPC during testing (Ernst et al., 2008). This is because MPC needs to solve a constrained finite-time optimal control problem at every time step while RL just needs to takes states as input and output actions; and (2) RL may have better performances than MPC as demonstrated by Lin et al. (2019).

2.3. Reinforcement learning

Reinforcement learning (RL) optimizes sequential decision-making problems by letting an RL agent interact with an environment. At time step t, the agent observes a state s_t and chooses an action a_t from some action space A based on a policy $\pi(a_t|s_t)$ that maps from state s_t to actions a_t. Meanwhile, the system gives a reward r_t to the agent, and transits to the next state s_{t+1}. This process continues until a terminal state is reached, then the agent restarts. The agent intends to get a maximum discounted, accumulated reward $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$, with the discount factor $\gamma \in (0, 1]$ (Li, 2017). In general, there are two types of RL methods: value-based and policy-based (Sutton and Barto, 1998).

2.3.1. Value-based reinforcement learning

A value function measures the quality of a state or state-action pair. The action value $Q^{\pi}(s, a) = E[R_t|s_t = s, a_t = a]$ is the expected return for selecting action a in state s and then following policy π. It represents the goodness of taking action a in a state s. Value-based RL methods intend to infer the action value function from historical experience. Q-learning is a typical value-based RL method. Beginning with a random Q-function, the agent keeps updating its Q-values based on the Bellman equation (Sutton and Barto, 1998).

$$Q(s, a) = E[r + \gamma \max a' Q(s', a')] \tag{1}$$

The intuition is that: maximum future reward for this state s and action a is the immediate reward r plus maximum future reward for the next state s'. Based on the estimated Q-values, the optimal policy is to take the action with the highest $Q(s, a)$ to get maximum expected future rewards.

2.3.2. Policy-based reinforcement learning

Different with value-based methods, policy-based methods try to improve the policy $\pi(a|s;\theta)$ directly, by updating its parameters θ with gradient ascent on $E[R_t]$. A typical policy-based method is REINFORCE, which updates the policy parameters θ with $\nabla_\theta \log \pi(a_t|s_t;\theta) R_t$ (Li, 2017).

To reduce the variance of policy gradients and increase learning speed, an actor-critic method is usually adopted. Two learning agents are used in an actor-critic algorithm: the actor (policy) and the critic (value function). The actor determines which action to take, and the critic tells the actor the quality of the action and how it should adjust the policy (Grondman et al., 2012).

2.4. Deep reinforcement learning

Deep reinforcement learning refers to reinforcement learning algorithms that use neural networks to approximate value function $V(s;\theta)$, policy$\pi(a|s;\theta)$, or system model.

2.4.1. Deep Q-Network

Instead of computing $Q(s, a)$ for each state-action pair, deep Q-learning uses neural network as function approximator to estimate the action-value function (Mnih et al., 2015). The action is selected with a maximum $Q(s, a)$ value. Deep Q networks (*DQN*) work well with discrete action spaces but fail in continuous action spaces, like in our case. To address this, Lillicrap et al. (2015) developed an algorithm called deep deterministic policy gradient (DDPG). DDPG introduced an actor-critic mechanism to *DQN* and can be used for continuous control problems.

2.4.2. Deep deterministic policy gradient

DDPG uses two separate networks to approximate the actor and critic respectively (Lillicrap et al., 2015). The critic network with weights θ^Q is responsible for estimating the action-value function $Q(s,a|\theta^Q)$. The actor network with weights θ^μ is responsible for explicitly representing the agent's policy $\mu(s|\theta^\mu)$. As proposed in *DQN*, experience replay and target network are adopted in DDPG to facilitate stable and robust learning.

- Experience replay
  A replay buffer was applied to avoid learning from sequentially generated, correlated experience samples. The replay buffer is a finite-sized cache D that stores transitions (s_t, a_t, r_t, s_{t+1}) sampled from the environment. The replay buffer is continually updated by replacing old samples with new ones. At each time step, the actor and critic networks are trained on random mini-batches of transitions from the replay buffer.
- Target network
  Target networks are used to represent target values of the main networks, to avoid divergence of the algorithm (Mnih et al., 2015). Two target networks, $Q'(s, a|\theta^{Q'})$ and $\mu'(s|\theta^{\mu'})$, were created for the main critic and actor networks respectively. They have the same architecture with the main networks but with different network parameters θ'. The parameters of target networks are updated by letting them slowly track the main networks: $\theta' = \tau\theta + (1-\tau)\theta'$ with $\tau \ll 1$. In this way, the target values are constrained to update slowly, greatly enhancing the stability of learning.

The full DDPG algorithm is listed in Algorithm I. It begins with initializing the replay buffer and the actor, critic and corresponding target networks. At each time step, an action a is taken according to the exploratory policy. Then, the reward r_t and new state s_{t+1} are observed and stored in the replay memory D. The critic is trained with mini-batches sampled from the replay memory. Afterward, the actor is updated by performing a gradient ascent step on the sampled policy gradient. Finally, the target networks with weights $\theta^{Q'}$ and $\theta^{\mu'}$ are updated to slowly track the actor and critic networks.

Algorithm 1. DDPG: Deep deterministic policy gradient for car-following velocity control

Randomly initialize critic $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ networks with weights θ^Q and θ^μ.
Initialize target network $Q'(s, a|\theta^{Q'})$ and $\mu'(s|\theta^{\mu'})$ with weights $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$
Set up empty replay buffer D
for episode = 1 to M **do**
  Begin with a random process N for action exploration
  Observe initial car-following state: initial gap, follower speed, and relative speed
  for t = 1 to T **do**
    Calculate reward r_t
    Choose follower acceleration $a_t = \mu(s_t, \theta^\mu) + N_t$ based on current actor network and exploration noise N_t
    Implement acceleration a_t and transfer to new state s_{t+1} based on kinematic point-mass model
    Save transition (s_t, a_t, r_t, s_{t+1}) into replay buffer D
    Sample random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from D
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic through minimizing loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update actor policy using 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 target networks: $\theta^{Q'} = \tau\theta^Q + (1-\tau)\theta^{Q'}$, $\theta^{\mu'} = \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
  end for
end for

3. Data preparation

Vehicle trajectory data of the Next Generation Simulation (NGSIM) project (U.D. of Transportation, 2009) were used in this study. The trajectory data were retrieved from the eastbound of I-80 in the San Francisco Bay area in Emeryville, CA, on April 13, 2005. The investigation region was around 500 meters (1,640 feet) long and comprised of six freeway lanes, including a high-occupancy vehicle (HOV) lane. An aggregate of 45 min of data are accessible in the full dataset, divided into three 15-min time spans: 4:00 p.m. to 4:15 p.m.; 5:00 p.m. to 5:15 p.m.; and 5:15 p.m. to 5:30 p.m. These periods contain the congestion buildup, or the inter-state between uncongested and congested traffic states, and full congestion during a peak period. The data provide precise location information for each vehicle, with the sampling rate being 10 Hz. To enhance data quality, the reconstructed NGSIM I-80 data (Montanino and Punzo, 2015) were utilized.

Car-following events were extracted by applying a car-following filter as described in Wang et al. (2018). A car-following event was defined as:

- The leading and following vehicle pairs stay in the same lane;
- Duration of the event > 15s: ensuring that the car-following persisted long enough to be analyzed.

A total of 1,341 car-following events were extracted and utilized in this study.

4. Features for reward function

In this section, features that capture relevant objectives of the car- following velocity control were proposed, with a final aim to

construct a proper reward function.

4.1. Safety

Safety should be the most important element of autonomous car following (Pu et al., 2020). Time to collision (TTC) was used to represent safety. As a widely used safety indicator, TTC represents the time left before two vehicles collide. It is computed as:

$$TTC(t) = -\frac{S_{n-1,n}(t)}{\Delta V_{n-1,n}(t)} \tag{2}$$

where t stands for time; $n-1$ and n represent the lead and following vehicles respectively; $n-1$, n combination denotes variables related to both the lead and following vehicles: $S_{n-1,n}$ being the clearance distance, and $\Delta V_{n-1,n}$ being the relative speed (lead-vehicle speed – following-vehicle speed).

TTC is inversely related to crash risk (smaller TTC values correspond to higher crash risks and vice versa) (Vogel, 2003). To apply TTC as a feature reflecting safety, a safety limit (a lower bound of TTC) should be determined. However, different thresholds (from 1.5s to 5s) are reported in the literature (Vogel, 2003). In this study, we tried safety limits from 1s to 9s and found they did not have a huge impact on the final car-following performance. A final limit of 4s was used because it led to the best overall performance. The TTC feature was constructed as:

$$F_{TTC} = \begin{cases} \log(\frac{TTC}{4}) & 0 \leqslant TTC \leqslant 4 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

In this way, if TTC is less than 4s, the TTC feature will be negative. And as TTC approaches zeros, the TTC feature will be close to negative infinity, which represents a severe punishment to near-crash situations.

4.2. Efficiency

Efficient driving in this study refers to keeping a safe and short time headway. Time headway is defined as the passed time between the arrival of the lead vehicle (LV) and the following vehicle (FV) at a designated point. Keeping a short headway within the safety bounds can improve traffic flow efficiency because short headways correspond to large roadway capacities (Zhang et al., 2007).

The rules of different countries are not quite the same, in regard to the legal or recommended time headway. In the U.S., several driver training programs state that it is difficult to follow a vehicle safely with headway being less than 2 s. In Germany, the recommended time headway is 1.8 s, and fines are imposed when the time headway is less than 0.9 s. In Sweden, the police use a time headway of 1 s as a threshold for imposing fines (Vogel, 2003).

This study determined the appropriate time headway based on the empirical NGSIM data. Fig. 1 presents the distribution of time headway in all of the extracted 1,341 car-following events. A lognormal distribution was fit on the data. The lognormal distribution is a probability distribution whose logarithm has a normal distribution. The probability density function of the lognormal distribution is:

$$f_{\text{lognorm}}(x|\mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{\frac{-(\ln x-\mu)^2}{2\sigma^2}}; x > 0 \tag{4}$$

where x is the distribution variable, time headway in this study, and μ, σ are the mean and log standard deviation of the variable x,

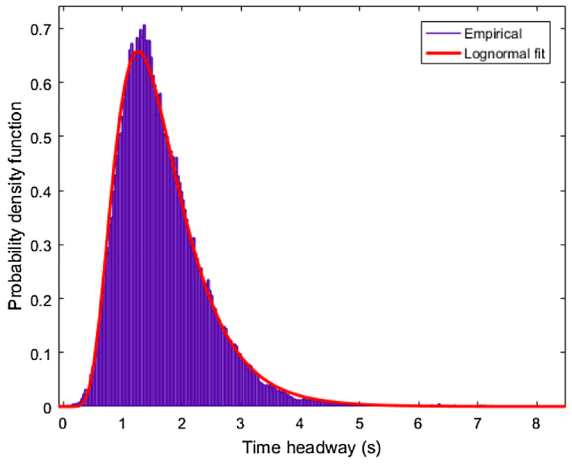

Fig. 1. Distribution of time headway in car-following events extracted from the NGSIM data.

respectively. Based on the empirical data, the estimated μ and σ were 0.4226 and 0.4365 respectively.

A headway feature was constructed as the probability density value of the estimated headway lognormal distribution:

$$F_{headway} = f_{\text{lognorm}}(headway|\mu = 0.4226, \sigma = 0.4365) \tag{5}$$

According to this headway feature, headway being 1.26s corresponds to the largest headway feature value (about 0.65); while headways being too long or too short correspond to low feature values. In this way, the RL agents are encouraged to keep a constant time headway around 1.26s. Note that to encourage a constant time headway, a density function of a normal distribution can also be used, but we found that the fitted lognormal density function happened to be better than a normal counterpart, which leads to unstable model performances.

4.3. Comfort

Jerk, defined as the change rate of acceleration, was used to measure driving comfort because it has a strong influence on the comfort of the passengers (Jacobson et al., 1980). A jerk feature was constructed as:

$$F_{jerk} = -\frac{jerk^2}{3600}, \tag{6}$$

with smaller values of jerk features corresponding to less comfortable driving. The squared jerk was divided by a base value (3600) to scale the feature into the range of [0 1]. The base value was determined by the following intuition:

1. The sample interval of the data is 0.1s;
2. The acceleration is bounded between -3 to 3 m/s^2 based on the observed FV acceleration of all the car-following events;
3. Therefore the largest jerk value is $\frac{3-(-3)}{0.1} = 60$ m/s^3, if squared, we get 3600.

5. Proposed approach

Since vehicle acceleration is a continuous variable, deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015) algorithm was used. In this section, the approach proposed to learn the velocity control strategy using DDPG is explained.

5.1. State and action

At a certain time step t, the state of a car-following process is described by the FV speed $V_n(t)$, clearance distance $S_{n-1,n}(t)$, and relative speed $\Delta V_{n-1,n}(t)$. The action is the longitudinal acceleration of the FV $a_n(t)$. Given state and action at time step t, the next-step state is updated by a kinematic point-mass model:

$$\begin{aligned} V_n(t+1) &= V_n(t) + a_n(t) * \Delta T \\ \Delta V_{n-1,n}(t+1) &= V_{n-1}(t+1) - V_n(t+1) \\ S_{n-1,n}(t+1) &= S_{n-1,n}(t) + \frac{\Delta V_{n-1,n}(t) + \Delta V_{n-1,n}(t+1)}{2} * \Delta T \end{aligned} \tag{7}$$

where ΔT is the simulation time interval, set as $0.1s$ in this study, and V_{n-1} is the velocity of lead vehicle (LV), which was externally inputted.

5.2. Simulation setup

To enable the RL agent to learn from trial and error, a simple numerical car-following simulation environment was implemented. The simulation only involves two agents: the LV and the FV, with the LV following empirical data while the FV controlled by the RL algorithm. Initialized with the empirically given following vehicle speed, clearance distance and velocity differences, $V_n(t=0) = V_n^{data}(t=0)$, $S_{n-1,n}(t=0) = S_{n-1,n}^{data}(t=0)$, and$\Delta V_{n-1,n}(t=0) = \Delta V_{n-1,n}^{data}(t=0)$, the RL agent is used to compute the acceleration $a_n(t)$ of the FV. Given acceleration, future FV velocity, relative speed, and clearance distance are then generated iteratively based on Eq. (7). At each time step, the simulation environment provide a reward value (calculated based on time headway, TTC, and jerk) to the RL agent as feedback. Once a car-following event reaches its ending, the state is re-initialized with empirical data of the next event. The events were randomly shuffled to avoid the impact of sequence.

5.3. Reward function

The reward function, $r(s, a)$, serves as a training signal to encourage or discourage behaviors in the context of a desired task. For the task of autonomous car following, a reward function was established based on a linear combination of the features constructed in Section 4:

$$r = w_1 F_{TTC} + w_2 F_{headway} + w_3 F_{jerk} \tag{8}$$

where w_1, w_2, and w_3 are coefficients of the features, all set as 1 in the current study.

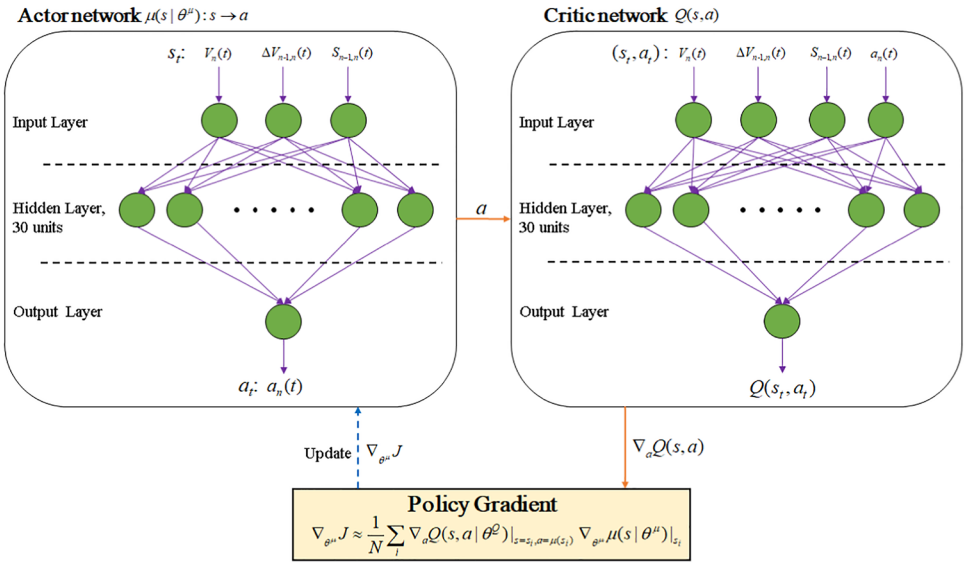

Fig. 2. Architecture of the actor and critic networks.

5.4. Network architecture

The actor and critic was each represented by a neural network. The input of the actor network is the state at time step t, $s_t = (V_n(t), \Delta V_{n-1,n}(t), S_{n-1,n}(t))$. Its output is FV's acceleration $a_n(t)$. The input of the critic network is a state-action pair (s_t, a_t). Its output is a scalar Q-value $Q(s_t, a_t)$.

Fig. 2 presents the architectures of the actor and critic networks (Zhu et al., 2018b). Both of them consist of three layers: an input layer, an output layer, and a hidden layer with 30 neurons. Deeper neural networks with more than one hidden layer were also tested, but the results showed that they did not perform significantly better.

For the hidden layers, the Rectified Linear Unit (ReLU) activation function ($f(x) = \max(0, x)$) was used. The ReLU can accelerate the convergence of network parameter optimization (Krizhevsky et al., 2012; Huang et al., 2019). For the output layer of the actor network, a *tanh* activation function was used. The *tanh* function maps real-valued numbers to the range $[-1, 1]$ and thus can bound the outputted accelerations between -3 to 3 m/s^2.

5.5. Network update and hyper parameters

The parameters of the networks were updated based on Kingma and Ba (2014) optimization algorithm. The critic network was updated by minimize the loss function $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$; the actor network was updated according to the gradient $\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s_i}$ (Lillicrap et al., 2015).

The hyperparameters (parameters set prior to the training process) adopted are presented in Table 1, these values were determined according to Lillicrap et al. (2015) and also by performing a test on a randomly sampled training dataset.

5.6. Exploration noise of action

An exploration policy was constructed by adding noise sampled from a noise process to the original actor policy. as suggested by Lillicrap et al. (2015), an Ornstein–Uhlenbeck process (Uhlenbeck and Ornstein, 1930) with $\theta = 0.15$ and $\sigma = 0.2$ was used. The Ornstein–Uhlenbeck process models the velocity of a Brownian particle with friction, generating temporally correlated values centered around zero. The temporally correlated noise enables the agent to explore well in a physical environment that has momentum.

Table 1
Hyperparameters and corresponding descriptions.

Hyperparameter	Value	Description
Learning rate	0.001	The learning rate used by Adam
Discount factor	0.90	Discount factor gamma used in the Q-learning update
Minibatch size	1024	Number of training cases over which each stochastic gradient descent (SGD) update is computed
Replay memory size	20000	Number of training samples in the replay memory
Soft target update τ	0.001	The update rate of target networks

5.7. Collision avoidance strategy

Although the safety rewards will penalize situations with small TTC values, it is still possible that the agent will take unsafe actions that lead to collisions, even after convergence. These potential collisions are not acceptable for safety-critical applications like autonomous driving. They can also result in simulation resets and thus slow down the training process (Nageshrao et al., 2019). To address this issue, several approaches have been proposed in previous studies, like barrier functions (Cheng et al., 2019) and explicit reference governor (Liu et al., 2019). In this study, we incorporated RL with a collision avoidance strategy for safety check that is used both while training and also during the implementation phase. This solution turned out to be effective and easy to understand.

Specifically, a kinematic-based stop distance algorithm (Wilson et al., 1997) was used. The algorithm determines the situation is unsafe if the range between the LV and FV is less than a safe distance threshold d_{safe} as defined by the following equation:

$$d_{safe} = V_n RT + \frac{V_n^2}{2a_{max}} - \frac{V_{n-1}^2}{2a_{max}} \tag{9}$$

where RT is the FV's reaction time (specified as 1s in this study), and a_{max} is the assumed max absolute deceleration rate (3 m/s^2). This basic idea behind this stop distance algorithm is that if the vehicle can keep a following distance larger than d_{safe} it should be able to avoid the collision in case the LV suddenly takes a full brake. During both the training and testing phases of the DDPG model, the collision avoidance algorithm was integrated with the RL algorithm in the following way:

$$a_n(t) = \begin{cases} -3m/s^2 & S_{n-1,n} < d_{safe} \\ \text{DDPG model output} & \text{otherwise.} \end{cases} \tag{10}$$

That is, once the following distance is less than d_{safe}, the vehicle will take a hard brake, otherwise, it will follow the DDPG model's outputs. In this way, the collision avoidance algorithm acts as a teacher and provides corrective action when necessary. The incorporation of collision avoidance strategy resulted in faster convergence (as will be mentioned in the next section) and zero collisions both while training and also during testing (without collision avoidance, 447 collisions occurred during 3000 episodes of training).

5.8. Training the DDPG velocity control model

For the 1,341 extracted car-following events, 70% (938) were used for training, and 30% were used for testing. At the training stage, the RL agent sequentially simulates the randomly shuffled car-following events in the training data. That is, when a car-following event terminates, a new event is randomly selected from the 938 training events, and the state of the agent is initialized with the empirical data of the new one. The training was repeated for 3000 episodes. An episode in this study means a car-following event.

Fig. 3 shows the changing of rolling mean episode reward with respect to the training episode. The mean episode reward is the average reward aggregated across all the time steps (sampling interval = 0.1s) of a car-following event, and the rolling mean episode reward is the average of mean episode rewards across a rolling window with size 100. Multiple runs of training were conducted and the results were aggregated: solid colored lines represent the mean across multiple runs and shaded areas represent the mean ± standard deviation interval. For easy interpretation of the reward values, the mean performances (aggregated across all the training episodes) of human drivers and the MPC-based ACC algorithm are also added for references.

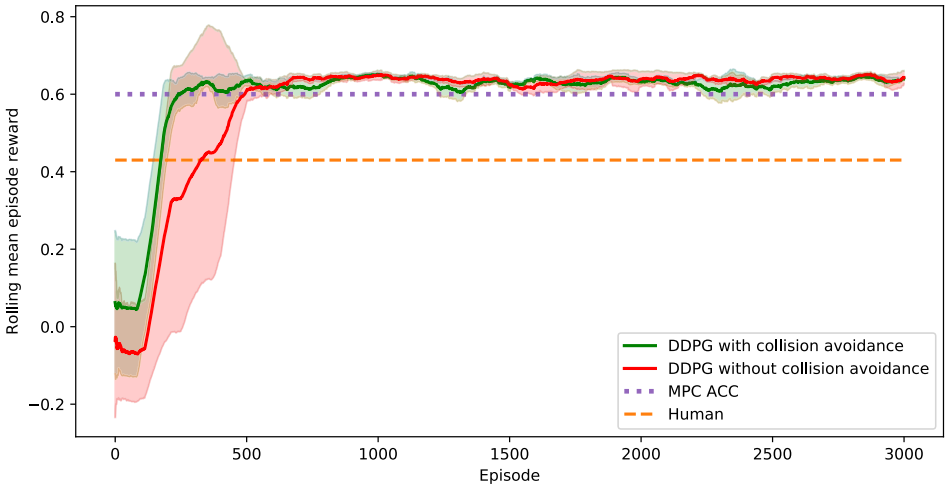

Fig. 3. Changing of reward during the training. Solid colored lines represent the mean and shaded areas represent the mean ± standard deviation interval. For easy interpretation, the mean performances of human drivers and MPC-based ACC algorithm are also added for references.

As can be seen, with the collision avoidance strategy, the DDPG model starts to converge when the training episode reaches around 250, faster than the one without collision avoidance strategy. When the model converges, the agent receives a reward value of about 0.64. This is achieved by selecting actions in a way that makes TTC and jerk feature values near 0 and get maximum headway features (0.65).

5.9. ACC baseline based on MPC

This section describes the ACC baseline algorithm implemented through MPC.

5.9.1. System model

The same kinematic point-mass model as mentioned in Equation was used but was written in matrix form as:

$$\boldsymbol{x}(t+1) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) \tag{11}$$

where t is the sampling time step (sampling interval = 0.1s), $\boldsymbol{x}(t) = (S_{n-1,n}(t), \Delta V_{n-1,n}(t), V_n(t))^T$, $\boldsymbol{u}(t) = a_n(t)$, and

$$\boldsymbol{A} = \begin{bmatrix} 1 & \Delta T & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{12}$$

$$\boldsymbol{B} = \begin{bmatrix} -0.5\Delta T^2 \\ -\Delta T \\ \Delta T \end{bmatrix}. \tag{13}$$

5.9.2. Control objectives and constraints

The primary goal of an ACC is to follow the lead vehicle at a desired distance $\widetilde{S}_{n-1,n}$. A constant time headway t_{hw} was adopted in this study, yielding

$$\widetilde{S}_{n-1,n} = V_n t_{hw} \tag{14}$$

where $t_{hw} = 1.2s$ in this study, which is consistent with the final time headway distribution generated by the DDPG algorithm.

Safety and comfort were considered by minimizing the absolute values of relative speed and jerk, respectively. The constraints include the following distance should be larger than 0 to avoid collisions; the velocity of the following vehicle should not be less than 0; and the acceleration of the following vehicle should be within a certain range.

5.9.3. MPC formulation

A constrained linear-quadratic MPC model was used, which solves at each time step the following finite-horizon optimal control problem

$$\min_{\boldsymbol{a}} \quad \sum_{t=0}^{N-1} \left[\left(\frac{S_{n-1,n}(t) - \widetilde{S}_{n-1,n}(t)}{S_{max}} \right)^2 + \left(\frac{\Delta V_{n-1,n}}{\Delta V_{max}} \right)^2 + \left(\frac{jerk(t)}{jerk_{max}} \right)^2 \right] \tag{15}$$

$$\text{s. t.} \quad \boldsymbol{x}(t+1) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) \tag{16}$$

$$S_{n-1,n} > 0 \tag{17}$$

$$V_n \geqslant 0 \tag{18}$$

$$-3m/s^2 \leqslant a_n \leqslant 3m/s^2 \tag{19}$$

where N is the prediction horizon (N = 10 in this study), S_{max}, ΔV_{max}, and $jerk_{max}$ are constants to normalize different kinds of tracking errors ($S_{max} = 15m$, $\Delta V_{max} = 8$ m/s, and $jerk_{max} = 60$ m/s^3 in this study), and $\boldsymbol{a} = [a(0), a(1), ..., a(N-1)]$ is the action sequence to be solved. After obtaining the optimal action sequence, only the first action $a(0)$ will be applied, and the process repeats at the next time step.

6. Results

In this section, car-following behavior observed in the empirical NGSIM data and that simulated by the DDPG and the MPC-based ACC models are compared, to demonstrate the model's ability to follow a leading vehicle safely, efficiently, and comfortably. All the analyses are based on testing data, and no collisions were observed for the DDPG model. The DDPG model produces the following vehicle trajectories by taking the leading vehicle trajectories as input.

6.1. Safe driving

Driving safety is evaluated based on TTC during the car-following events. Fig. 4 shows the cumulative distributions of TTC for NGSIM empirical human data, DDPG simulation, and MPC-based ACC simulation. For better interpretation, only TTC values in the

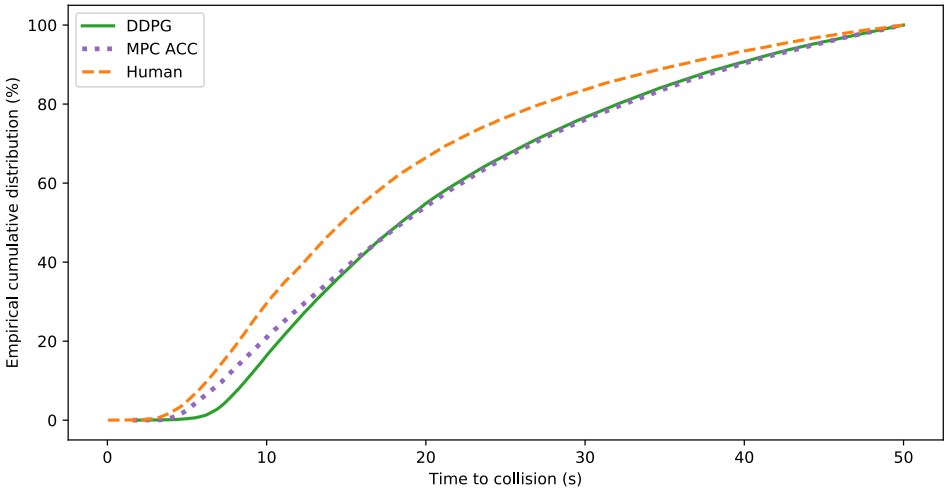

Fig. 4. Empirical cumulative distribution of TTC during car following.

range of 0 to 50s were presented. As can be seen, the DDPG model has larger TTC values than those of human drivers and MPC-based ACC algorithm. This means that car-following behavior generated by the DDPG model is safer than drivers' behavior observed in the NGSIM data and MPC-based ACC algorithm. Also, no collisions were observed with the DDPG model in both the training and testing phases.

6.2. Efficient driving

Driving efficiency was evaluated based on time headway during the car-following process. Time headway was calculated at every time step of a car-following event, and the cumulative distributions of these time headways are shown in Fig. 5. The average time headway for the DDPG model, MPC-based ACC algorithm, and human drivers are 1.24s, 1.23s, and 1.61s, respectively. As can be seen, both the DDPG model and the MPC-based ACC algorithm produced car-following trajectories that maintained a time headway around 1.2s. While the NGSIM data had a much wider range of time headway distribution (0s to 6s). This included some dangerous headways that were less than 1s, and also some inefficient headways that were larger than 3s. Therefore, it can be concluded that both the DDPG model and the MPC-based ACC algorithm can follow the leading vehicle with an efficient and safe time headway.

6.3. Comfortable driving

Driving comfort was evaluated based on jerk values during the car following process. Similar to time headway, it was calculated for every time step of a car-following event. Fig. 6 presents the cumulative distributions of jerk values during the car following events. The The mean values of jerk for the DDPG model, MPC-based ACC algorithm, and human drivers are 0.63 m/s^3, 0.70 m/s^3, and 1.73

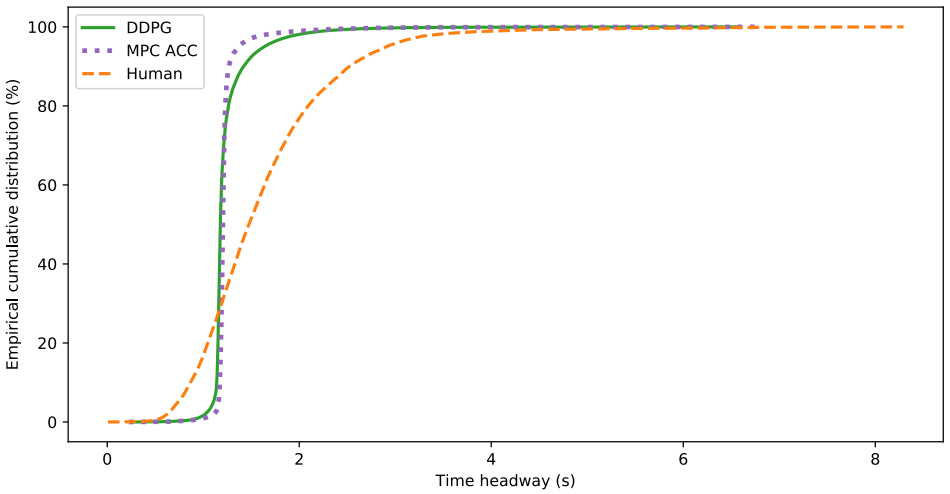

Fig. 5. Empirical cumulative distribution of time headway during car following.

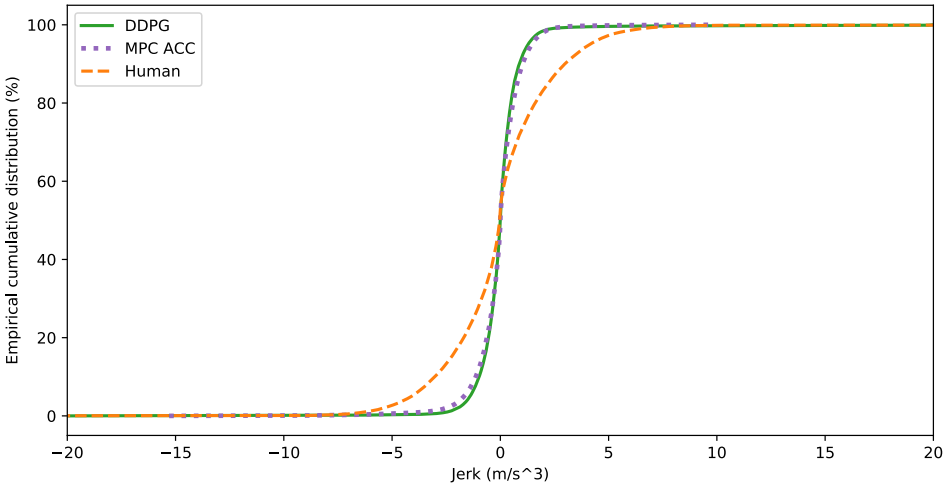

Fig. 6. Empirical cumulative distribution of jerk during car following.

m/s^3, respectively. It is obvious that the DDPG model and the MPC-based ACC algorithm produced trajectories with lower values of jerk. As smaller absolute values of jerk correspond to more comfortable driving, it can be concluded that the DDPG model can control vehicle velocity in a more comfortable way than human drivers in the NGSIM data and is slightly better than the ACC algorithm.

6.4. Demonstrations with sampled events

To give illustrations of the safe and comfortable driving of the DDPG model, two car-following events were randomly chosen from the NGSIM dataset. Fig. 7 and 8 show the observed speed, spacing, TTC, acceleration, and jerk, and the corresponding ones generated by the DDPG model and the MPC-based ACC algorithm. For better interpretation, TTC values were bounded in the range of -2 to 50s (i.e., TTC values larger than 50s were set as 50s, and those less than -2s were set as -2s). In some time intervals (e.g., 12 to 15s in Fig. 7), the driver in the NGSIM data drove in a way that produced very small TTC values, while the DDPG model keeps larger TTC values. Also, the driver in the NGSIM data drove in a way with frequent acceleration changes and large jerk values, while the DDPG model can remain a nearly constant acceleration and produced low jerk values.

To summarize, the DDPG model demonstrated the capability of safe, efficient, and comfortable driving in that it 1) has larger TTC values than those of human drivers and the MPC-based ACC algorithm, 2) could maintain efficient and safe headways within the range of 1s to 2s, and 3) followed the leading vehicle comfortably with smooth acceleration.

It should be noted that although the DDPG model performed only slightly better than the MPC-based ACC algorithm, it has the following advantages versus the MPC one:

- The proposed RL method has a faster running speed during the testing phase. Specifically, with the 403 car-following events in the testing data, the total running time of the DDPG model and the MPC-based ACC algorithm are 20.7s and 5305.9s, respectively. The average running time per car-following event of the DDPG model (0.05s) is much shorter than that of the MPC-based ACC algorithm (13.17s). The reason is that MPC needs to solve a constrained finite-time optimal control problem at every time step while RL only needs to take states as input through the actor-network and output actions.
- For the MPC-based ACC algorithm, sometimes the optimizer may not be able to find a feasible solution to the constrained finite-time optimal control problem. This will cause failed future vehicle acceleration generation. However, this is not the case for the proposed RL method, which generates future accelerations based on the actor-network, without the need for solving an optimization problem. Moreover, the proposed RL method is incorporated with a collision avoidance strategy for a safety check in case the actor-network generates wrong actions.
- The proposed RL method has fewer TTC values that are in the range of 0 to 10s than the MPC one, which means it is safer than the MPC-based ACC algorithm.

7. Discussion and conclusion

To sum up, this study uses RL to learn how to control vehicle velocity during car following in a safe, efficient, and comfortable way. Human driving data of the real world, from the NGSIM study, were used to train the model. The model was compared with an MPC-based ACC algorithm and the empirical NGSIM data, to evaluate the model's performance. Results show that the proposed model demonstrated the capability of safe, efficient, and comfortable driving, and significantly outperformed human drivers. Compared with the MPC-based ACC algorithm, the proposed model has better performance in terms of safety, comfort, and especially running speed during testing. The results indicate that reinforcement learning methods could contribute to the development of autonomous driving systems.

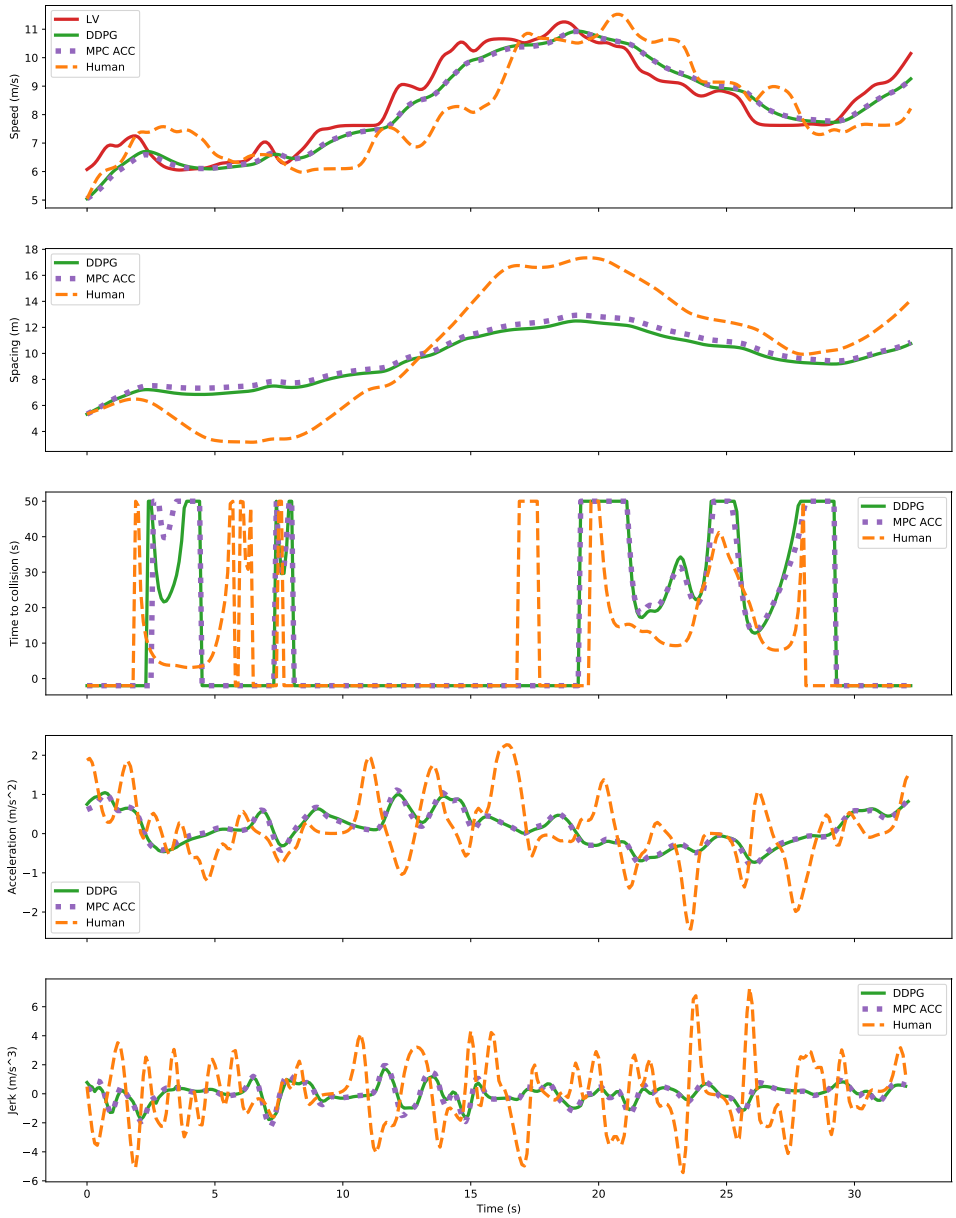

Fig. 7. #1 sampled car-following trajectories for comparing the performance of NGSIM human drivers, DDPG model, and the MPC-based ACC algorithm.

The proposed model can be further extended in the following aspects:

1. More objectives can be added, such as energy-saving driving.
2. The weights of the objectives can be adjusted to reflect users' individual preferences.
3. More complicated reward function forms can be adopted to express more complex reward mechanisms, such as a non-linear function.

This study can further be improved by designing better experience replay mechanisms. Experience replay lets RL agents remember and reuse experiences from the past. In the currently adopted DDPG algorithm, experience transitions were uniformly sampled, without considering their significance (Schaul et al., 2015). In future work, prioritizing experience can be utilized to replay important transitions more frequently, and therefore learn in a more efficient way.

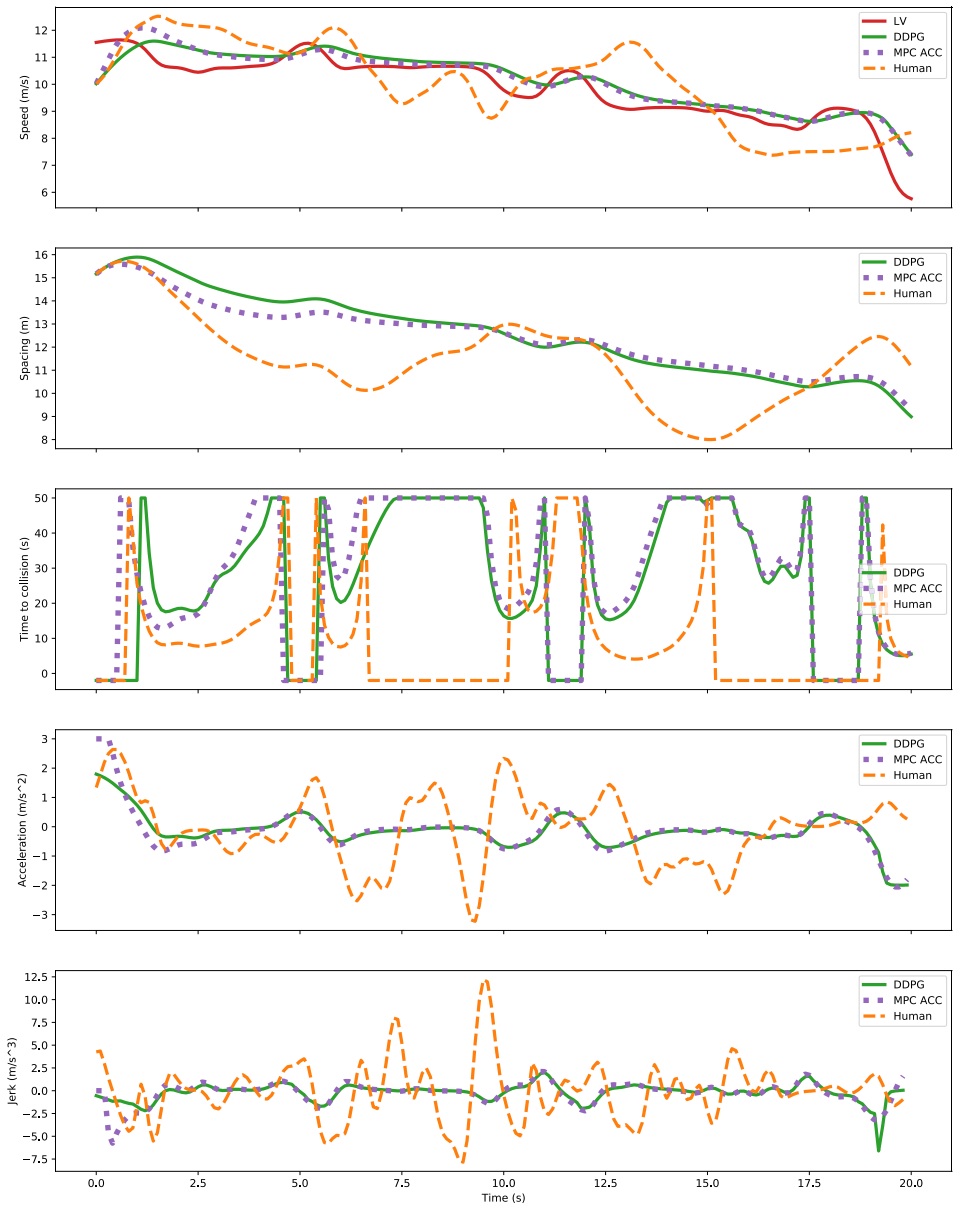

Fig. 8. #2 sampled car-following trajectories for comparing the performance of NGSIM human drivers, DDPG model, and the MPC-based ACC algorithm.

CRediT authorship contribution statement

Meixin Zhu: Methodology, Writing - original draft, Software. **Yinhai Wang:** Conceptualization, Supervision, Writing - review & editing. **Ziyuan Pu:** Writing - review & editing, Data curation, Investigation. **Jingyun Hu:** Writing - review & editing, Investigation, Visualization. **Xuesong Wang:** Conceptualization, Supervision, Writing - review & editing. **Ruimin Ke:** Writing - review & editing, Investigation.

Acknowledgements

This study was partly sponsored by the Chinese National Science Foundation (51878498).

References

Bando, M., Hasebe, K., Nakayama, A., Shibata, A., Sugiyama, Y., 1995. Dynamical model of traffic congestion and numerical simulation. Phys. Rev. E 51 (2), 1035.

Basu, C., Yang, Q., Hungerman, D., Singhal, M., Dragan, A.D., 2017. "Do you want your autonomous car to drive like you?" in. In: Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction. ACM, pp. 417–425.
Brackstone, M., McDonald, M., 1999. Car-following: a historical review. Transport. Res. Part F: Traffic Psychol. Behav. 2 (4), 181–196.
Camacho, E.F., Alba, C.B., 2013. Model Predictive Control. Springer Science & Business Media.
Chai, C., Wong, Y.D., 2015. Fuzzy cellular automata model for signalized intersections. Comput.-Aid. Civil Infrastruct. Eng. 30 (12), 951–964.
Cheng, R., Orosz, G., Murray, R.M., Burdick, J.W., 2019. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 3387–3395.
Cheng, J., Yuan, G., Zhou, M., Gao, S., Liu, C., Duan, H., Zeng, Q., 2020. Accessibility analysis and modeling for iov in an urban scene. IEEE Trans. Veh. Technol.
Ernst, D., Glavic, M., Capitanescu, F., Wehenkel, L., 2008. Reinforcement learning versus model predictive control: a comparison on a power system problem. IEEE Trans. Syst., Man, Cybernet., Part B (Cybernetics) 39 (2), 517–529.
Gazis, D.C., Herman, R., Rothery, R.W., 1961. Nonlinear follow-the-leader models of traffic flow. Oper. Res. 9 (4), 545–567.
Gipps, P.G., 1981. A behavioural car-following model for computer simulation. Transport. Res. Part B: Methodol. 15 (2), 105–111.
Grondman, I., Busoniu, L., Lopes, G.A., Babuska, R., 2012. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. IEEE Trans. Syst., Man, Cybernet. , Part C (Appl. Rev.) 42 (6), 1291–1307.
Helly, W., 1959. Simulation of bottlenecks in single-lane traffic flow.
Huang, T.-W., Cai, J., Yang, H., Hsu, H.-M., Hwang, J.-N., 2019. Multi-view vehicle re-identification using temporal attention model and metadata re-ranking. In: Proc. CVPR Workshops, pp. 434–442.
Jacobson, I.D., Richards, L.G., Kuhlthau, A.R., 1980. Models of human comfort in vehicle environments. Human Factors in Transport Research Edited by D.J. Oborne, J.A. Levis, 2.
Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. Adv. Neural Informat. Process. Syst. 1097–1105.
Kuefler, A., Morton, J., Wheeler, T., Kochenderfer, M., 2017. Imitating Driver Behavior with Generative Adversarial Networks, arXiv preprint arXiv:1701.06699.
Lefevre, S., Carvalho, A., Borrelli, F., 2015. Autonomous car following: A learning-based approach. In: 2015 IEEE Intelligent Vehicles Symposium (IV). IEEE, pp. 920–926.
Li, Y., 2017. Deep reinforcement learning: An overview, arXiv preprint arXiv:1701.07274.
Li, S., Li, K., Rajamani, R., Wang, J., 2010. Model predictive multi-objective vehicular adaptive cruise control. IEEE Trans. Control Syst. Technol. 19 (3), 556–566.
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, Y., McPhee, J., Azad, N.L., 2019. Comparison of deep reinforcement learning and model predictive control for adaptive cruise control, arXiv preprint arXiv:1910.12047.
Liu, K., Li, N., Rizzo, D., Garone, E., Kolmanovsky, I., Girard, A., 2019. Model-free learning to avoid constraint violations: An explicit reference governor approach. In: 2019 American Control Conference (ACC). IEEE, pp. 934–940.
Luo, L.-H., Liu, H., Li, P., Wang, H., 2010. Model predictive control for adaptive cruise control with multi-objectives: comfort, fuel-economy, safety and car-following. J. Zhejiang Univ. Sci. A 11 (3), 191–201.
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 (7540), 529.
Montanino, M., Punzo, V., 2015. Trajectory data reconstruction and simulation-based validation against macroscopic traffic patterns. Transport. Res. Part B: Methodol. 80 (1), 82–106.
Nageshrao, S., Tseng, H.E., Filev, D., 2019. Autonomous highway driving using deep reinforcement learning. In: 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC). IEEE, pp. 2326–2331.
Pipes, L.A., 1953. An operational analysis of traffic dynamics. J. Appl. Phys. 24 (3), 274–281.
Pu, Z., Li, Z., Jiang, Y., Wang, Y., 2020. Full bayesian before-after analysis of safety effects of variable speed limit system. IEEE Trans. Intell. Transp. Syst.
Saifuzzaman, M., Zheng, Z., 2014. Incorporating human-factors in car-following models: a review of recent developments and research needs. Transport. Res. Part C: Emerg. Technol. 48, 379–403.
Schaul, T., Quan, J., Antonoglou, I., Silver, D., 2015. Prioritized experience replay, arXiv preprint arXiv:1511.05952.
Sutton, R.S., Barto, A.G., 1998. Reinforcement Learning: An introduction. MIT Press, Cambridge.
Takahama, T., Akasaka, D., 2018. Model predictive control approach to design practical adaptive cruise control for traffic jam. Int. J. Automotive Eng. 9 (3), 99–104.
Treiber, M., Hennecke, A., Helbing, D., 2000. Congested traffic states in empirical observations and microscopic simulations. Phys. Rev. E 62 (2), 1805.
Treiber, M., Hennecke, A., Helbing, D., 2000. Congested traffic states in empirical observations and microscopic simulations. Phys. Rev. E 62 (2), 1805.
U.D. of Transportation, 2009. Ngsim–next generation simulation.
Uhlenbeck, G.E., Ornstein, L.S., 1930. On the theory of the brownian motion. Phys. Rev. 36 (5), 823.
Vogel, K., 2003. A Comparison of Headway and Time to Collision as Safety Indicators. Accid. Anal. Prev. 35 (3), 427–433.
Wang, J., Zhang, L., Zhang, D., Li, K., 2013. An adaptive longitudinal driving assistance system based on driver characteristics. IEEE Trans. Intell. Transp. Syst. 14, 1.
Wang, X., Zhu, M., Chen, M., Tremont, P., 2016a. Drivers' rear end collision avoidance behaviors under different levels of situational urgency. Transport. Res. Part C: Emerg. Technol. 71, 419–433.
Wang, X., Chen, M., Zhu, M., Tremont, P., 2016b. Development of a kinematic-based forward collision warning algorithm using an advanced driving simulator. IEEE Trans. Intell. Transp. Syst. 17 (9), 2583–2591.
Wang, X., Jiang, R., Li, L., Lin, Y., Zheng, X., Wang, F.-Y., 2018. Capturing car-following behaviors by deep learning. IEEE Trans. Intell. Transp. Syst. 19 (3), 910–920.
Wei, J., Dolan, J.M., Litkouhi, B., 2010. A learning-based autonomous driver: Emulating a human driver's intelligence in low-speed car following. In: Proceedings of the SPIE Defense, Security, and Sensing Conference, Unmanned Systems Technology, vol. 7693.
Wiedemann, R., 1974. Simulation des strabenverkehrsflusses in schriftenreihe des tnstituts fir verkehrswesen der universitiit karlsruhe.
Wilson, T.B., Butler, W., McGehee, D.V., Dingus, T.A., 1997. Forward-looking collision warning system performance guidelines. SAE Trans. 701–725.
Zhang, G., Wang, Y., Wei, H., Chen, Y., 2007. Examining headway distribution models with urban freeway loop event data. SAGE Publications Sage CA: Los Angeles, CA, 1999(1).
Zhang, J., Wang, F.-Y., Wang, K., Lin, W.-H., Xu, X., Chen, C., et al., 2011. Data-driven intelligent transportation systems: A survey. IEEE Trans. Intell. Transp. Syst. 12 (4), 1624–1639.
Zhu, M., Wang, X., Tarko, A., et al., 2018a. Modeling car-following behavior on urban expressways in shanghai: A naturalistic driving study. Transport. Res. Part C: Emerg. Technol. 93, 425–445.
Zhu, M., Wang, X., Wang, Y., 2018b. Human-like autonomous car-following model with deep reinforcement learning. Transport. Res. Part C: Emerg. Technol. 97, 348–368.
Zhu, M., Wang, X., Hu, J., 2020. Impact on car following behavior of a forward collision warning system with headway monitoring. Transport. Res. Part C: Emerg. Technol. 111, 226–244.