Reinforcement Learning Based Velocity Control for Autonomous Driving with Multi-Objectives: Safety, Efficiency, and Comfort

Rune Yu
Graduate Student
School of Transportation Engineering
Tongji University
4800 Cao'an Road, Jiading District
Shanghai, 201804, China
Phone: +86-15021901395
Email: yurune1111@outlook.com
(Corresponding author)

Hangfei Lin, PhD
Professor
School of Transportation Engineering
Tongji University
4800 Cao'an Road, Jiading District
Shanghai, 201804, China
Email: linhangfei@126.com

Meixin Zhu
Graduate Student
School of Transportation Engineering
Tongji University
4800 Cao'an Road, Jiading District
Shanghai, 201804, China
Email: 1151208@tongji.edu.cn

Word Count: 4,936
Tables and Figures (10 at 250 words each): 2,500
Total Count: 7,436

July 2017
Prepared for Presentation at the TRB Annual Meeting

ABSTRACT

A model used for velocity control during car following was proposed based on deep reinforcement learning (RL). To fulfil the multi-objectives of car following, a reward function reflecting driving safety, efficiency, and comfort was constructed. With the 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. A total of 1,341 car-following events extracted from the public Next Generation Simulation (NGSIM) dataset were used to train the model. And car-following behavior produced by the DDPG model were compared with that observed in the empirical NGSIM data, to demonstrate the model's ability to follow a lead vehicle safely, efficiently, and comfortably.

Results show that the model demonstrates the capability of safe, efficient, and comfortable velocity control in that it 1) has small percentages (8%) of dangerous minimum time to collision values (< 5s) than human drivers in the NGSIM data (35%); 2) can maintain efficient and safe headways in the range of 1s to 2s; and 3) can follow the lead vehicle comfortably with smooth acceleration. The results indicate that reinforcement learning methods could contribute to the development of autonomous driving systems.

Keywords: Car Following, Velocity Control, Reinforcement Learning, NGSIM, and Autonomous Driving

1 INTRODUCTION

Car following is the most frequent driving scenario. The main task of car following is controlling vehicle velocity to maintain a safe and comfortable following gap. Autonomous car-following velocity control has the promise to mitigate drivers' workload, to reduce traffic accidents, and to increase traffic flow (*1*).

A driver model is critical for velocity control systems. In general, driver models related to car following have been established with two approaches: rule-based and supervised learning (*2*). Rule-based approach mainly refers to traditional car-following models, such as the Gaxis-Herman-Rothery model (*3*) and the intelligent driver model (*4*). Supervised learning approach relies on data typically provided through human demonstration in order to approximate the relationship between car-following state and acceleration.

These two approaches all intend to emulate human drivers' car-following behavior. However, imitating human driving behaviors may not be the best solution in autonomous driving. Firstly, human drivers may not drive in an optimal way. Secondly, users may not want their autonomous vehicles driving in a way like them (*5*). Thirdly, driving should be optimized with respect to safety, efficiency, comfort, and so on, rather than imitating human drivers.

To resolve the problem, we propose a deep reinforcement learning (RL) based on car-following model for autonomous velocity control. This model does not try to emulate human drivers, rather, it directly optimizes driving safety, efficiency, and comfort, by learning from trial and interaction with a simulation environment.

Specifically, the deep deterministic policy gradient (DDPG) algorithm (*6*) that performs well in continuous control field was utilized to learn an actor network together with a critic network. The actor is responsible for policy generation: outputting following vehicle accelerations based on speed, relative speed and spacing. The critic is responsible for policy improvement: update the actor's policy parameters in the direction of performance improvement.

To evaluate the proposed model, real-world driving data collected in the NGSIM project (*7*) were used to train the model. And car-following behavior simulated by the DDPG model were compared with that observed in the empirical NGSIM data, to demonstrate the model's ability to follow a leading vehicle safely, efficiently, and comfortably.

2 BACKGROUND

2.1 Reinforcement Learning

Reinforcement learning usually solves sequential decision making problems. An RL agent interacts with an environment over time. At each time step t, the agent receives a state s_t and selects an action a_t from some action space A, following a policy $\pi(a_t|s_t)$, which is the agent's behavior, i.e., a mapping from state s_t to actions a_t, receives a scalar reward r_t, and transitions to the next state s_{t+1}, according to the environment dynamics, or model. In an episodic problem, this process continues until the agent reaches a terminal state and then it restarts. The return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$

is the discounted, accumulated reward with the discount factor $\gamma \in (0,1]$. The agent aims to maximize the expectation of such long term return from each state (*8*). In general, there are two types of RL methods: value-based and policy-based (*9*).

2.1.1 Value-Based Reinforcement Learning

A value function is a prediction of the expected, accumulative, discounted, and future reward, measuring how good each state, or state-action pair is. The action value $Q^{\pi}(s,a) = E[R_t \mid s_t = s, a_t = a]$ is the expected return for selecting action *a* in state *s* and then following policy π. Tt represents the "quality" of a certain action *a* in a given state *s*. The task of value-based RL methods is to learn the action value function from experience. One example of such an algorithm is Q -learning. Starting from a random Q -function, the agent continuously updates its Q -values by playing the game and obtaining rewards. The iterative updates are derived from the Bellman equation (*9*):

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

This is based on the following intuition: maximum future reward for this state *s* and action *a* is the immediate reward *r* plus maximum future reward for the next state s'. Using these evolving *Q*-values, the agent chooses the action with the highest $Q(s,a)$ to maximize its expected future rewards.

2.1.2 Policy-Based Reinforcement Learning

In contrast to value-based methods, policy-based methods optimize the policy $\pi(a \mid s;\theta)$ (with function approximation) directly, and update the parameters θ by gradient ascent on $E[R_t]$. REINFORCE is a policy gradient method, updating θ in the direction of $\nabla_\theta \log \pi(a_t \mid s_t;\theta) R_t$ (*8*).

In order to update the gradients with lower variance and speed up learning process, an actor-critic method is usually used. In an actor-critic algorithm, the learning agent is split into two separate entities: the actor (policy) and the critic (value function). The actor is only responsible for generating an action *a*, given the current state *s*. The critic is responsible for evaluating the current policy prescribed by the actor. The critic approximates and updates the value function through samples. The value function is then used to update the actor's policy parameters in the direction of performance improvement (*10*).

2.2 Deep Reinforcement Learning

Deep reinforcement learning refers to reinforcement learning algorithms that use neural networks to approximate any of the following component of reinforcement learning: value function, $V(s;\theta)$ or, policy $\pi(a \mid s;\theta)$, and model (state transition and reward). Here, the parameters θ are the

weights in deep neural networks. This field seeks to combine the advances in deep neural networks with reinforcement learning algorithms to create agents capable of acting intelligently in complex environments.

2.2.1 Deep Q-Network

Deep RL in discrete action spaces can be performed using the Deep *Q*-Learning method introduced by Mnih et al. (*11*) which employs a single deep network to estimate the value function of each discrete action and, when acting, selects the maximally valued output for a given state input. Deep *Q* networks (*DQN*) work well when there are only a few possible actions but do not function in continuous action spaces, like in our case. Therefore, Lillicrap et al. (*6*) proposed an actor-critic algorithm, deep deterministic policy gradient (DDPG), which builds on *DQN* but can be applied for continuous control problems.

2.2.2 Deep Deterministic Policy Gradient

Like *DQN*, DDPG uses deep neural network function approximators. However, unlike *DQN*, DDPG uses two separate actor and critic networks (*6*). The critic network with weights θ^Q approximates the action-value function $Q(s,a|\theta^Q)$. The actor network with weights θ^μ explicitly represents the agent's current greedy policy $\mu(s|\theta^\mu)$ of the *Q*-function, which maps from a state of the environment *s* to an action *a*. To make the learning stable and robust, similar to *DQN*, DDPG deploys the experience replay and an idea similar to target network：

(1) Experience replay

One challenge when using neural networks for reinforcement learning is that most optimization algorithms assume that the samples are independently and identically distributed. Obviously, when the samples are generated from exploring sequentially in an environment this assumption no longer holds. A replay buffer was applied to address these issues.

The replay buffer is a finite sized cache *D*. Transitions were sampled from the environment according to the exploration policy and the tuple (s_t, a_t, r_t, s_{t+1}) was stored in the replay buffer. When the replay buffer was full the oldest samples were discarded. At each time step the actor and critic are updated by sampling a minibatch uniformly from the buffer. Because DDPG is an off-policy algorithm, the replay buffer can be large, allowing the algorithm to benefit from learning across a set of uncorrelated transitions.

(2) Target network

Directly implementing *Q* learning with neural networks proved to be unstable in many environments. Since the network $Q(s,a|\theta^Q)$ being updated is also used in calculating the target value, the *Q* update is prone to divergence. A solution to this problem is to use a separate network (target network) for calculating the target values. In DDPG algorithm, two target networks, $Q'(s,a|\theta^{Q'})$ and $\mu'(s|\theta^{\mu'})$, were created for the main critic and actor networks respectively. They

are identically in shape to the main networks but have different network weights θ'. The weights of these target networks are updated by having them slowly track the learned networks: $\theta' = \tau\theta + (1-\tau)\theta'$ with $\tau \ll 1$. This means that the target values are constrained to change slowly, greatly improving the stability of learning.

The full DDPG algorithm is listed in Algorithm 1. DDPG starts by initializing the replay buffer as well as its actor, critic and corresponding target networks. For each episode step, an action is then selected according to the exploratory policy. Afterwards, the reward rt and new state st+1 are observed and the information is stored in the replay memory *D*.

During training, minibatches are sampled from replay memory. The critic is then updated with the same loss function as in *DQN*. In a next step 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 slowly moved in the direction of the updated weights of the actor and critic networks.

Algorithm 1 Deep deterministic policy gradient (*6*)

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.

Initialize target network $Q'(s,a|\theta^{Q'})$ and $\mu'(s|\theta^{\mu'})$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$

Initialize replay buffer *D*

for episode = 1 to M **do**

Initialize a random process N for action exploration

Receive initial observation state s_1

for t = 1 to T **do**

Select action $a_t = \mu(s_t|\theta^\mu) + \mathbb{N}_t$ according to the current policy and exploration noise $\mathbb{N}_t$.

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 replay buffer *D*

Sample a random minibatch of *N* transitions (s_i, a_i, r_i, s_{i+1}) from *D*

Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

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:

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

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

end for

end for

3 DATA PREPARATION

Data in the Next Generation Simulation (NGSIM) project (*7*) were used. NGSIM provides high-quality trajectory data for traffic research. The data used for the paper is retrieved from the southbound direction of U.S. Highway 101 (Hollywood Freeway) in Los Angeles, CA, USA. The sampling period ranges from 4:00 p.m. to 4:15 p.m. on April 13, 2005. As is shown in Figure 1, the study area is approximately 630 meters and consists of 6 lanes. To eliminate the unnecessary error, the reconstructed NGSIM I-80 data (*12*) were used.

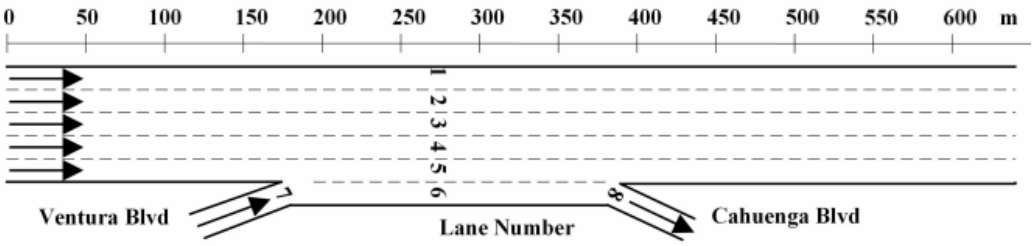

FIGURE 1 The layout of the road segment studied in NGSIM U.S. Highway 101 dataset.

Car-following events were automatically extracted for this analysis from the massive volume of data by applying a car-following filter. A car-following event was extracted if the following criteria were met simultaneously:

- The leading and following vehicle pairs stay in the same lane;
- Length of car-following event>15s: this criterion guaranteed that the car-following persisted long enough to reach a stable state.

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

4 FEATURES FOR REWARD FUNCTION

In this section, features that capture relevant objectives of 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. Time to collision (TTC) was used to emphasize safety. As a widely used safety indicator, TTC indicates the time span left before two vehicles collide, if nobody takes evasive action, that is computed as:

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

where $S_{n-1,n}(t)$ is the following gap, $\Delta V_{n\text{-}1,n}(t)$ is the relative speed.

TTC in car-following situations is only defined when the speed of the following vehicle is

higher than the speed of the leading vehicle. TTC is an indicator for a traffic conflict and is, thus, inversely related to accident risk (smaller TTC values indicate higher accident risks and vice versa) (*13*).

To apply TTC as feature reflecting safety, a safety limit, (a lower bound of TTC) should be determined. However, different opinions can be found from the literature, as to which value should be used as safety limit—suggestions range from 1.5s in urban areas to 5s (*13*).

To address this problem, we determine the safety limit based on the NGSIM empirical data. Figure 2 shows the cumulative distribution of TTC values in car-following events extracted from the NGSIM data. A 7-second safety limit corresponding to the 10 percentiles of TTC distribution was chosen. Then the TTC feature was constructed as:

$$F_{TTC} = \log(TTC/7) \tag{3}$$

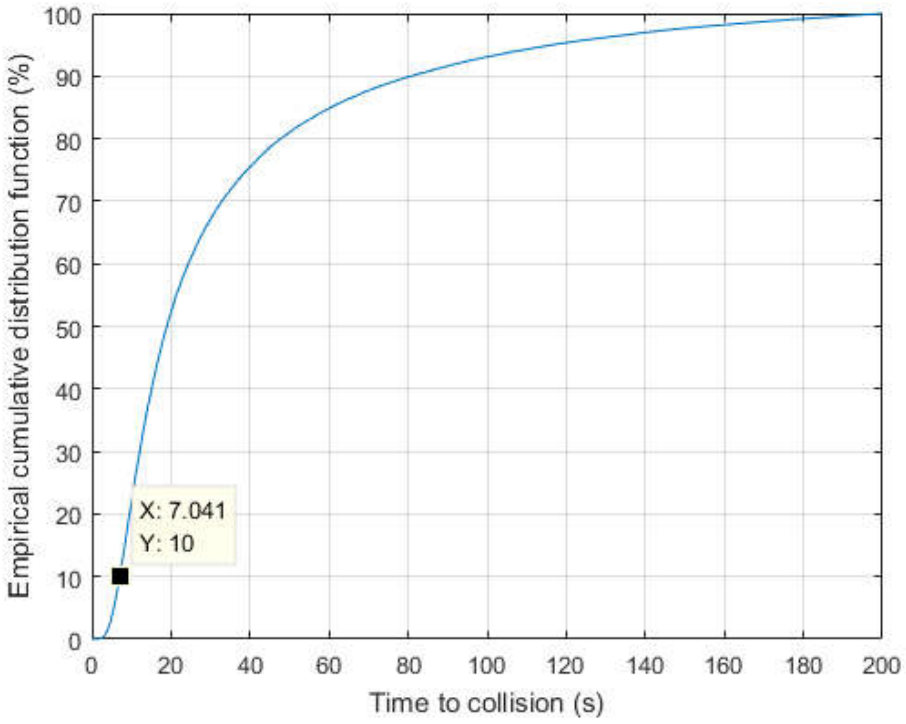

FIGURE 2 Cumulative distribution of TTC in car-following events extracted from the NGSIM data.

4.2 Efficiency

Time headway is defined as the elapsed 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 (*14*).

The rules of different countries are not quite the same, in regard of the legal or

recommended time headway. In the US, e.g. several driver training programs state that it is impossible to follow a vehicle safely with a headway of less than 2 s. In Germany, the recommended time headway is 1.8 s. Fines are imposed when the time headway is smaller than 0.9 s. In Sweden the National Road Administration recommends a time headway of 3 s in rural areas, and the police use a time headway of 1 s as orientation for imposing fines (*13*).

This study determined the appropriate time headway based on the empirical NGSIM data. Figure 3 presents the distribution of time headway in all of the extracted 1,341 car-following events.

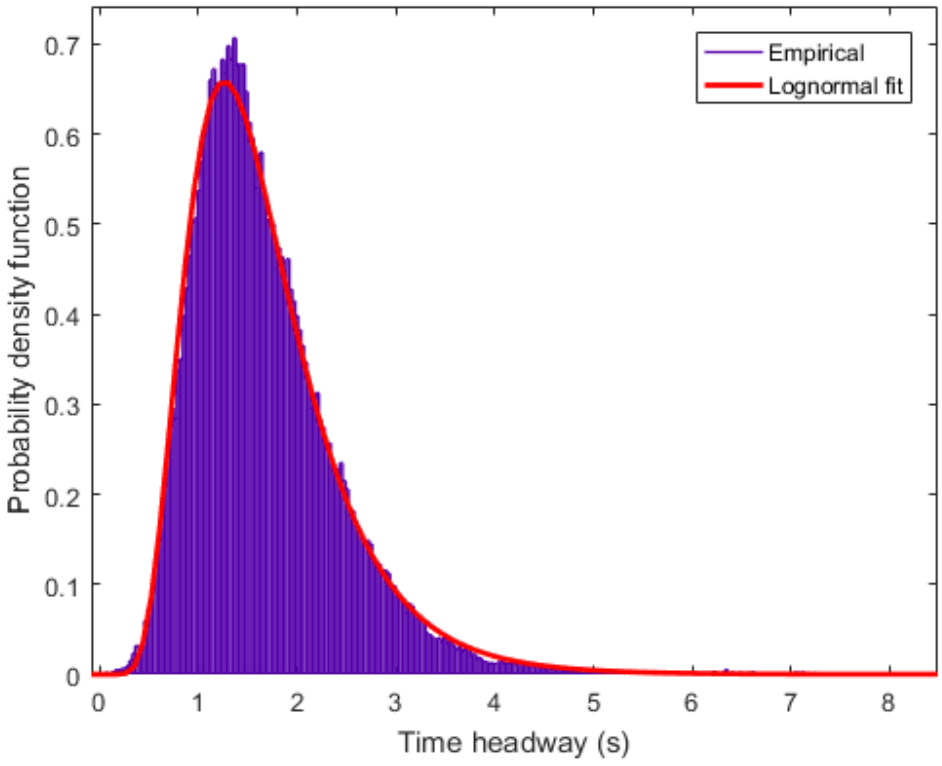

FIGURE 3 Distribution of time headway in car-following events extracted from the NGSIM data.

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:

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

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, 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:

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

According to this headway feature, headways around 1.3 seconds correspond to large headway feature values (about 0.65); while headways being too long or too short correspond to low feature values. In this way, efficient headways are encouraged while unsafe or too long headways are discouraged.

4.3 Comfort

Jerk, defined as the change rate of acceleration, was used to measure driving comfort because it has a strong influence on comfort of the passengers (*15*). A jerk feature was constructed as:

$$F_{jerk} = \frac{jerk^2}{3600}$$

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;
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 the acceleration of a vehicle is continuous, a deep RL method called deep deterministic policy gradient (DDPG) (*6*) that performs well in continuous action space was utilized to learn velocity control policy. In this section, the approach proposed to learn 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 $V_n(t)$, spacing $S_{n-1,n}(t)$, and relative speed $\Delta V_{n\text{-}1,n}(t)$. The action is the longitudinal acceleration of the FV $a_n(t)$. The acceleration of FV was confined between -3 m/s^2 and 3 m/s^2 based on the observed FV acceleration of all the car-following events. With state and action at time step *t*, a kinematic point-mass model was used for state updating:

$$V_n(t+1) = V_n(t) + a_n(t) \cdot \Delta T$$

$$\Delta V_{n\text{-}1,n}(t+1) = V_{n-1}(t+1) - V_n(t+1) \quad (5)$$

$$S_{n\text{-}1,n}(t+1) = S_{n-1,n}(t) + \Delta V_{n-1,n}(t+1) \cdot \Delta T$$

where ΔT is the update time interval, set as 0.1s in this study, and $V_{n\text{-}1}$ is the velocity of LV, which is externally inputted.

5.2 Simulation Setup

A simple car-following simulation environment was implemented to enable the RL agent to interact with the environment through a sequence of states, actions and rewards. The NGSIM data contains velocities of both the leading and the following vehicle. These data therefore allow for a direct comparison between the measured driver behavior and trajectories simulated by a RL agent with the leading vehicle serving as externally controlled input.

Initialized with the empirically given following vehicle speed, spacing and velocity differences, $V_n(t=0) = V_n^{rdata}(t=0)$ and $S_{n-1,n}(t=0) = S_{n-1,n}^{data}(t=0)$, the RL agent is used to compute the acceleration $a_n(t)$. The future states of the following car are then generated iteratively based on the state updating rules defined in section 5.1. The state is reset to the value in the empirical dataset when the simulating car-following event terminates at its maximum time step.

5.3 Reward function

In RL, the reward function, $r(s, a)$, is used as a training signal to encourage or discourage behaviors in the context of a desired task. The reward provides a scalar value reflecting the desirability of a particular state transition that is observed by performing action a starting in the initial state s and resulting in a successor state s'.

For the task of autonomous car following, a reward function was established based on a linear combination of the feature constructed in section 4:

$$r = -w_1 F_{TTC} + w_2 F_{headway} - w_3 F_{jerk} \quad (6)$$

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

5.4 Network Architecture

Two separate neural networks were used to represent the actor and critic respectively. At time step t, the actor network takes a state $s_t = (v_n(t), \Delta v_{n-1,n}(t), \Delta S_{n-1,n}(t))$ as input and outputs a continuous action: the FV acceleration $a_n(t)$; the critic network takes as input a state s_t and action a_t and

outputs a scalar Q-value $Q(s_t, a_t)$.

As shown in Figure 4, both the actor and critic networks have three layers: an input layer taking the input signals to the whole neural network, an output layer generating the output signal, and a hidden layer containing 30 neurons between the former two layers. We had tested even deeper neural networks, but experiment results showed that considering neural networks deeper than one hidden layers was unnecessary for our problem, which has only three or four input variables.

For the hidden layers, each neuron has a Rectified Linear Unit (RLU) activation function that transfers its input to its output signal. The RLU function computes as $f(x) = \max(0, x)$ and has been shown to greatly accelerate the convergence of network parameter optimization (*16*). The final output layer of the actor network used a tanh activation function, which maps a real-valued number to the range [-1, 1] and thus can be used to bound the output actions.

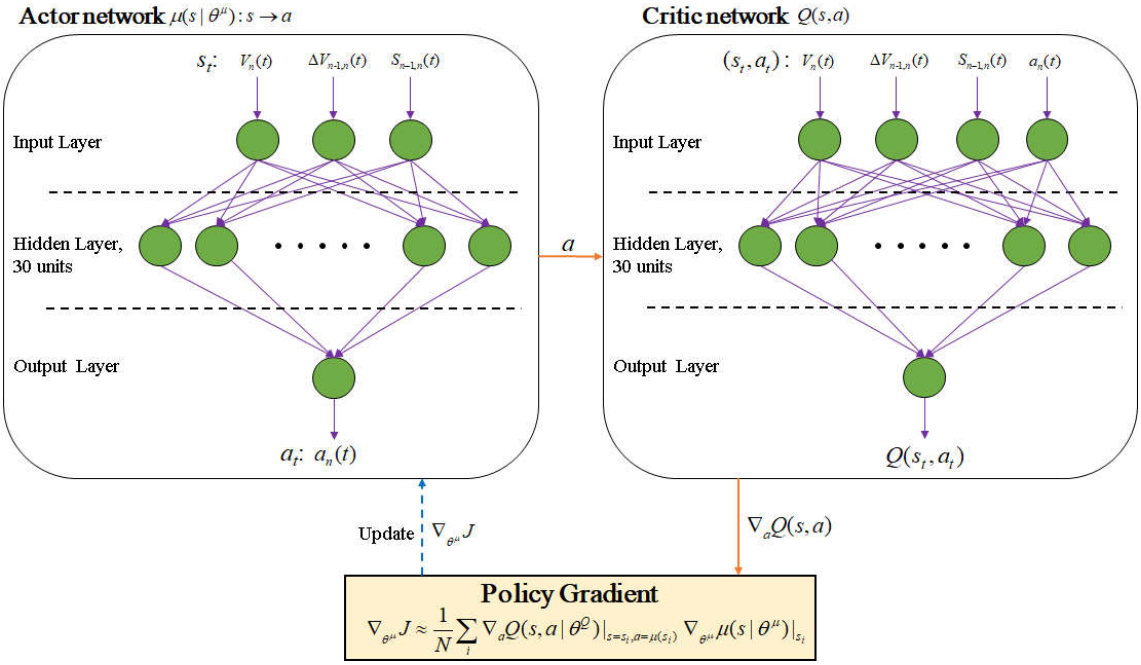

FIGURE 4 Architecture of the actor and critic networks.

5.5 Network Update and Hyper Parameters

At each learning step, the weight coefficients of the critic network were updated by stochastic gradient decent algorithm together with the adaptive learning rate trick Adam (*17*) to minimize the loss function $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$. As shown in Figure 4, the actor network parameters were updated as follows. On a forward pass, the actor's output is passed forward into the critic and evaluated. Next, the estimated Q-Value is back-propagated through the critic,

producing gradients $\nabla_a Q(s,a)$ that indicate how the action should change in order to increase the Q-Value. On the backwards pass, these gradients flow from the critic through the actor. An update is then performed only over the actor's parameters θ^{μ} according to the gradient $\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a \mid \theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s \mid \theta^{\mu})|_{s_i}$.

The hyperparameters (parameters set prior to the commencement of the learning process) adopted are shown in Table 1, the values of these hyperparameters were selected according to Lillicrap et al. (*6*) and also by performing an test on a randomly sampled training dataset.

TABLE 1 Hyperparameters and Corresponding Descriptions

Hyperparameter	Value	Description
Learning rate	0.001	The learning rate used by Adam
Discount factor	0.99	Discount factor gamma used in the Q-learning update
Minibatch size	32	Number of training cases over which each stochastic gradient descent (SGD) update is computed
Replay memory size	7000	Number of training cases in the replay memory
Soft target update t	0.001	The update rate of target networks

5.6 Exploration Noise of Action

A major challenge of learning in continuous action spaces is exploration. An exploration policy was constructed by adding noise sampled from a noise process to the original actor policy. An Ornstein-Uhlenbeck process (*18*) with $\theta = 0.15$ and $\sigma = 0.2$ were used as suggested by Lillicrap et al. (*6*). The Ornstein-Uhlenbeck process models the velocity of a Brownian particle with friction, which results in temporally correlated values centered around zero. The temporally correlated noise enables the agent to explore well in physical environment that have momentum.

5.7 Training DDPG Velocity Control Model

For the 1,341 extracted car-following events, 70% (938) of them were used as the training data, and the remaining 30% were used as testing data. For each episode of training, the 938 training car-following events were simulated sequentially by the RL agent. The state was initialized according to the empirical data whenever a new car-following event was to be simulated.

The training was repeated for 60 episodes, and the RL agent generated the smallest sum of training and testing errors was selected. Figure 5 shows the change of average reward with respect to training episode. As can be seen, the performance of the DDPG model starts to converge when training episode reaches 20. 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). It should be noted that the model has similar performance on test data as that on training data. This demonstrates that the model can generalize

well on new data.

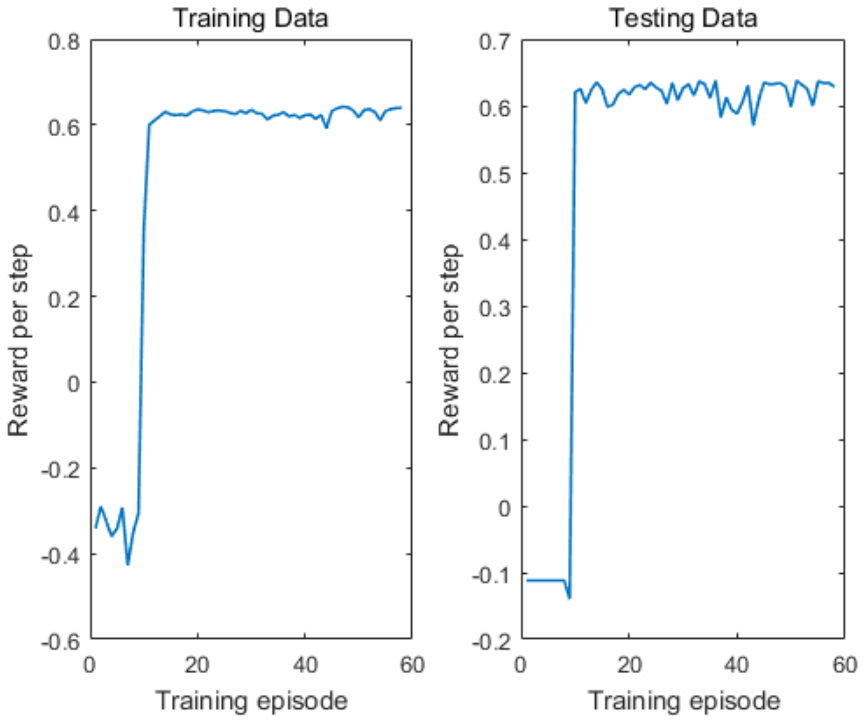

FIGURE 5 Reward curves during training.

6 RESULTS

In this section, two types of car-following behavior, observed in the empirical NGSIM data and that simulated by the DDPG model, were compared, to demonstrate the model's ability to follow a leading vehicle safely, efficiently, and comfortably. 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 minimum TTC during a car-following event. Figure 6 shows the cumulative distributions of minimum TTC for NGSIM empirical data and DDPG simulation. Nearly 35% of NGSIM minimum TTCs were lower than 5s, while only about 8% of DDPG minimum TTCs were lower than 5s. It means car-following behavior generated by DDPG model is much safer than drivers' behavior recorded in NGSIM data.

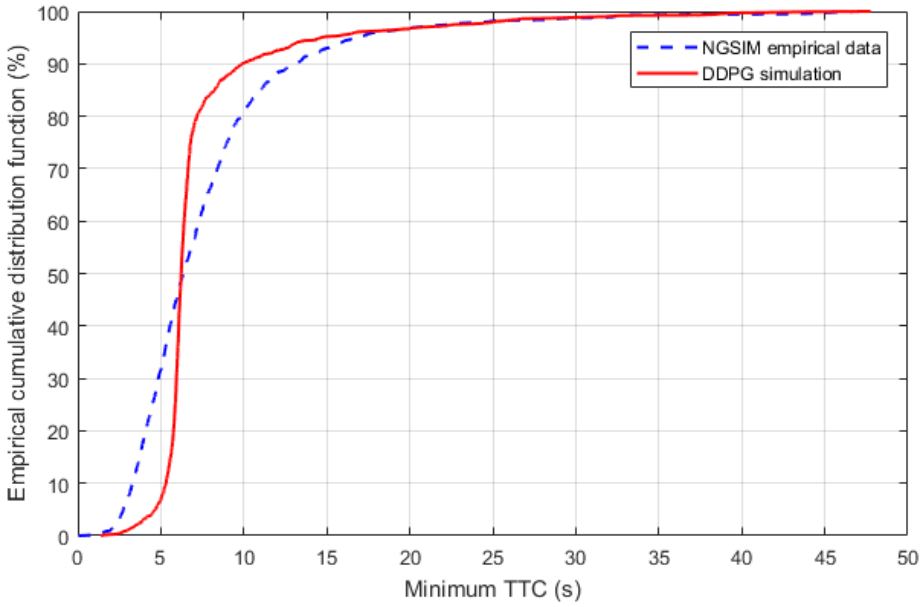

FIGURE 6 Cumulative distribution of minimum TTC during car following.

To give an illustration of the safe driving of the DDPG model, a car-following event was randomly chosen from the NGSIM dataset. Figure 7 shows the observed speed, spacing, and acceleration, and those replicated by the DDPG model. The driver in the NGSIM data drove in a way that produced very small inter-vehicle spacing, while the DDPG model keeps a safe following gap around 10m.

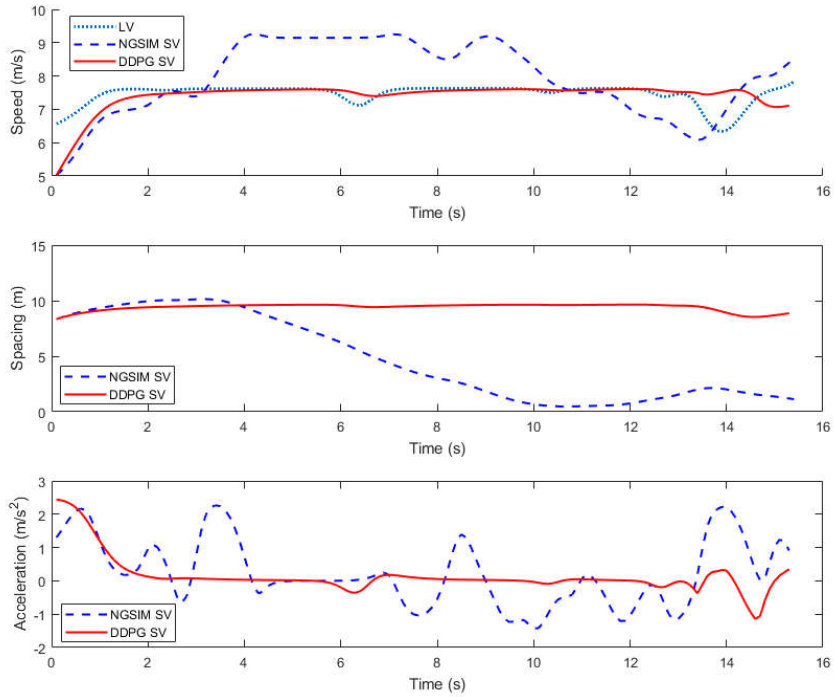

FIGURE 7 Comparison of driving safety between NGSIM data and the DDPG model.

6.2 Efficient Driving

Time headway during car-following process was used to evaluate driving efficiency. Time headway was calculated at every time step of a car-following event, and the distribution of these time headways is shown in Figure 8.

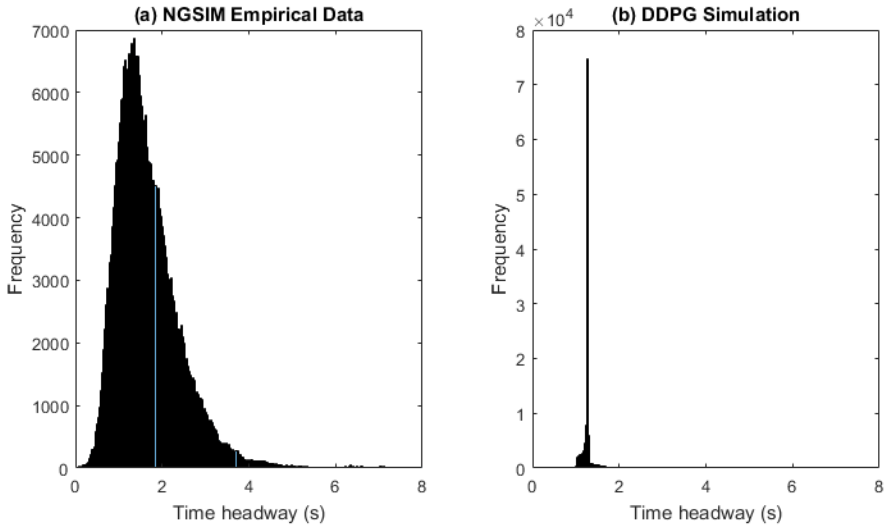

FIGURE 8 Histograms of time headway during car following for (a) NGSIM empirical data and (b) DDPG simulation.

As can been seen, the DDPG model produced car-following trajectories that always maintained a time headway in the range of 1s to 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, drawing to the conclusion that the DDPG model has the ability to follow the leading vehicle with an efficient and safe time headway.

6.3 Comfortable Driving

Driving comfort was evaluated based on jerk values during car following. Similar to time headway, it was calculated for every time step of a car-following event. Figure 9 presents the histograms of jerk values during car following.

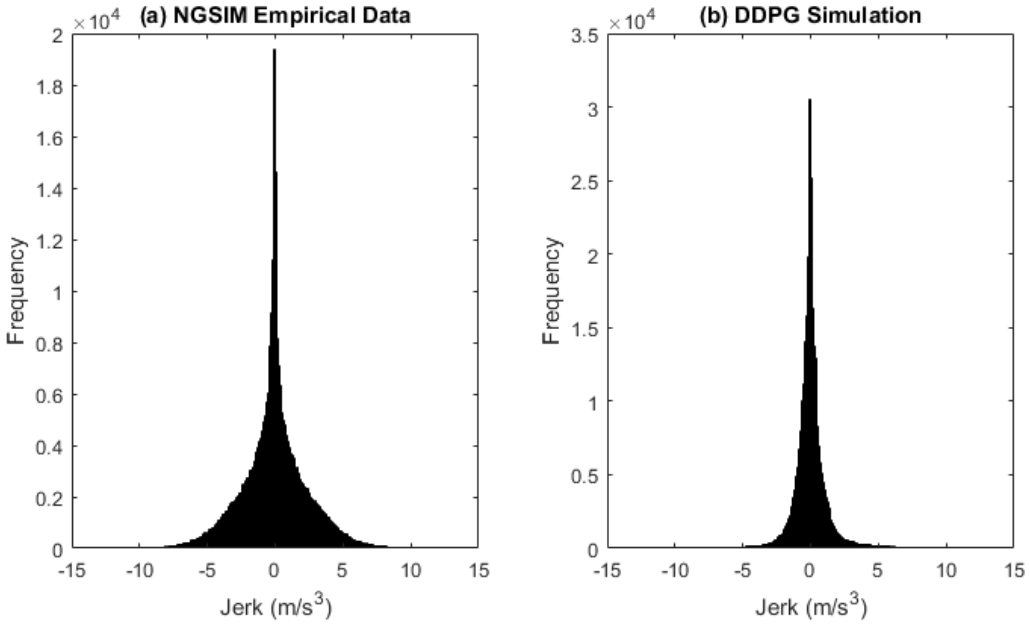

FIGURE 9 Histograms of jerk during car following for (a) NGSIM empirical data and (b) DDPG simulation.

It is obvious that the DDPG model produced trajectories with lower values of jerk. Firstly, the DDPG trajectories had a narrow jerk distribution range (–5 to 5 m/s^3) than NGSIM data (–5 to 5 m/s^3). Second, jerk values were centered more closely to zero in DDPG simulation trajectories than in NGSIM data. 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.

To give an illustration of the comfortable driving of the DDPG model, a car-following event was randomly chosen in the NGSIM dataset. Figure 10 shows the observed speed, spacing, acceleration, and jerk, and those elements generated by the DDPG model. 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.

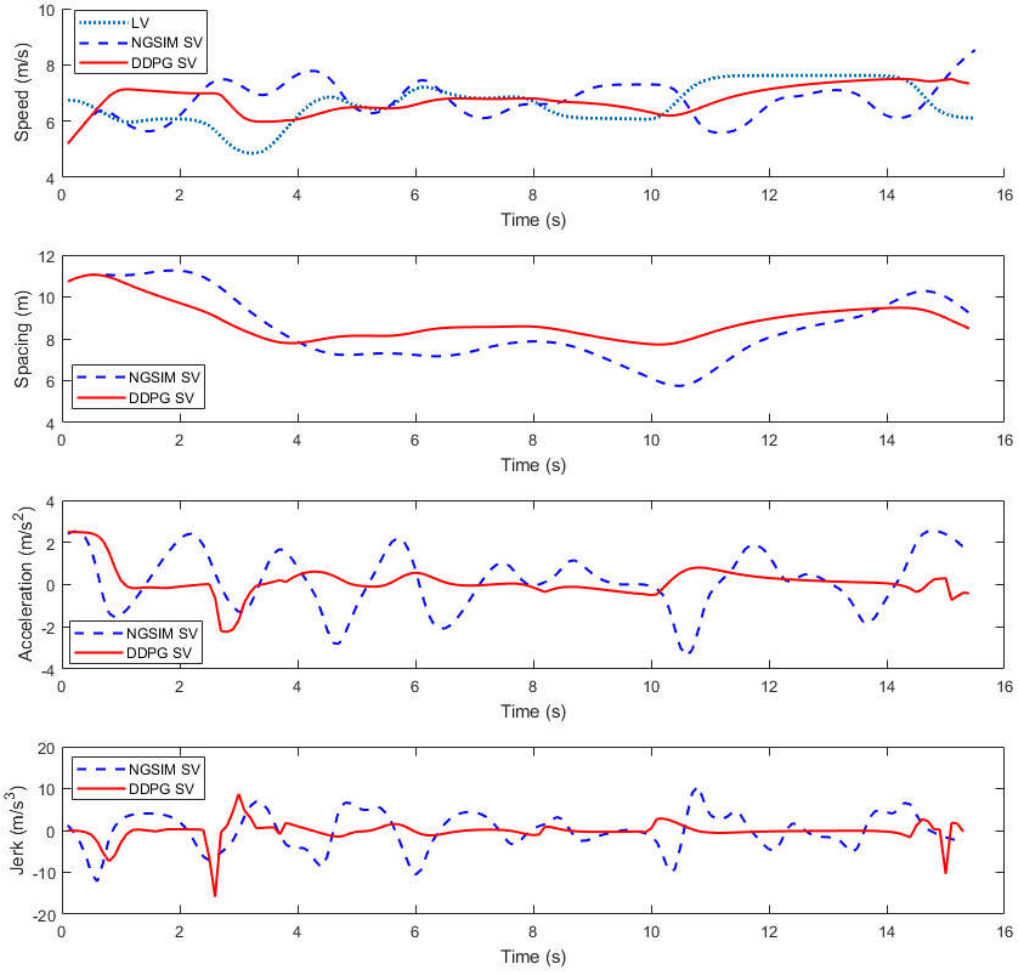

FIGURE 10 Comparison of driving comfort between NGSIM data and the DDPG model.

To summarize, the DDPG model demonstrated the capability of safe, efficient, and comfortable driving in that it 1) had small percentages of dangerous minimum TTC values that is less than 5 seconds; 2) could maintain efficient and safe headways within the range of 1s to 2s; and 3) followed the leading vehicle comfortably with smooth acceleration.

7 DISCUSSION AND CONCLUSION

The model used for velocity control during car-following was proposed based on deep RL. The model uses deep deterministic policy gradient (DDPG) algorithm to learn from trials and interaction, with a reward function signaling the RL agent performed in terms of driving safety, efficiency, and comfort. Results show that the proposed DDPG car-following model demonstrated a better capability of safe, efficient, and comfortable driving compared to human drivers in the real

world.

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) In the current, a linear function was used to combine different objectives (features). More complicated reward function formed can be adopted to express more complex reward mechanisms, such as a non-linear function.

Although the network structure utilized in our study with only one hidden layer cannot be regarded as a truly "deep" neural network, it fully exploits the main idea of deep reinforcement learning methods and can be easily extended, inputting provided variables.

Another potential improvement to our model is related to experience replay. Experience replay lets RL agents remember and reuse experiences from the past. In the currently adopted DDPG algorithm, experience transitions were uniformly sampled from a replay memory. However, this approach simply replays transitions at the same frequency that they were originally experienced, regardless of their significance (*19*). In future work, we can utilize a framework for prioritizing experience, so as to replay important transitions more frequently, and therefore learn in a more efficient way.

To sum up, this study uses deep reinforcement learning to learn how control vehicle velocity during car following in a safe, efficient, and comfortable way. Human driving data of real world, from the NGSIM study, was used to train the model. Car-following behavior produced by the model were, then, compared with the observed one in 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 may even performance better than human drivers. The results indicate that reinforcement learning methods could contribute to the development autonomous driving systems.

REFERENCES

1. Wang, J., L. Zhang, D. Zhang, and K. Li. An Adaptive Longitudinal Driving Assistance System Based on Driver Characteristics. *IEEE Transactions on Intelligent Transportation Systems*, Vol.14, No.1, 2013, pp. 1–12.
2. Kuefler, A., Morton, J., Wheeler, T., and Kochenderfer, M. Imitating Driver Behavior with Generative Adversarial Networks. arXiv preprint arXiv:1701.06699, 2017.
3. Gazis, D.C., R. Herman, and R. W. Rothery. Nonlinear Follow-the Leader Models of Traffic Flow. *Operations Research,* Vol. 9, 1961, pp. 545–567.
4. Treiber, M., A. Hennecke, and D. Helbing. Congested Traffic States in Empirical Observations and Microscopic Simulations. *Physical Review E,* Vol. 62, 2000, pp. 1805–1824.
5. Basu, C., Yang, Q., Hungerman, D., Singhal, M., and Dragan, A. D. Do You Want Your Autonomous Car to Drive Like You? ACM/IEEE International Conference, 2017, pp.417–

425.
6. Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, and, Y. Tassa. Continuous Control with Deep Reinforcement Learning. *Computer Science*, Vol. 8, No. 6, 2015, pp. 187.
7. NGSIM, 2005. Next Generation SIMulation. US Department of Transportation, Federal Highway Administration http://ops.fhwa.dot.gov/trafficanalysistools/ngsim.htm (last accessed 12. 11.16.).
8. Li, Y. Deep Reinforcement Learning: An Overview. 2017.
9. Sutton, R. S., and A. G. Barto. Reinforcement Learning: An introduction. *Cambridge: MIT press*, 1998.
10. Grondman, I., L Busoniu, G. A. Lopes, and R. Babuska. A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C* (Applications and Reviews), Vol. 42, No. 6, 2012, pp. 1291–1307.
11. Mnih, V., K., Kavukcuoglu, D., Silver, A. A., Rusu, J., Veness, M. G., Bellemare, and S., Petersen. Human-Level Control through Deep Reinforcement Learning. *Nature*, Vol. 518, No. 7540, 2015, pp. 529–533.
12. Montanino, M., and Punzo, V. (2015). Trajectory Data Reconstruction and Simulation-based Validation Against Macroscopic Traffic Patterns. *Transportation Research Part B: Methodological*, Vol. 80, No. 1, 2015, pp. 82–106.
13. Vogel, K. A Comparison of Headway and Time to Collision as Safety Indicators. *Accident Analysis and Prevention*, Vol. 35, No. 3, 2003, pp. 427-433.
14. Zhang, G. H., Y. H. Wang, H. Wei, and Y. Y. Chen. Examining Headway Distribution Models Using Urban Freeway Loop Event Data. In *Transportation Research Record: Journal of the Transportation Research Board, No. 1999*, Transportation Research Board of the National Academies, Washington, D. C., 2007, pp. 141–149.
15. Jacobson, I. D., L. G., Richards, and A. R. Kuhlthau. Models of Human Comfort in Vehicle Environments. *Human Factors in Transport Research*, Vol. 2, 1980, pp. 24–32.
16. Krizhevsky, A., I. Sutskever, and G. E. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.
17. Kingma, D., and J. Ba. Adam: A Method for Stochastic Optimization. *Computer Science*, 2014, pp. 1412–6980.
18. Morikazu, T. On the Theory of the Brownian Motion. *Journal of the Physical Society of Japan*. Vol. 13, No. 11, 2007, pp. 1266–1280.
19. Schaul, T., Quan, J., Antonoglou, I., and Silver, D., Prioritized Experience Replay. *International Conference on Learning Representations*, 2016.