Lane Keeping Assist for an Autonomous Vehicle Based on Deep Reinforcement Learning

Conference Paper · April 2020

DOI: 10.4271/2020-01-0728

4 authors:

Qun Wang
Nanjing University of Science and Technology
9 PUBLICATIONS **29** CITATIONS
SEE PROFILE

Weichao Zhuang
Southeast University (China)
99 PUBLICATIONS **1,277** CITATIONS
SEE PROFILE

Liangmo Wang
Nanjing University of Science and Technology
116 PUBLICATIONS **2,046** CITATIONS
SEE PROFILE

Fei Ju
Nanjing University of Science and Technology
16 PUBLICATIONS **210** CITATIONS
SEE PROFILE

2020-01-0728 Published 14 Apr 2020

Lane Keeping Assist for an Autonomous Vehicle Based on Deep Reinforcement Learning

Qun Wang Nanjing University of Science and Technology

Weichao Zhuang Southeast University

Liangmo Wang and Fei Ju Nanjing University of Science and Technology

Citation: Wang, Q., Zhuang, W., Wang, L., and Ju, F., "Lane Keeping Assist for an Autonomous Vehicle Based on Deep Reinforcement Learning," SAE Technical Paper 2020-01-0728, 2020, doi:10.4271/2020-01-0728.

Abstract

Lane keeping assist (LKA) is an autonomous driving technique that enables vehicles to travel along a desired line of lanes by adjusting the front steering angle. Reinforcement learning (RL) is one kind of machine learning. Agents or machines are not told how to act but instead learn from interaction with the environment. It also frees us from coding complex policies manually. But it has not yet been successfully applied to autonomous driving. Two control strategies using different deep reinforcement learning (DRL) algorithms have been proposed and used in the lane keeping assist scenario in this paper. Deep Q-network (DQN) algorithm with discrete action space and deep deterministic policy gradient (DDPG) algorithm with continuous action space have been implemented, respectively. Based on MATLAB/Simulink, deep neural networks representing the control policy are designed. The environment as well as the vehicle dynamics are also modelled in Simulink. By integrating the proposed control method and a vehicle dynamics model, the lane keeping assist simulation is performed. Experimental results demonstrate that the vehicle travel along the centerline of the path and the controller reaches a steady state after a short time, validating the effectiveness of the proposed control method.

Introduction

With the development of automobile industry, traffic density continuously increases and the rate of traffic accidents as well as the burdens for drivers are increasing. Among all highway traffic accidents, lane departure crashes account for the vast majority of accidents and lead to hundreds of human deaths and huge economic losses every year. Miscellaneous advanced driver assistance systems (ADAS) have been paid more and more attention in the last few years. Mitsubishi Electric Corporation has developed multiple technologies to realize these systems [1, 2]. Lane keeping system is one of the ADAS systems. Over the last decade, LKA system has absorbed massive interests of the automobile and computer vision (CV) communities. Based on the lane-departure warning system (LDWS), the LKA system further controls the brake and steering coordination device. Both two systems utilize camera systems mounted behind the windscreen to detect the lane markings. If the vehicle approaches the identified lane markings and may detach from the current driving lane, the driver's attention is drawn to it by the vibration of the steering wheel, or by the sound. This greatly eases the pressure of the car driver and meanwhile provides information of the vehicle. Nowadays many vehicle manufacturers already offer lane-keeping assist systems. But the point is that the driver remains responsible for controlling the vehicle even after LKA system has been activated.

Before the application of artificial intelligence, people generally use feedback techniques to control ground vehicles. Sensory information collected from cameras or other sensors are processed in order to stabilize the vehicle [3]. The commands to the actuators, such as steering angle, throttle and break are generated by the controller to fulfill the tasks. A front wheel active steering mechanism was proposed which can automatically change steering angle to make sure the stability of vehicles [4]. Due to the outstanding success of the neural networks in the classification and regression problems, more researchers have focused on learning-based approaches to these problems. In fully autonomous driving scenario, the vehicle can recognize the lane and adjust the steering angle automatically, without human's participation which is an end-to-end control method. This can be achieved through the interaction between the vehicle and environment. Combined with deep learning (DL), DRL has made a lot of breakthroughs recently. Agents trained with deep reinforcement learning outperformed top human players in Atari game playing [5, 6], go playing [7], and complex strategic games [8]. With reinforcement learning, a policy can be learned without any labelled expert data.

In this paper, we propose two control strategies in the lane keeping assist scenario using two different state-of-the-art deep reinforcement learning algorithms. Both two methods take raw sensory information as inputs and output driving actions. The main difference between two algorithms

is that the action space of deep Q-network is discrete and of deep deterministic policy gradient is continuous. The learned policy is analyzed with explicit interpretation, and the policy can be used in other scenarios which means the agent can robustly learn to drive following the centerline.

The reminder of this paper is organized as follows. In the second part we introduce the background of RL in details. In the third part we deal with the knowledge of DRL. The fourth section describes the environment and creates the neural networks. Simulation and results are given in the fifth part. Finally, the conclusions are given in the last section.

Reinforcement Learning

In this section, we define the notation used in subsequent sections. If you want a more comprehensive overview of reinforcement learning, please refer to the second edition of Richard S. Sutton's book [9]. The reinforcement learning framework was formulated in [10] as a model to provide the best policy an agent can follow (best action to take in a given state), such that the total accumulated rewards are maximized when the agent follows that policy from the current and until a terminal state is reached.

MDP

Reinforcement learning is a branch of machine learning. In RL, an agent acts in an environment and tries to learn a policy, π, that maximizes a cumulative reward function. The agent is not told how to do like in supervised learning, but learns from trial and error. The policy defines which action, a, to take, given a state, s. Then the environment will turn into a new state, s', and return a scalar reward, r. *Markov decision process* (MDP) can be used to model the reinforcement learning problems [11], defined by the tuple $(\mathcal{S},\mathcal{A},P,r,\rho_0,\gamma,T)$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $P:\mathcal{S}\times\mathcal{A}\times\mathcal{S}\rightarrow\mathbb{R}_{\geq 0}$ is the transition probability distribution, $r:\mathcal{S}\times\mathcal{A}\rightarrow\mathbb{R}$ is the reward function, $\rho_0:\mathcal{S}\rightarrow\mathbb{R}_{\geq 0}$ is the initial state distribution, $\gamma\in(0,1)$ is the discount factor, and T is the horizon.

With access to the state and reward, the objective of reinforcement learning is to find the optimal policy π^* which optimize the expected sum of the discounted rewards as shown in equation (1).

$$R(\pi,r)=\mathbb{E}_{\pi}\left[\sum_{t=0}\gamma^{t}r(s_t,a_t)\right] \tag{1}$$

The solution $\pi^*(a_t|s_t)$ can then be utilized as the controller of the agent, which takes the current input state s_t and output the control action a_t to be applied to the vehicle. There are popular algorithms to solve this problem for finite state-space like *value iteration* and *policy iteration* [10].

Q-Learning

Q-learning is one of the most common algorithms in solving the MDP problems [12]. It is one of Temporal Difference (TD) learning which combines the ideas of Monte Carlo method and Dynamic Programming method [10]. Like Monte Carlo method, TD method does not require the environment dynamics and it is model-free. It only learns through the interaction with the environment. For every time step, we obtain action a_t, given current state s_t, based on the action-value function called Q: $\mathcal{S}\times\mathcal{A}\rightarrow\mathbb{R}$. The Q-learning algorithm is based on the Bellman equation as shown in equation (2).

$$Q_{t+1}(s,a)=Q_t(s,a)+\alpha\left[r+\gamma maxQ_t(s',a')-Q_t(s,a)\right] \tag{2}$$

where Q is the value corresponding to state s_t and action a_t, α is learning rate, r is the transition reward from state s_t to s_{t+1}. Term $r+\gamma maxQ_t(s',a')$ represents the total maximum reward in the transition from state s_t to s_{t+1}, which includes instantaneous reward r and maximum reward from state s_{t+1} to the future. This shares the same idea with DP bellman equation.

Deep Q-Network (DQN)

When the state space is discrete, the Q-function can easily be formulated as a table. However, as the number of states increases, or when the states are continuous, it is not possible to search through all possible state-action pairs for finding the optimal Q value. Thus, we use a deep neural network (DNN) to parametrize the Q-function as $Q(s,a;\theta)$, where θ are parameters of the neural network. With the help of deep neural network, the state space can be extended to high dimensional or even continuous. DQN builds a replay buffer $\mathcal{D}$, and defines an additional target Q network with parameters θ^-. During training, the transition pairs (s,a,r,s') are stored in the replay buffer and uniformly sampled a mini-batch at each step. The objective of deep Q-network is to minimize the mean square error (MSE) of the Q-values by equation (3).

$$L(\theta_i)=\mathrm{E}\left[(r+\gamma maxQ_t(s',|a',|\theta_i^-)-Q_t(s,|a,|\theta_i))^2\right] \tag{3}$$

The objective function is differentiable with respect to θ_i. Differentiating the loss function with respect to the weights, we arrive at the following gradient by equation (4).

$$\begin{aligned}\nabla_{\theta_i}L(\theta_i)=E[(r+\gamma maxQ_t(s',|a',|\theta_i^-)\\ -Q_t(s,|a,|\theta_i))\nabla_{\theta_i}Q_t(s,|a,|\theta_i)]\end{aligned} \tag{4}$$

Thus, the optimization problem can be solved using stochastic gradient descend (SGD).

Deep Deterministic Policy Gradient (DDPG)

While DQN is able to deal with continuous state problems, the action space is still discrete due to the limitation. Q-learning is not possible to be directly applied to continuous action spaces. In continuous spaces, finding the greedy policy requires an optimization of a_t at every timestep and this optimization is too slow to be practical with large, unconstrained function approximators and nontrivial action spaces. Combined with deep neural networks, the deep deterministic policy gradient algorithm is proposed [13].

DDPG builds a critic network $Q(s,a|\theta^Q)$ and an actor network $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ, as well as a target network Q' and μ' with weights $\theta^{Q'}$ and $\theta^{\mu'}$. The critic evaluates the value of the action taken in the given state and the actor takes states and outputs the corresponding action that maximizes the long-term reward. Transitions are stored in a replay buffer and mini-batches are sampled at each step. The Q value targets are set as shown in equation (5).

$$y_i = r_i + \gamma Q'\left(s_{i+1},|\mu'\left(s_{i+1}|\theta^{\mu'}\right),|\theta^{Q'}\right) \tag{5}$$

The critic is updated by minimizing the loss in equation (6).

$$L = \frac{1}{N}\sum_i \left(y_i - Q\left(s_i, a_i,|\theta^Q\right)\right)^2 \tag{6}$$

The policy network is updated using the sampled policy gradient in equation (7).

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q\left(s,|a|\theta^Q\right)\Big|_{s=s_i,a=\mu(s_i)} \nabla_{\theta^\mu}\mu\left(s|\theta^\mu\right)\Big|_{s_i} \tag{7}$$

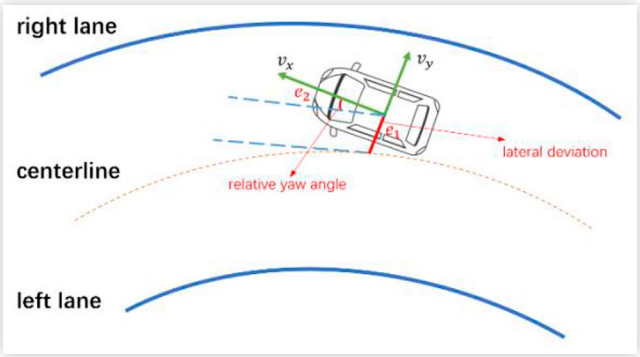
The target Q and policy networks are updated using temporal difference with respect to the online networks in equation (8).

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \tag{8}$$

DRL System for Lane Keeping Assist

In this section, we will focus on the DRL system for Lane Keeping Assist. More specifically, we will deal with agent creation and data processing. Both the discrete and continuous methods depend on the inputs from the surrounding environment in which we construct the input states, and the agent will output the driving actions for the vehicle autonomously. The training goal is to keep the vehicle travelling along the centerline of lanes by adjusting the steering angle as shown in Fig. 1. The vehicle equipped with a lane-keeping assist system has a sensor that measures the lateral deviation and relative yaw angle between the centerline of a lane and the car. The sensor also measures the current lane curvature and curvature deviation. The curvature of the lane is defined by a constant 0.001(m^{-1}). We then initialize the value for lateral deviation and relative yaw angle, and after each episode we replace these two values with two random values to begin a new episode.

FIGURE 1 Lane keeping control for reward design.



Vehicle Dynamics Modeling

The vehicle dynamics model is mainly used to calculate the load of the steering system and the needed parameters are β and ω_r . Given that the sideslip angle of the wheels in lane-changing steering is generally very small, the nonlinearity characteristic of the tire is ignored. So in this paper, we adopt a two DOF bicycle model as shown in Fig. 2. According to Newton's motion theorem:

$$\sum F_y = m\left(\dot{v} + u\omega_r\right)$$
$$\sum M_z = I_z\dot{\omega}_r \tag{9}$$

Where $\sum F_y$ is the sum of lateral forces, v is the velocity component of the centroid along the y direction, u is the longitudinal velocity of vehicle centroid, m is the vehicle mass, $\sum M_z$ is the torque around the Z axis of the vehicle (yaw torque), and I_z represents the vehicle moment of inertia around the Z axis.

According to the mechanical relations between tire and pavement:

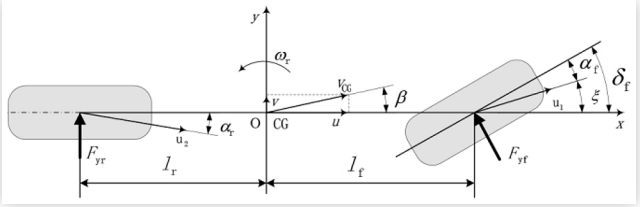
$$\sum F_y = F_{yf}cos\delta_f + F_{yr}$$
$$\sum M_z = F_{yf}cos\delta_f l_f - F_{yr}l_r \tag{10}$$

Where F_{yf}, F_{yr} are the lateral forces of front and rear wheels respectively, l_r is the horizontal distance between the vehicle centroid and rear axle. Considering that during a lane keeping scenario, the steering angle is usually relatively small, so an assumption is made that lateral acceleration is less than 0.4g, then the cornering properties can be considered as linear. In addition, $cos\delta_f \approx 1$ can be used in case that δ_f is very small. Then formula (10) can be rewritten as:

$$\sum F_y = C_f\alpha_f + C_r\alpha_r$$
$$\sum M_z = C_f\alpha_f l_f - C_r\alpha_r l_r \tag{11}$$

Where C_f, C_r are the cornering stiffness of front and rear wheels respectively, α_f, α_r are cornering angle of front and rear wheels respectively. Given that the steering angle is relatively

FIGURE 2 2DoF vehicle dynamics model



small, the side slip angle $\beta \approx tan\beta = v/u$, and the heading angle can be approximated by its tangent, that is $\xi \approx tan\xi$.

The vehicle motion differential equations can be written as:

$$\dot{\omega}_r = \frac{C_f l_f^2 + C_r l_r^2}{I_z u}\omega_r + \frac{C_f l_f - C_r l_r}{I_z}\beta - \frac{C_f l_f}{I_z}\delta_f$$

$$\dot{\beta} = \frac{C_f l_f - C_r l_r - u}{mu}\omega_r + \frac{C_f + C_r}{mu}\beta - \frac{C_f}{mu}\delta_f \quad (12)$$

DQN Agent Creation

The observations from the environment are the lateral deviation e_1, relative yaw angle e_2, their derivatives $\dot{e}_1, \dot{e}_2$ and their integrals $\int e_1, \int e_2$. The output of the agent is the steering angle in $[-15^\circ, 15^\circ]$ where the negative value is for turning right and the positive value is for turning left. The interface has a discrete action space where the agent can apply one of 31 possible steering angles from -15 degrees to 15 degrees. In order to balance exploitation and exploration, we take $\epsilon - greedy$ policy. During the training phase, the initial value is set as 1.0 and it decays linearly with the decay rate 0.005. The minimum value of ϵ will decrease to 0.01 which is used to keep a low-level exploration.

As an important part of RL framework, the reward signal drives the agent to reach the goal by rewarding good actions. In the lane keeping task, the goal is to control the vehicle following the centerline. As shown in Fig. 1, reward is a function of lateral deviation e_1 and relative yaw angle e_2 and their derivatives and the control input from previous time step. The simulation is terminated when lateral deviation $|e_1| > 1$.

A DQN agent approximates the long-term reward given observations and actions using a critic value function representation. The DQN critic network architecture is shown in Fig. 3. Here we use a deep neural network to represent the critic where *FC* stands for the fully connected layer. The critic network has two inputs, the state and action, and one output. Note that the dimensions of the input layers must match the dimensions of the corresponding environment observation and action specifications.

FIGURE 3 DQN critic network architecture

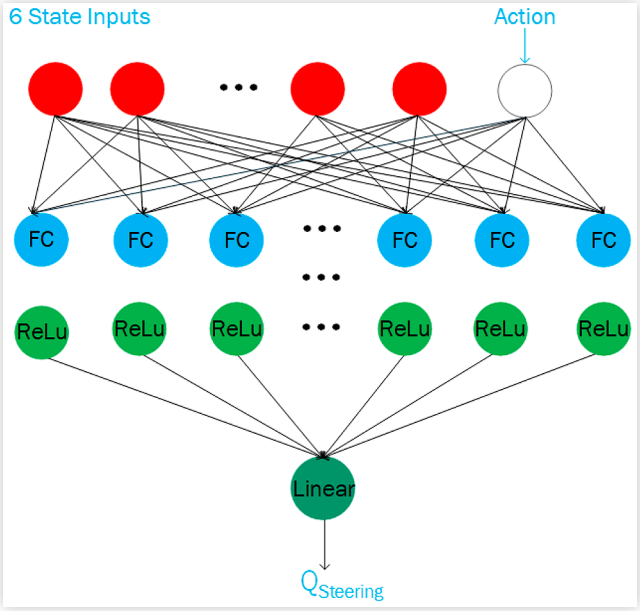


TABLE 1 Hyperparameters used to train the DQN agent

Discount factor, γ	0.99
Replay memory size	1000000
Learning rate	0.001
Mini-batch size	64
Max episodes	5000
Simulation duration	15 s
Sample time	0.1 s
Initial ϵ value	1
ϵ decay rate	0.005

Generally speaking, using the nonlinear neural network to represent the action-value function $Q(s_t, a_t)$ is unstable or even diverge. To solve this problem, we utilize two techniques proposed in [6], i.e. experience replay [14] and target networks. Experience replay mechanism is used in which we store the agent's experiences (s_t, a_t, r_t, s_{t+1}) at each time-step in a data set $\mathcal{D}$. During the training process, a mini-batch experiences are uniformly sampled from the buffer to update the critic network weights. The target network is the same as the critic network excepting the network weights. Then every C steps, update the target network weights by copying the weights of critic network. This slowly updating property keeps the target of critic more stable.

The hyperparameters of the DQN agent training are summarized in Table 1.

DDPG Agent Creation

In the lane keeping task, the agent makes a decision based on the current observation and receives the next observation and reward. This process repeats at every time step. We utilize RL to tackle this sequence decision-making problem. In DDPG, the steering command is a continuous value in $[-15^\circ, 15^\circ]$. Different from the stochastic policy, the deterministic policy $\mu(s_t)$ outputs a scalar action and executes it with probability 1, while stochastic policy $\pi(s_t, a_t)$ outputs the probability for every action. The reward function is set the same as in the DQN algorithm. We set the learning rate 0.001 for both the actor and critic network and the other parameters are the same as in the DQN settings.

The DDPG aims to directly search an optimal policy that maximizes the total reward. We utilize the actor-critic approach to approximate the optimal policy in the neural dynamic programming framework. The network architecture shown in Fig. 4 consists of two networks, the actor network for selecting an action and the critic network for evaluating the action in the underlying state.

FIGURE 4 DDPG actor-critic network architecture

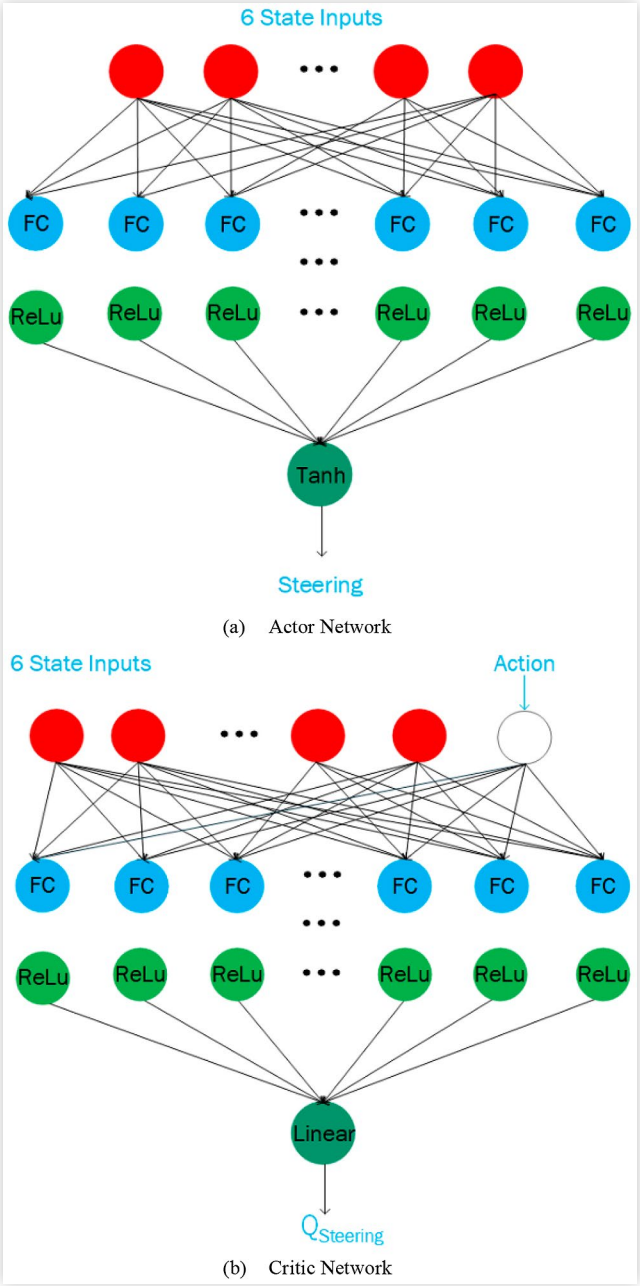


FIGURE 5 Episode reward of DQN agent.

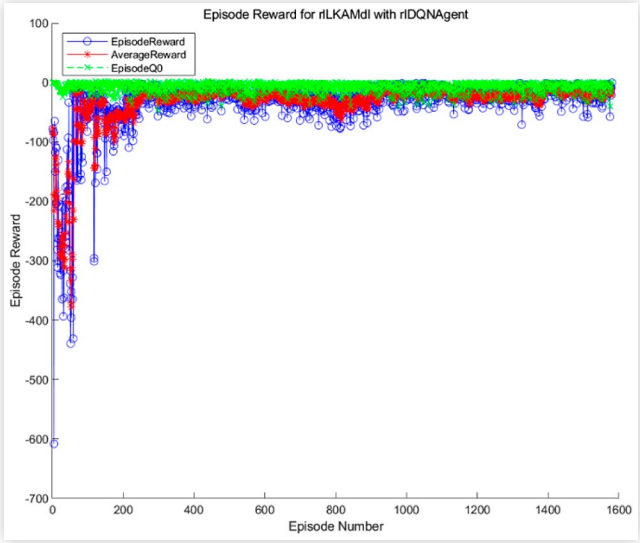
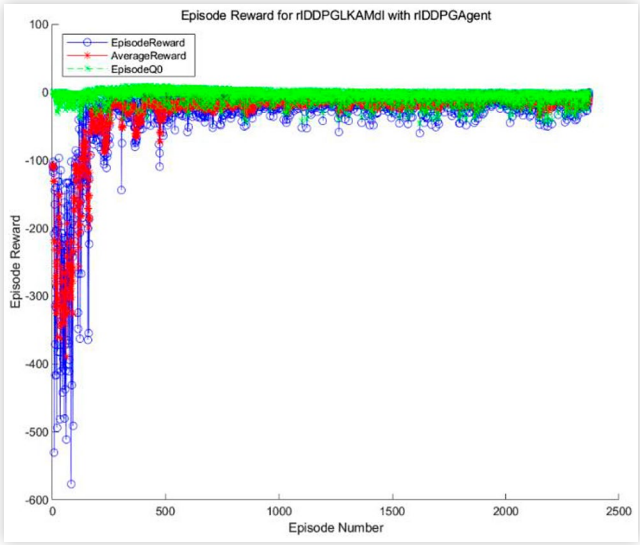


FIGURE 6 Episode reward of DDPG agent.



Simulation and Results

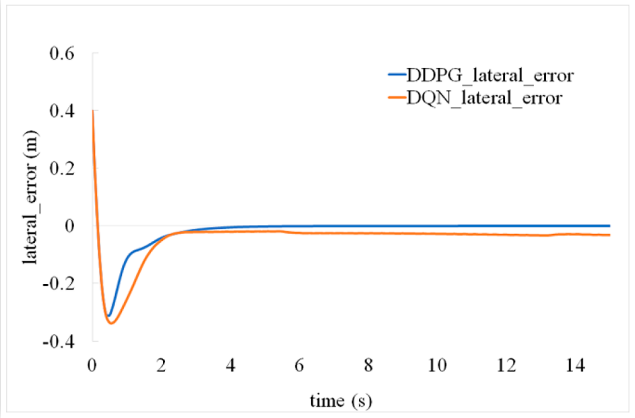
In this section, we discuss how we trained the agents and how we tested lane keeping function. At the beginning of each episode, we initialize the value of e_1 and e_2. Then the agent begins the training process. Each episode consists of 150 timesteps. When one episode is completed or when $|e_1| > 1$, the simulation will be terminated and then the simulator will restart for the next episode of training. In all experiments, the discount rate $\gamma = 0.99$, and the optimizer is Adam optimizer. We set the *MaxEpisodes* to be 5000 and *StopTrainingValue* to be -1. The training will be stopped when either of the two values is reached.

The accumulated reward per episode and average reward are depicted in Fig. 5 and Fig. 6, from which we can conclude that the accumulated reward is tending to converge, indicating that the agents successfully learn to take actions to maximize the reward. These trends manifest that the agents gradually learn effective lane keeping policy during the training process. We evaluate the performance of the DQN and DDPG algorithms by comparing their elapsed time and average reward. The results are summarized in Table 2. We can see that although the DDPG agent ran more episodes than the DQN agent, it took less time and gained a higher average reward which means a better performance. It is simple to figure out why the DQN algorithm takes more time. A common approach to adapting DQN to continuous domains is to simply discretize the action space. However, this has many limitations, most notably the curse of dimensionality: the number of actions increases exponentially with the number of degrees

TABLE 2 Comparison between different approaches

	DQN	DDPG
Number of training episodes	1580	2369
Elapsed time (s)	19678	10815
Average reward	-10.3303	-6.6313
Total number of steps	229923	337331
Hardware resource	CPU	CPU

FIGURE 7 Lane keeping lateral error evaluation.



of freedom. In this simplified simulation, the action space of DQN is discretized into 30 dimensionalities. And this leads to an 82% increase in time spent. It is hardly to implement DQN in real life to gain a fine control. Additionally, naive discretization of action spaces needlessly throws away information about the structure of the action domain, which may be essential for solving many problems.

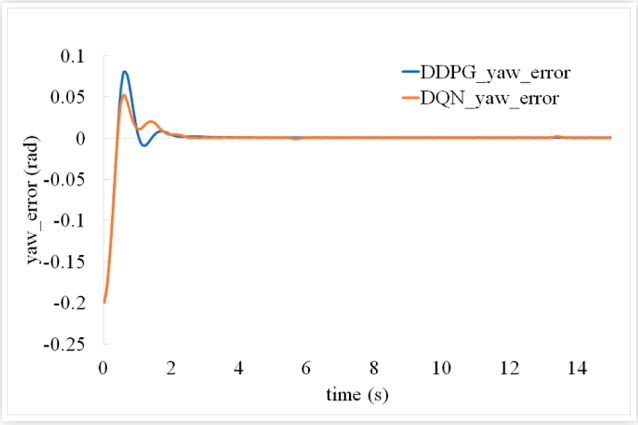
To better evaluate the performance of the model, we chose a same set of initial value of lateral deviation and relative yaw angle. Fig. 7 demonstrates that both algorithms manage to keep the lateral error in an acceptable range. From the perspective of peak value and convergence trend, the DDPG algorithm outperforms the DQN algorithm. The lateral deviation of DDPG algorithm is almost zero while DQN method tends to converge to *-0.03rad*. Actually, the action space of DDQN is continuous, so the agent can take any steering angle within the range to decrease the lateral deviation. The action space of DQN is discrete so the agent can only take a fixed steering angle and thus the lateral deviation converges to a certain value.

Fig. 8 shows the yaw error of the two algorithms in a lane keeping scenario. Note that both two policies converge to an ideal state in about *2.3s*. It means the policies can correctly guide the vehicle to drive.

Conclusion

In this paper, we implemented two deep reinforcement learning algorithms in lane keeping scenario to handle the problem of high-level lane keeping in an end-to-end manner. The proposed policy makes no assumptions about the

FIGURE 8 Lane keeping yaw error evaluation.



environment, it does not require any knowledge about the system dynamics. From the simulation results, we can conclude that the proposed two approaches are able to solve the lane keeping problem with respect to the reward function designed. The accumulated reward shows the trend of convergence which implies the agent has learned to make lane keeping decisions correctly under different situations.

Since the fully autonomous driving control includes both lateral and longitudinal control and due to the limitations of DQN, our future work is to integrate both two control methods to realize functions like lane changing and overtaking using DDPG. More vehicles will be added into the environment to enable multi-agent interaction.

References

1. Yokoyama, K., Iezawa, M., Akashi, Y., Satake, T. et al., "Speed Control of Parking Assist System for Electrified Vehicle," SAE Technical Paper 2015-01-0316, 2015, doi:https://doi.org/10.4271/2015-01-0316.
2. Yokoyama, K., Iezawa, M., Tanaka, H., and Enoki, K., "Development of Three-Motor Electric Vehicle "EMIRAI 2 xEV"," *SAE Int. J. Commer. Veh.* 8(1):197-204, 2015, doi:https://doi.org/10.4271/2015-01-1597.
3. Falcone, P., Borrelli, F., Asgari, J., Tseng, H.E. et al., "Predictive Active Steering Control for Autonomous Vehicle Systems," *IEEE Transactions on Control Systems Technology*, doi:https://doi.org/10.1109/TCST.2007.894653.
4. Singh, S., "Design of Front Wheel Active Steering for Improved Vehicle Handling and Stability," SAE Technical Paper 2000-01-1619, 2000, doi:https://doi.org/10.4271/2000-01-1619.
5. Mnih, K.K., Silver, D. et al., "Playing Atari with Deep Reinforcement Learning," arXiv preprint arXiv:1312.5602, 2013.
6. Mnih, V. et al., "Human-Level Control through Deep Reinforcement Learning," *Nature* 518(7540):529, 2015.
7. Silver, D., Huang, A., Maddison, C.J., Guez, A. et al., "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature* 529(7587):484, 2016.

8. Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M. et al., "AlphaStar: Mastering the Real-Time Strategy Game StarCraft II," DeepMind Blog, 2019.
9. Sutton, R.S. and Barto, A.G., *Reinforcement Learning: An Introduction* (MIT Press, 2018).
10. Sutton, R.S., "Learning to Predict by the Methods of Temporal Differences," *Machine Learning* 3(1):9-44, 1988.
11. Bellman, R., "A Markovian Decision Process," *Journal of Mathematics and Mechanics* 679-684, 1957.
12. Watkins, C.J. and Dayan, P., "Q-Learning," *Machine Learning* 8(3-4):279-292, 1992.
13. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N. et al., "Continuous Control with Deep Reinforcement Learning," arXiv preprint arXiv:1509.02971, 2015.
14. Lin, L.J., "Reinforcement Learning for Robots Using Neural Networks (No. CMU-CS-93-103)," Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

Contact Information

Qun Wang, Ph.D.
School of Mechanical Engineering, Nanjing University of Science & Technology
wangqun@njust.edu.cn,
+86-182-0509-3516.

ISSN 0148-7191