



Article

Self-Optimizing Path Tracking Controller for Intelligent Vehicles Based on Reinforcement Learning

Jichang Ma , Hui Xie ^{*}, Kang Song  and Hao Liu

State Key Laboratory of Engines, Tianjin University, Tianjin 300072, China; majichang@tju.edu.cn (J.M.); songkangtju@tju.edu.cn (K.S.); hao.liu@uisee.com (H.L.)

^{*} Correspondence: xiehui@tju.edu.cn

Abstract: The path tracking control system is a crucial component for autonomous vehicles; it is challenging to realize accurate tracking control when approaching a wide range of uncertain situations and dynamic environments, particularly when such control must perform as well as, or better than, human drivers. While many methods provide state-of-the-art tracking performance, they tend to emphasize constant PID control parameters, calibrated by human experience, to improve tracking accuracy. A detailed analysis shows that PID controllers inefficiently reduce the lateral error under various conditions, such as complex trajectories and variable speed. In addition, intelligent driving vehicles are highly non-linear objects, and high-fidelity models are unavailable in most autonomous systems. As for the model-based controller (MPC or LQR), the complex modeling process may increase the computational burden. With that in mind, a self-optimizing, path tracking controller structure, based on reinforcement learning, is proposed. For the lateral control of the vehicle, a steering method based on the fusion of the reinforcement learning and traditional PID controllers is designed to adapt to various tracking scenarios. According to the pre-defined path geometry and the real-time status of the vehicle, the interactive learning mechanism, based on an RL framework (actor-critic—a symmetric network structure), can realize the online optimization of PID control parameters in order to better deal with the tracking error under complex trajectories and dynamic changes of vehicle model parameters. The adaptive performance of velocity changes was also considered in the tracking process. The proposed controlling approach was tested in different path tracking scenarios, both the driving simulator platforms and on-site vehicle experiments have verified the effects of our proposed self-optimizing controller. The results show that the approach can adaptively change the weights of PID to maintain a tracking error (simulation: within ± 0.071 m; realistic vehicle: within ± 0.272 m) and steering wheel vibration standard deviations (simulation: within $\pm 0.04^\circ$; realistic vehicle: within $\pm 80.69^\circ$); additionally, it can adapt to high-speed simulation scenarios (the maximum speed is above 100 km/h and the average speed through curves is 63–76 km/h).

Keywords: autonomous vehicle; path tracking; reinforcement learning; adaptive PID; self-optimizing controller; vehicle control



Citation: Ma, J.; Xie, H.; Song, K.; Liu, H. Self-Optimizing Path Tracking Controller for Intelligent Vehicles Based on Reinforcement Learning. *Symmetry* **2022**, *14*, 31. <https://doi.org/10.3390/sym14010031>

Academic Editors: Rudolf Kawalla and Beloglazov Ilya

Received: 17 November 2021

Accepted: 17 December 2021

Published: 27 December 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Autonomous driving is an active research topic that has attracted considerable attention from both academic institutions and manufacturing companies, owing to its broad application prospects in intelligent transportation systems. Automated vehicle software mainly involve environmental perception, decision planning, and motion control. Intelligent vehicles are non-linear motion systems, and their dynamic parameters change significantly with different speeds and road conditions, especially at high speeds of motion and during complex trajectories. This makes the path tracking control problem one of the most challenging aspects of this field. A closed-loop control system, which is composed of people, vehicles, and roads, as shown in Figure 1, is influenced by inevitable disturbances

both inside and outside the vehicle, such as road adhesion coefficients, driving air resistance, and power output device, etc. As a result, the vehicle's model parameters change in real time, and there is a dynamic deviation between the vehicle's operating state and the desired state. Therefore, the human driver needs to constantly adjust the vehicle's state of motion to keep it on the desired path. The development process of the intelligent controller should learn the control mechanism of the human driver. With that in mind, the question of whether the path tracking controller can realize online self-optimization, according to the changes of the environment, is a key point for research.

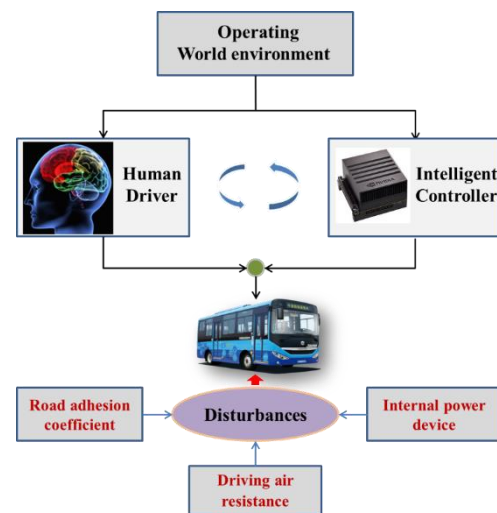


Figure 1. Closed-loop control system for vehicles.

There are many methods of providing state-of-the-art tracking performance, which can be divided into three typical categories, as follows: traditional classical control, model-based control, and intelligent adaptive control.

With regard to classic approaches, proportional–integral–derivative (PID) control is one of the most widely used methods in actual systems, with the advantages of a simple structure and easy implementation [1]. Previous studies [2,3] have presented an algorithm for using a PID controller to solve the path tracking problem for autonomous ground robots. Their results showed that the PID controller was capable of tracking a path. Regarding the traditional proportional–integral–derivative control strategy, due to the fixed constant PID control parameters, its application scenarios have limitations. A detailed analysis shows that a PID controller inefficiently reduces the lateral error under complex trajectories and variable speed conditions; when the road curvature is large, or when the vehicle is driving at high speeds, it is easy to deviate from the expected trajectory—as shown by the red arrows in Figure 2.

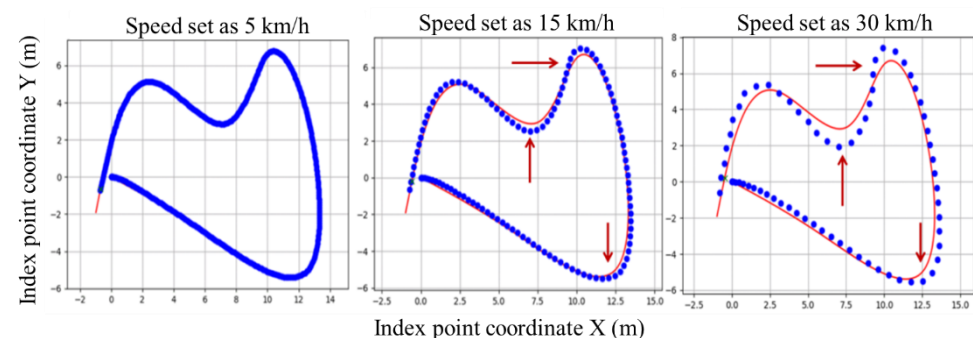


Figure 2. The performance of traditional PID control in different speed conditions.

As for the model-based categories, most of the proposed methods for path tracking control are based on modeling the vehicle dynamics [4–6], including the tire forces and

the moments generated by the wheels. Some previous studies [7–9] have used model predictive control (MPC), in which an autonomous vehicle was directed to follow a pre-planned trajectory and a dynamic model of the system was used to predict an optimal sequence. However, MPC requires a heavy computational load, owing to its complex design. Therefore, this algorithm is unsuitable for high-speed autonomous driving and complex road trajectories. In [10], the authors modified the lateral dynamics of a vehicle and used a linear quadratic regulator (LQR) controller. A bicycle model was used to obtain the feedforward and feedback parts of the steering input. The optimal control parameters were obtained, based on a cost function. The authors of [11] compared and analyzed various control strategies for path tracking applications by running a vehicle model in a prescribed environment. In general, as described in the literature, the results may vary when additional control inputs, such as brake control and accelerator control, are brought into the system. We can conclude that complex trajectories and high-speed driving have important impacts on a vehicle motion model; thus, model-based controllers have limitations and can only be applied to simple roads in low-speed driving scenarios. In addition, intelligent driving vehicles are highly non-linear objects and a high-fidelity model is unavailable in most autonomous systems. As for the model-based controller (MPC or LQR), the complex modeling process may increase the computational burden; moreover, changes in model parameters may lead to a decrease in control performance. Therefore, for the vehicle motion control system, it is urgent to develop a tracking method with an adaptive and effective control framework for real-time implementation.

More recently, approaches in the third category—i.e., intelligent adaptive approaches—have been proposed to mitigate the aforementioned problems. These types of methods provide the ability to adapt; for example, they can undertake corrective control actions based on changes in the environment. Several studies have been conducted based on these methods, aiming to solve the problems noted above. A fuzzy controller with a parameter PID self-tuning module was introduced in [12,13] to provide a mobile robot with complete path tracking control; this showed advantages in providing a rapid response, high stability, and high tracking accuracy. However, the design of fuzzy adaptive PID control requires a significant amount of prior knowledge, and, in reality, it is difficult to obtain such comprehensive prior knowledge when a vehicle travels in unknown situations. An adaptive PID control method, based on neural networks, was presented in [14,15]. Nevertheless, a neural network generally uses supervised learning to optimize parameters, so it is also limited by some application conditions; for instance, it is difficult to obtain the exact teacher signal for supervised learning. Moreover, it does not work in real-time in the context of line optimization.

In order for automated vehicles to improve their adaptability, it is essential for them to interact with their environments and promote the natural evolution of a control policy. The essence of reinforcement learning is to learn an optimal control policy through interaction with the environment, which provides an effective way to solve the online optimization control problem of path tracking. In recent years, many exciting RL applications have been proposed in the context of self-driving vehicle control; for example, previous studies [16–18] proposed a framework for autonomous driving using deep RL. They adopted the deep deterministic policy gradient (DDPG) algorithm to manage complex road curvatures, states, and action spaces in a continuous domain and tested the approach in an open-source 3D car racing simulator called “TORCS” [19]. The Robotics and Perception Group at the University of Zurich created an autonomous agent for a GT Sport car racing simulator [20] that matched or outperformed human experts in time trials; this worked by defining a reward function for formulating the racing problem and a neural network policy for mapping input states to actions, then, the policy parameters were optimized by maximizing the reward function using the soft actor–critic algorithm [21]. Reference [22] introduced a robust drift controller based on an RL framework with a soft actor–critic algorithm and used a “CARLA” simulator [23] for training and validation. The controller was capable of making the vehicle drift through various sharp corners quickly and stably in an unseen map

and was further shown to have excellent generalization ability. It could directly manage unseen vehicle types with different physical properties, such as mass and tire friction. Reinforcement learning, as a method for solving the optimization problem of continuous action space under uncertain environments, has also been extensively researched in the path tracking control process of UAVs and robots [24–30]. It is a data-driven control strategy that does not depend on the precise model of the controlled object [31]; therefore, the path-following control problem of autonomous vehicles can be quantitatively described as a sequential data optimization control problem [32–36].

The success of the deep RL algorithms proves that control problems can be naturally solved by optimizing policy-guided agents in a continuous state and action space. However, so far, RL research on automated vehicle control is mainly limited to simulation environments, such as TORCS, GT Sport, and CARLA, and only a few, comparably simple examples have been deployed in real systems, such as in references [37,38], which demonstrated the first applications of deep RL to realistic autonomous driving. In those studies, the RL agent evaluated and improved its control policy in a trial-and-error manner; thus, it would be dangerous and costly to train such an agent on a real vehicle; moreover, particularly with dynamically balancing systems, such a process is complicated and expensive.

Nevertheless, the generality of RL makes it a useful framework for autonomous driving. Most importantly, it provides a corrective mechanism for improving an online control policy, based on interacting with the environment. Thus, in this paper, a novel RL-based method is proposed for use in path tracking control. We demonstrate a self-optimizing controller structure incorporating a simple physics-based model and adaptive PID control based on RL; additionally, we present a newly developed approach for training an actor–critic network policy on a simulator and transferring it to a state-of-the-art realistic vehicle. This system can be used to track a path under complex trajectories and different speed conditions, and its performance is comparable to that of professional drivers. The main contributions of this paper are as follows:

- In this paper, we propose a self-optimized PID controller with a new adaptive updating rule, based on a reinforcement learning framework for autonomous vehicle path tracking control systems, in order to track a predefined path with high accuracy and, simultaneously, provide a comfortable riding experience.
- According to the pre-defined path geometry and the real-time status of the vehicle, the environment interactive learning mechanism, based on RL framework, can realize the online self-tuning of PID control parameters.
- In order to verify the stability and generalizability of the controller under complex paths and variable speed conditions, the proposed self-optimizing controller was tested in different path tracking scenarios. Finally, a realistic vehicle platform test was carried out to validate the practicability.

The remainder of this paper is organized as follows. In Section 2, we introduce the vehicle dynamics and kinetics models and define the state–action spaces and reward function. In Section 3, we provide an overview of the proposed self-optimizing controller structure and then introduce the actor–critic framework and algorithm. In Section 4, we introduce the simulation system and realistic autonomous platform, describe the experimental settings, and analyze the test results. Finally, we draw conclusions in Section 5 and propose future work in Section 6.

2. Vehicle Dynamic Constraints and Reference Trajectory Generation

An intelligent vehicle is a multi-input and multi-output electrical system with non-linear characteristics, and it is difficult to construct an accurate dynamic model for it. In addition, the dynamic characteristics of the system are also affected by the operating speed and environment, especially for unmanned vehicles running at high speeds, and the dynamic parameters will change significantly with the vehicle speed. When accounting for the non-linearity and time-varying characteristics of an intelligent vehicle system, traditional control methods, based on PID, LQR, and MPC, experience difficulties in meeting the

current control requirements. Moreover, the design of a path tracking controller should provide online learning and self-optimization abilities. Therefore, the development of intelligent control algorithms combining mechanism models and data-driven methods has become a popular research topic in the field of control engineering applications. Here, we discuss a self-optimizing controller, based on online RL, and show that a simple path tracking architecture can enable an automated vehicle to track a path accurately, while using a complex trajectory. The essence of this approach is to reduce the error between the vehicle and reference path by controlling the lateral and longitudinal movement of the vehicle. Therefore, the key is to calculate control variables that satisfy the constraints of the dynamic model and the geometric constraints of the actuator. The proposed self-optimizing control structure, based on RL, begins with the vehicle dynamic constraints, which are based on a simplified bicycle model. Schematics of the vehicle dynamic model and kinematic state model are shown in Figure 3a,b, respectively. As shown in Figure 3, XOY is the inertial coordinate system fixed on the ground and xoy is the vehicle coordinate system fixed on the vehicle body.

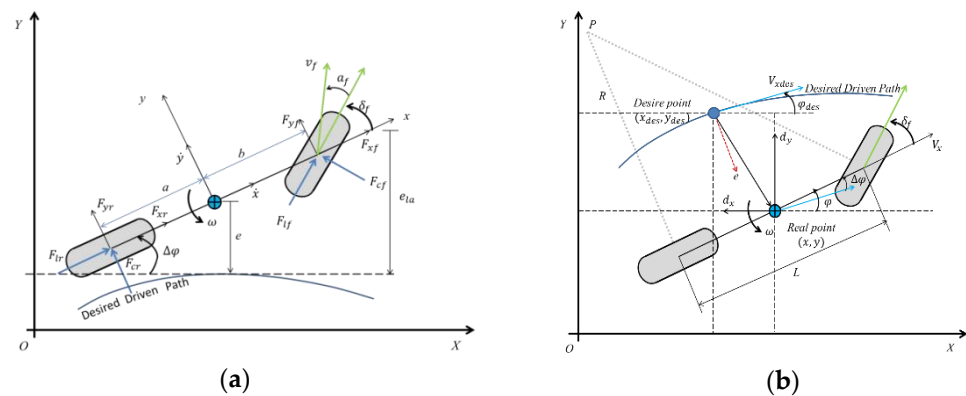


Figure 3. Schematic diagram of the vehicle model and description of the path tracking states. (a) Vehicle dynamics model; (b) vehicle kinematics model.

We can construct a 2 degrees of freedom (2-DOF) vehicle dynamic model for describing the motion of the vehicle, based on the following key assumptions [39,40]:

1. By ignoring the movement in the Z-axis direction, only the movement in the XY horizontal plane is considered; this is referred to as the planar bicycle model.
2. By assuming that the rotation angles of the tires on the left and right sides of the vehicle body are identical, the tires on both sides can be combined into one tire.
3. The rear wheels are not considered as steering wheels; only the front wheels are.
4. The aerodynamic forces are ignored.

The actuation of the steering angle, δ_f , at the front wheel results in the generation of lateral tire forces. According to Figure 3a, F_{cf} and F_{cr} are the two lateral forces acting on the front and rear tires, respectively, while α_f is the slip angle of the front wheel. The two lateral forces cause the vehicle to produce the yaw rate ω , which describes the angular rotation of the vehicle. In addition to the vehicle dynamics model constraints, two additional state variables are required, to account for the vehicle kinematics model, which shows the vehicle's position relative to the desired driven path. As shown in Figure 3b, the lateral path deviation, also referred to as the lateral error, e , is the distance from the vehicle's center of gravity to the closest point on the desired driven path. The vehicle heading deviation, also referred to as the heading error, $\Delta\varphi$, is defined as the angle between the vehicle's center line and a tangent line drawn on the desired driven path at the closest point. The specific descriptions and meanings of the remaining parameters are listed in Table 1.

Table 1. Specific definitions and meanings of the vehicle model parameters.

Symbol	Parameter	Units
F_{lf}, F_{lr}	Front and rear tires longitudinal force	N
F_{cf}, F_{cr}	Front and rear tires lateral force	N
F_{xf}, F_{xr}	Front and rear tires force in the x direction	N
F_{yf}, F_{yr}	Front and rear tires force in the y direction	N
a	Front axle to center of gravity (CG)	m
b	Rear axle to CG	m
δ_f	Steer angle input	Rad
α_f	Front tire slip	rad
ω	Yaw rate	rad/s
e	Lateral path deviation	m
$\Delta\varphi$	Vehicle heading deviation	rad
V_x	Longitudinal velocity	m/s

Figure 1 depicts a diagram of the two-wheel vehicle model, which considers the longitudinal, lateral, and yaw motions. By analyzing the forces on the x -axis, y -axis, and z -axis, respectively, the equations of motion for the 2-DOF states are given as follows:

$$X - axis\ direction\ ma_x = 2(F_{xf} + F_{xr}) \quad Y - axis\ direction\ ma_y = 2(F_{yf} + F_{yr}) \quad Z - axis\ direction\ I_z\dot{\omega} = 2aF_{yf} - 2bF_{yr} \quad (1)$$

The acceleration in the Y -axis direction consists of two aspects: the displacement acceleration, \ddot{y} , and centripetal acceleration, $V_x \cdot \omega$. Then, Formula (1) can be rewritten as follows:

$$m(\ddot{y} + V_x \cdot \omega) = 2(F_{yf} + F_{yr}) \quad (2)$$

According to the lateral force of the tire, the slip angle of the front wheel is $\alpha_f = \delta - \delta_f$, where δ is front wheel angle, and δ_f is the angle between the front wheel speed direction and the vehicle speed direction. Then, the lateral force experienced by the front wheels can be expressed as follows:

$$F_{yf} = C_{af}(\delta - \delta_f) \quad (3)$$

Similarly, the lateral force of the rear wheel can be expressed as $F_{yr} = C_{ar}(-\delta_r)$, where C_{af} and C_{ar} are the cornering stiffness values of the front and rear wheels, respectively.

δ_f and δ_r can be approximated by the following formula:

$$\delta_f = (V_y + a\omega) / V_x \delta_r = (V_y - b\omega) / V_x \quad (4)$$

As shown in Figure 3b, e is the lateral path deviation, $\Delta\varphi$ is the vehicle heading deviation, φ is vehicle heading angle, and φ_{des} is the road desired heading angle. According to the kinematic formula, the desired angular velocity required by the vehicle at the turning radius R can be denoted as the following formula:

$$\Delta\varphi = \varphi - \varphi_{des} \dot{\varphi}_{des} = V_x / R \quad (5)$$

The desired lateral acceleration required by the vehicle at the turning radius R can be written as the following formula:

$$a_{ydes} = V_x^2 / R \quad (6)$$

The lateral acceleration error is recorded as \ddot{e} , $\omega = \dot{\varphi}$.

$$\ddot{e} = a_y - a_{ydes} = (\ddot{y} + V_x\Delta\omega) - V_x^2 / R = \ddot{y} + V_x(\dot{\varphi} - \dot{\varphi}_{des}) \quad (7)$$

That is:

$$\ddot{e} = \ddot{y} + V_x\dot{\Delta\varphi} \quad (8)$$

φ_{des} is the desired heading angle of the reference driven path and is calculated using the path planning formula, as follows:

$$\dot{\varphi}_{des} = V_x / R = V_x * K \quad (9)$$

where K is the desired road curvature, which can be obtained from the collected high-precision map data. Substituting Formulas (5) and (9) into Formula (1) can obtain the following expression of $\ddot{\Delta\varphi}$:

$$\ddot{\Delta\varphi} = \frac{2aF_{yf} - 2bF_{yr}}{I_z} - K\dot{V}_x - \dot{K}V_x \quad (10)$$

$$\begin{cases} e = d_x * \cos\varphi_{des} + d_y * \sin\varphi_{des} \\ \dot{e} = V_x * \sin\Delta\varphi \\ \Delta\varphi = \varphi - \varphi_{des} \\ \dot{\Delta\varphi} = \dot{\varphi} - \dot{\varphi}_{des} \end{cases} \quad (11)$$

Here, e is the lateral error, \dot{e} is the rate of the lateral error, $\Delta\varphi$ is the heading error, and $\dot{\Delta\varphi}$ is the rate of the heading error. φ is the heading angle of the vehicle body, which can be obtained using a vehicle-mounted inertial measurement unit (IMU) sensor.

$$\begin{cases} \dot{X} = AX + Bu \\ Y = CX + Du \end{cases} \quad (12)$$

According to the state space Equation (12), the dynamic model of the steering wheel control can be obtained as follows:

$$\frac{d}{dt} \begin{bmatrix} e \\ \dot{e} \\ \Delta\varphi \\ \dot{\Delta\varphi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & A_1/V_x & -A_1 & A_2/V_x \\ 0 & 0 & 0 & 1 \\ 0 & A_3/V_x & -A_3 & A_4/V_x \end{bmatrix} \begin{bmatrix} e \\ \dot{e} \\ \Delta\varphi \\ \dot{\Delta\varphi} \end{bmatrix} + \begin{bmatrix} 0 \\ B_1 \\ 0 \\ B_2 \end{bmatrix} \delta + \begin{bmatrix} 0 \\ \frac{A_2}{V_x} - V_x \\ 0 \\ A_4/V_x \end{bmatrix} \varphi_{des} \quad (13)$$

For the above calculations, a , b , and c are, respectively, determined as follows:

$$\begin{cases} A_1 = -2(C_{af} + C_{ar})/m \\ A_2 = -2(C_{af}l_f - C_{ar}l_r)/m \\ A_3 = -2(C_{af}l_f - C_{ar}l_r)/I_z \\ A_4 = -2(C_{af}l_f^2 + C_{ar}l_r^2)/I_z \end{cases} \quad (14)$$

$$\begin{cases} B_1 = 2C_{af}/m \\ B_2 = 2C_{af}l_f/I_z \end{cases} \quad (15)$$

The time series data, e , \dot{e} , $\Delta\varphi$, $\dot{\Delta\varphi}$, are taken as the state variables, while δ is the control variable. Aiming at the path tracking control problem of automatic driving, the conventional PID control law is expressed as follows:

$$u(t) = K_{p1}e(t) + K_{d1}\dot{e}(t) - K_{p2}\Delta\varphi(t) - K_{d2}\dot{\Delta\varphi}(t) \quad (16)$$

Here, K_p and K_d are the proportional and differential gain coefficients, respectively.

The above traditional PID control is just a preliminary approach under ideal dynamics models; however, the dynamic characteristics of the system will, in fact, be affected by the operating speed and environment. Especially for unmanned vehicles running at a high speed, the dynamic parameters can change significantly with the vehicle speed, making it difficult for automatic path tracking control to guarantee performance and stability over a wide range of parameter changes. A key point of this paper is to set the path tracking process of an autonomous vehicle as a Markov decision process (MDP) of sequence data. Therefore, we need to accurately define the state space (S) and action space (A) and design a reward function (R) in combination with the vehicle dynamics model.

(a) State space variable description

The parameters of the state space are the environmental observation data S_t received by the controller at each time step. Many sensors are carried by driverless cars, including cameras, light detection units, ranging units, IMU, and GPS units. However, this paper

focuses on path tracking control, where the control of vehicle position and pose is the key issue; therefore, the IMU and GPS output data are selected, together with the vehicle dynamic constraints, and we can obtain the state space variable S_t . Figure 3b demonstrates a desired driven path and the related error variables, which are the lateral track error, e , its time derivative, \dot{e} , the heading angle error, $\Delta\varphi$, and its time derivative, $\dot{\Delta\varphi}$. We use the parameters mentioned above to describe the state of the vehicle in a specific traffic scene, as given by $S_t = \{e, \dot{e}, \Delta\varphi, \dot{\Delta\varphi}\}$

(b) Action space variable description

As mentioned above, the dynamic characteristics of the system will be affected by the operating speed and environment, especially for unmanned vehicles running at high speeds. To allow the system to automatically adapt to changes in the environment and parameters, we designed a self-optimizing PID controller based on an RL framework, in which control parameters could be adjusted automatically online, based on real-time performance requirements. The calculation can be expressed as follows:

$$K(t) = K_0 + \Delta K \quad (17)$$

In the above, K_0 is a constant vector, determined by expert experience, and ΔK is the self-learning gain vector. Thus, the traditional PID control (Equation (16)) can be rewritten, as follows:

$$u(t) = (K_{p1} + \Delta K_{p1})e(t) + (K_{d1} + \Delta K_{d1})\dot{e}(t) - (K_{p2} + \Delta K_{p2})\Delta\varphi(t) - (K_{d2} + \Delta K_{d2})\dot{\Delta\varphi}(t) \quad (18)$$

The control parameters are adjusted to realize the dynamic compensation of the system. We use the parameters mentioned above to describe the action space, which is given by $A_t = \{\Delta K_{p1}, \Delta K_{d1}, \Delta K_{p2}, \Delta K_{d2}\}$.

(c) Reward function description

As a key element of the RL framework, the reward signal drives the agent to reach the goal by rewarding good actions and penalizing poor actions. In a path tracking control task, the goal of the reward function design is to find the optimal control strategy for making the vehicle follow the reference trajectory as closely as possible while reducing the tracking error. To optimize the path tracking performance, in this paper we adopt a piecewise linear error reward function. Its design criteria are as follows:

$$R_{t1} = \begin{cases} k|y - y_D|, & |y - y_D| > e_1 \\ -c & e_2 \leq |y - y_D| \leq e_1 \\ 0 & |y - y_D| \leq e_2 \end{cases} \quad (19)$$

Here, k , c , e_1 , and e_2 are preset constants; $e_2 \leq e_1$, $k \leq 0$ is a proportional coefficient; and the lateral deviation is $e = |y - y_D|$, as shown in Figure 3b. The design goal of the above reward function is to make the vehicle's lateral deviation as close as possible to the given reference trajectory, which exhibits exponential convergence. In addition, by combining the constraints of the vehicle dynamics model to design the reward function, R_{t2} , such that the vehicle's heading deviation is parallel to the road curvature as much as possible (as shown in the figure), the reward function expression of R_{t2} can be expressed as follows:

$$R_{t2} = V_x \cos(\Delta\varphi) - V_y \sin(\Delta\varphi) - V_x |y - y_D| \quad (20)$$

As shown in Figure 4, the vehicle needs to drive along the centerline of the lane. In an ideal state, the lateral deviation, e , and the heading angle deviation, $\Delta\varphi$, between the center axis of the vehicle and the centerline of the lane, are close to zero in value. The objective of the controller is to minimize its lateral deviation, e , and heading angle deviation, $\Delta\varphi$, from the lane centerline. In the above, $\Delta\varphi$ is the heading angle deviation. The design principle of the reward function is based on maximizing the axial speed of the vehicle (V_x) and minimizing the lateral speed of the vehicle (V_y). We add a penalty term if the control object continues to deviate significantly from the center of the road (the third term);

this will greatly improve the stability of the control system. The final reward function is $R_t = R_{t1} + R_{t2}$. The optimization goal of the RL controller is to maximize the total reward, as follows:

$$J = \sum_{t=0}^T \gamma * R_t \quad (21)$$

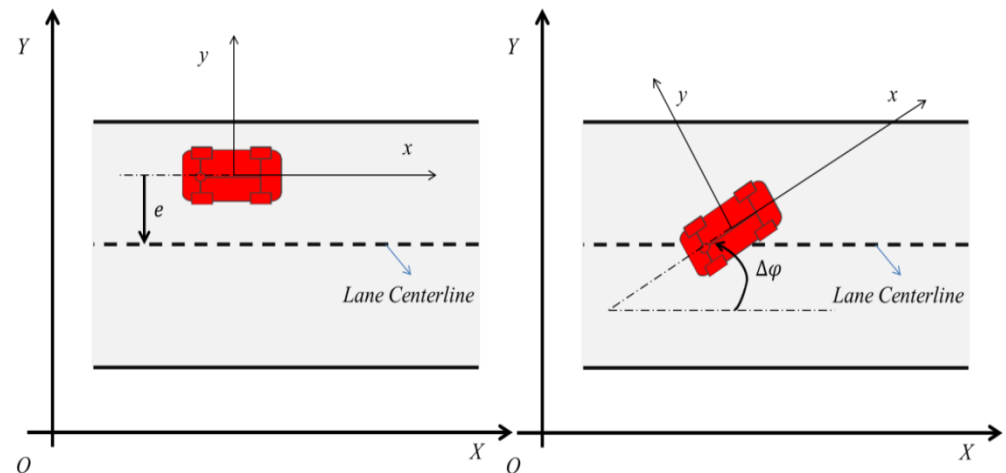


Figure 4. Schematic diagram of the vehicle lateral error deviation and heading angle deviation.

Here, γ is a discount factor and is usually a constant close to 1; in this paper, $\gamma = 0.95$. The objective of the controller is to maximize the total reward, J , and to minimize its lateral deviation (e) and heading angle deviation ($\Delta\varphi$) from the original lane. By optimizing the performance indicators, the state of the controlled system can be made to follow the reference state.

3. Self-Optimizing Path Tracking Controller Based on a Reinforcement Learning (RL) Framework

We aimed to find a control policy that minimizes the distance to the center line of the track for a given physical model and road trajectory, as well as for different vehicle speeds. In contrast to previous approaches relying on classical trajectory control, our approach leverages RL to train an actor–critic network that directly maps from observations and then provides an input to the adaptive PID controller, to calculate the vehicle control commands. To achieve this goal, we first introduced a physical model and defined a reward function for formulating the path tracking problem; these were used to perform the online adaptive tuning of the PID parameters so as to improve the path tracking effects of autonomous vehicles under complex road trajectories. In this paper, we show that, in a self-optimizing path tracking controller structure, based on reinforcement learning, the design of the controller has three advantages. Firstly, it introduces an online self-learning mechanism into the traditional controller, with the environment interactive learning mechanism, based on reinforcement learning, which can realize the optimization of PID control parameters. Secondly, it reduces the exploration space of the RL to find the optimal control parameters, which will greatly improve the learning efficiency. Thirdly, it breaks through the limitations of RL, in regard to only being used in simulation and game scenarios. To the best of our knowledge, this is the first demonstration of a deep RL agent driving on real autobus vehicle. In this section, we first present an overview of our proposed framework for the self-optimization controller and then describe each module. Our architecture consists of four modules, as follows: the operating environment, the data bridge, the RL framework, and the vehicle control module. Figure 5 shows the structure of the self-optimizing PID controller based on the RL framework.

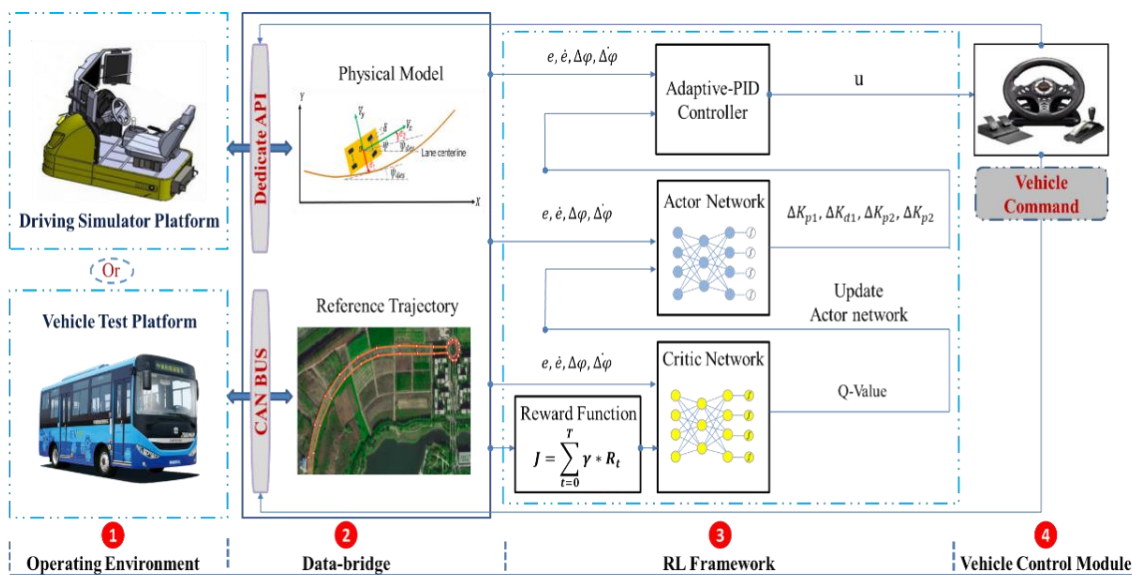


Figure 5. Structure of a self-optimizing proportional–integral–derivative (PID) controller based on a reinforcement learning (RL) framework.

The operating environment receives a series of actions from the vehicle control module, evaluates the quality of these actions, and converts them into a scalar reward, J , to be fed back to the RL framework using a data bridge.

The data bridge (or data buffer) allows for the interactions between the operating environment and the RL framework. Based on the physical model and reference trajectory, the current state values $S_t = \{e, \dot{e}, \Delta\varphi, \dot{\Delta\varphi}\}$ are extracted from the operating environment. The data bridge forwards the vehicle command (steering angle) to the environment object (driving simulator platform and vehicle test platform) for execution. After execution, the research object returns the corresponding reward value and the next state: S_{t+1} .

The RL framework consists of two parts: an actor network and a critic network. The actor network comprises a policy function, responsible for generating the actions, $A_t = \{\Delta K_{p1}, \Delta K_{d1}, \Delta K_{p2}, \Delta K_{d2}\}$. The critic network comprises a value function used to calculate the Q-value, which is responsible for evaluating the performance of the actor network, based on the DDPG algorithm, and for guiding the actor network to generate the appropriate actions, A_{t+1} , for the next state to maximize the future expected cumulative reward.

The vehicle control module receives the output, $u(t)$, from the self-optimizing PID controller and then forwards it to the environment object for execution, using the data bridge. To obtain the equations of motion for the self-optimizing controller, the expression of u is defined as Formula (18).

In the above, K_p , K_d are fixed gain constants, determined based on the developer's experience, while ΔK_p and ΔK_d are the output values of the actuator network and are used to adjust the fixed gain constant. The self-optimizing controller generates a time series control quantity $u(t)$ and acts on the steering wheel control command, for the vehicle to realize the adaptive ability in the path tracking control. Compared with traditional PID controllers (Equation (9)), our self-optimizing RL-based controller increases the system's ability to compensate for dynamic errors.

Summarizing the process in Figure 3, the self-optimizing PID controller, based on RL, is mainly composed of two parts, as follows: an actor network and a critic network. The actor network is a strategy function responsible for generating actions and interacting with the environment. The critic network is a value function responsible for evaluating the performance of the actor and guiding the output of the actor network in the next stage. Based on the content discussed in Section 2, the design of the reward function needs to

consider the tracking performance of the system with regard to the reference trajectory and the constraints of the dynamic model. The calculation process and workflow architecture of the self-optimizing PID controller, based on RL, are as follows:

- (1) Initialize the state of the controlled object, including the initial position and heading angle of the vehicle.
- (2) Pre-set the parameters for the optimizing controller, including the weight of the actor–critic network, the learning rate, the discount factor, and the selection of the activation function.
- (3) Adopt the DDPG algorithm to train the model, where the actor network outputs the PID gain, and the critic network maximizes the total reward value.
- (4) According to the calculation formula for the self-optimizing PID controller, calculate the control commands.
- (5) Use the time series control commands to act on the controlled object, while simultaneously observing the state of the environment at the next moment and calculating the reward function value.
- (6) The actor network uses the DDPG algorithm to update its own weights. The critic network updates its weight, based on the mean squared error (MSE) loss function.
- (7) If the system performance indicators meet the given requirements, or the maximum number of run episodes is reached, the training is terminated, the execution process is exited, and the experiment state is reset.

An overview of the workflow and architecture for the efficient training of the algorithm is shown in Figure 6.

1: While True do

```

2:   Reference trajectory and physical model;
3:   Waiting for environment state initialization or reset;
4:   Set task flag and pre-set hyper parameters.
5:   if task flag = train then
6:     Run episode with DDPG algorithm;
7:     Maximize total reward;
8:     Output gain variable;
9:     Calculate self-optimizing control command.
10:  if episode is over set value then
11:    Save model and reset environment state.
12:  end if
13: else if task flag = test then
14:   Run episode with trained model;
15:   Verify model control performance.
16: else if task flag = undo then
17:   Exit execution process and reset experiment state.
18: end if
19: End while

```

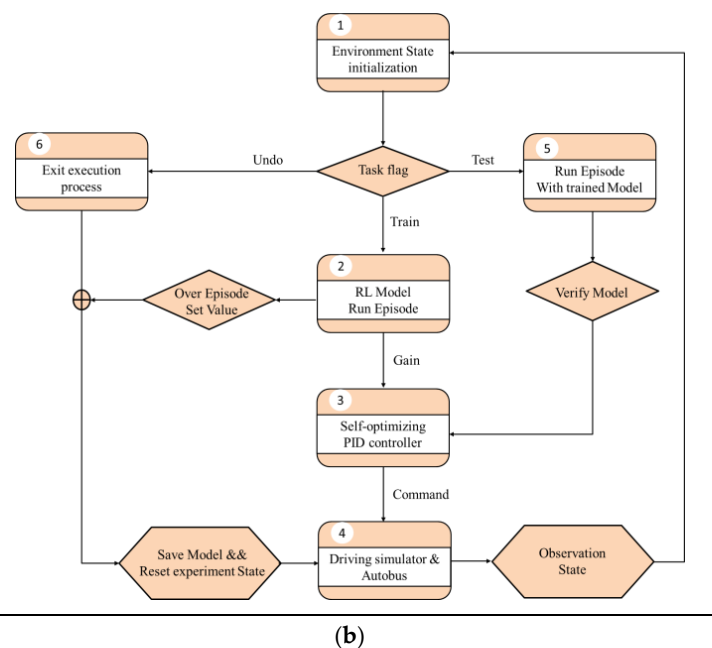


Figure 6. Outline of the workflow and the architecture used for efficiently training the algorithm. (a) Workflow for self-optimizing PID controller based on reinforcement learning. (b) Software execution architecture run episodes during model training or testing.

a. Actor–critic network architecture design

The objective of this research was to consider the path-following control problem as an optimal control problem for minimizing the lateral position deviation and lateral angle deviation of the controlled object from the reference trajectory. Summarizing the process in Figure 1, the self-optimizing PID controller in this study, based on the RL framework, mainly comprises two parts: an actor network and a critic network. The focus of the DDPG algorithm is on the design and optimization of the actor–critic network structure, with

the aim of finding the optimal control strategy. Through the method of RL, the control policy of the agent is updated to maximize the value of the reward function. The actor network is a strategy function that is responsible for generating actions and interacting with the environment. The critic network is a value function that is responsible for evaluating the performance of the actor and guiding the output of the actor network in the next stage. Theoretically, a neural network with only one hidden layer is sufficient to achieve a global approximation and a description of the arbitrary nonlinear functions. In the process of training the model, a fully connected actor neural network and critic neural network are initiated to approximate the optimal control policy and true value function. Figure 7 presents the architectures of the actor and critic networks [41]. Both consist of three layers: an input layer, an output layer, and a hidden layer, with 600 neurons.

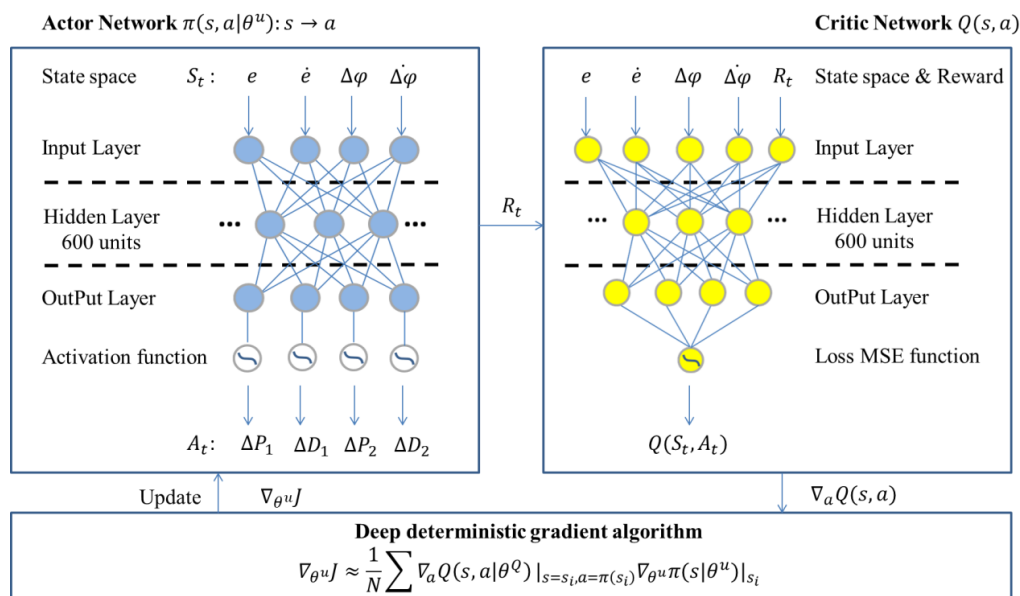


Figure 7. Architectures of the actor and critic networks.

The real-time environment state space data, $S_t = \{e, \dot{e}, \Delta\varphi, \dot{\Delta\varphi}\}$, from a virtual simulator and real-world autobus, are used as the original inputs to the deep RL, to solve the path tracking problem for autonomous vehicles. The actor network takes the preprocessed data as the input and connects with the fully connected layer; the sigmoid activation function is used to map the action directly to the range of $[-1, 1]$. The final output of the actor network represents the action space $A_t = \{\Delta K_{p1}, \Delta K_{d1}, \Delta K_{p2}, \Delta K_{d2}\}$. The critic network combines the reward value with the environment state as the input, connects through the fully connected layer, and finally outputs the Q-value. In this paper, the parameters of the networks were updated, based on the Adam [42] optimization algorithm. The DDPG algorithm [43] was employed to iteratively update the actor network weights, θ^u ; the critic network weights, θ^Q , were updated to minimize the MSE loss function.

The input to the actor network is the number of features in the state space, whereas the input to the critic network is the sum of the state features and rewards. Both networks have only one fully connected layer between the input and output layers, with 600 neural units. The adopted hyper-parameters (parameters set prior to the training process) are presented in Table 2.

Table 2. Actor–critic network structure hyper-parameters.

Hyper-Parameter	Pre-Set Value
Actor network learning rate	0.001
Critic network learning rate	0.01
State space dimension	4
Action space dimension	4
Discount factor	0.95
Run max episode	200,000

The real-time environment state space data is $S_t = \{e, \dot{e}, \Delta\varphi, \dot{\Delta\varphi}\}$, so that state space dimension's pre-set value is 4; the final output of the actor network represents the action space, $A_t = \{\Delta K_{p1}, \Delta K_{d1}, \Delta K_{p2}, \Delta K_{d2}\}$, so that action space dimension's pre-set value is 4. Here, γ is a discount factor, and is usually a constant close to 1; in this paper, the discount factor's pre-set value is 0.95. In fact, with a larger γ , the agent considers more steps forward, but the difficulty of training is higher; whereas, with a smaller γ , the agent pays more attention to the immediate benefits, and the training is less difficult. In short, the principle of the value of the discount factor is to be as large as possible on the premise that the algorithm can converge. The learning rate of the actor–critic network in this article adopts the same default value as in Reference [43], where the pre-set values are 0.001 and 0.01, respectively.

b. RL deep deterministic policy gradient (DDPG) algorithm

The main research objective of this paper was to design an actor–critic network with a DDPG algorithm to control the path tracking behaviors of autonomous vehicles and characterize the adaptive ability, through considering different reference road paths and designing the reward function. The process of automatic driving path tracking control requires an autonomous agent system to address the current environmental situation and vehicle status and then implement comprehensive lateral and longitudinal control; the adaptive ability of the controller is especially important under variable speeds and complex reference trajectories. Therefore, in order to address more complex scenarios, a self-optimizing PID path tracking method for intelligent vehicles, based on RL, is presented. This is a typical data-driven and self-learning method that enables an agent to find an optimal control strategy to complete tasks through continuous “trial and error”, while interacting with the environment and changes the action(s), based on a feedback reward system, based on the environment. The RL framework is used to solve practical engineering problems and can be described as an MDP [44–46].

In this paper, the environment state space and action space are continuous variables, so we define the tracking control problem as an MDP of the sequence data, which comprises a 5-tuple $(S, A, R(s_t, a_t), P(s_{t+1} | s_t, a_t), \gamma)$. As shown in Figure 8, S_t is the state space set and A_t is the action space set. At time step, t , the agent selects the action $a_t \in A_t$ by following policy π . After executing a_t , the agent is transferred to the next state, s_{t+1} , with the probability $P(s_{t+1} | s_t, a_t)$. Additionally, a reward signal, $R(s_t, a_t)$, is received to describe whether the underlying action, a_t , is beneficial for reaching the goal. By repeating this process, the agent interacts with the environment and obtains a sequence of trajectories, $\tau = s_1, a_1, r_1, \dots, s_T, a_T, r_T$, at the terminal time step, T . The discounted cumulative reward from each time step, t , can be formulated as $R_t = \sum_{t=1}^T \gamma^{t-1} r_t$, where $\gamma \in (0, 1)$ is a discount rate for determining the importance of future rewards. The goal is to learn an optimal policy, π^* , that maximizes the expected overall discounted reward under this strategy, which is defined as follows:

$$J = E_{s,a \sim \pi,r} \left[\sum_{t=1}^T \gamma^{t-1} r_t \right] \quad (22)$$

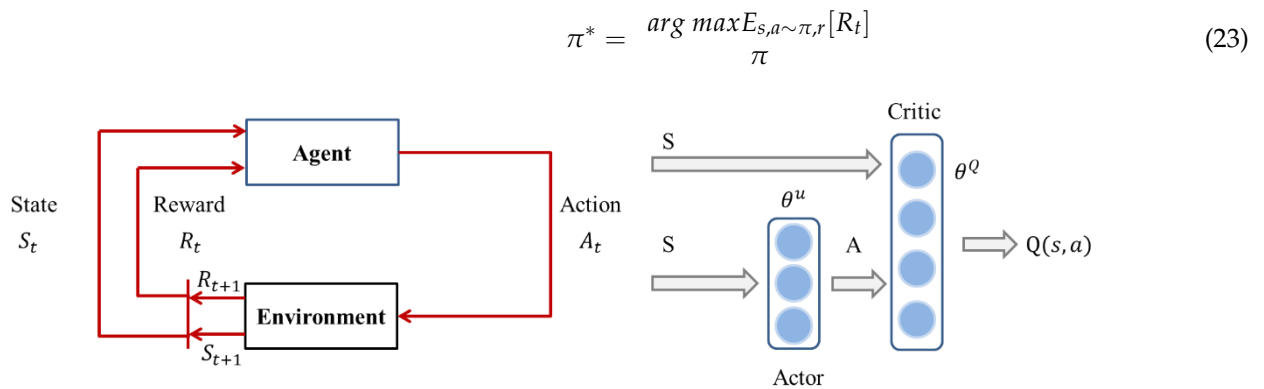


Figure 8. RL framework workflow diagram.

The framework of the actor–critic algorithm is based on the concept of the DDPG algorithm, which is widely used [47,48] and integrates the policy search and value function approximation theories. As illustrated in Figure 3b, the actor is used to adjust the network parameter, θ^u , and output determination action, A , based on the optimal control strategy, $\pi(s, a|\theta^u)$. The critic approximates the value function, $Q(s, a)$, and updates the network parameter, θ^Q . To iteratively update these neural network parameters until convergence in a near-optimal control policy, we employed the DDPG algorithm to iteratively update the actor network weights, θ^u . Additionally, the critic network weightings, θ^Q , were updated so as to minimize the MSE loss function. The updated calculations are as follows:

$$\nabla_{\theta^u} J \approx \frac{1}{N} \sum \nabla_a Q(s, a|\theta^Q) \Big|_{s=s_i, a=\pi(s_i)} \nabla_{\theta^u} \pi(s|\theta^u) \Big|_{s=s_i} \quad (24)$$

$$L(\theta^Q) = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2 \quad (25)$$

$$y_i = r_i + \gamma Q'(s_{i+1}, \pi'(s_{i+1}|\theta^u)|\theta^Q) \quad (26)$$

For the sake of achieving a more stable training process, a target actor neural network weighting parameter, $\theta^{u'}$, and target critic neural network parameter, $\theta^{Q'}$, were also initialized, and these were updated as follows:

$$\theta^{u'} \leftarrow \tau \theta^u + (1 - \tau) \theta^{u'} \quad (27)$$

$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'} \quad (28)$$

Here, τ is a hyperparameter that is pre-set to prevent the overfitting of these neural networks and to maintain the training stability. The pseudo-code programming process for the DDPG algorithm is presented in Algorithm 1.

The entire procedure is repeated until the optimal control policy is learned, and we use the same framework for both simulation and real-world experiments; the controller system learns basic path tracking skills to adapt to different reference trajectories and dynamic model parameters.

Algorithm 1. Pseudo-code programming process of the deterministic policy gradient (DDPG) algorithm

Actor uses a gradient algorithm to update the network parameters;

Critic uses the mean squared error (MSE) loss function to update the network parameters.

Algorithm input: Episode number, T; state dimension, n; action set, A; learning rate, α, β ; discount, γ ; exploration rate, τ ; actor-critic network structure; randomly initialize the weighting parameter.

Algorithm output: Actor network parameters, θ^u , critic network parameters, θ^Q .

```

1: for Episode from 1 to (Max Episode -1) do
2:   Receive initial observation state, obtain environment state vector  $s_t$ .
3:   Initialize buffer replay data-buff.
3:   for t from 1 to T do
4:     Select action  $a_t = \pi(s_t|\theta^u) + \mathcal{N}_t$ .
5:     Execute action  $a_t$  and observe new state  $s_{t+1}$ . Calculate instant reward feedback.  $r_t$ 
6:     Store transition  $\langle s_t, a_t, s_{t+1}, r_t \rangle$  in data-buff.
7:     Random mini-batch of N transitions  $\langle s_i, a_i, s_{i+1}, r_i \rangle$  from data-buff.
8:     Set  $y_i = r_i + \gamma \theta^Q(s_{i+1}, \pi'(s_{i+1}|\theta^u)|\theta^Q)$ .
9:     Update critic by minimizing MSE loss function:
10:    
$$L(\theta^Q) = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2.$$

11:    Update the actor policy using the policy gradient function:
12:    
$$\nabla_{\theta^u} J \approx \frac{1}{N} \sum \nabla_a Q(s, a|\theta^Q) \Big|_{s=s_i, a=\pi(s_i)} \nabla_{\theta^u} \pi(s|\theta^u) \Big|_{s=s_i}.$$

13:    Update the target networks:
14:    
$$\theta^{u'} \leftarrow \tau \theta^u + (1 - \tau) \theta^{u'}$$

15:    
$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$$

16:   End for time step
17: End for Episode
  
```

4. Experiment and Analysis of Results

This section describes the simulation environment, the path tracking controller training process, and the employment of the physical autonomous system, with a controller performance evaluation and a generalization ability verification. In the following, we describe each step in detail.

4.1. Experimental Setting

a. Simulation experiment platform

We conducted a hardware-in-the-loop test on a driving simulator to analyze the effectiveness of the proposed self-optimizing controller, based on RL. A schematic of the simulation experiment platform is shown in Figure 9.

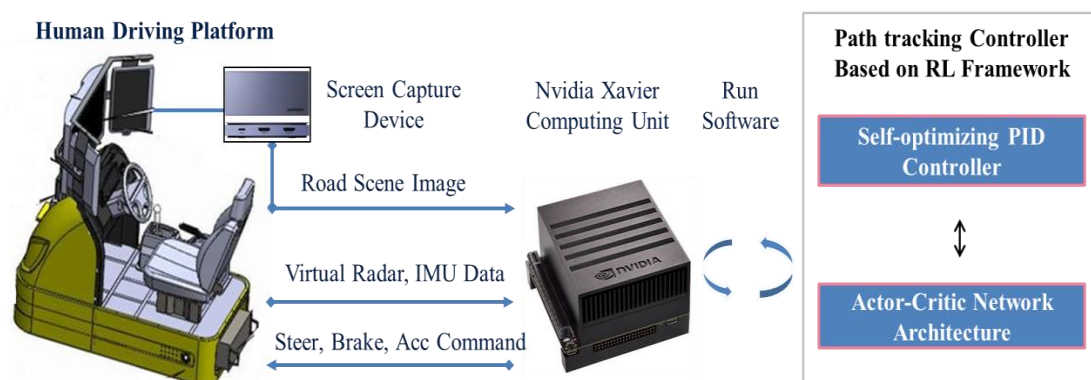


Figure 9. Hardware-in-the-loop simulation platform based on driving simulator.

To follow the desired driven path, we projected a set of trajectories onto our environment map to examine the performance of the presented controller. We selected the candidate that best minimized the lateral position deviation and lateral angle deviation of the controlled object from the reference trajectory. Thus, five urban road maps (Figure 10) with various levels of difficulty were designed for the self-optimizing path tracking controller, with reference to the tracks of the car racing games TORCS [19] and GT Sport [20]. These road maps were generated using the SCANer™ studio engine (OKTAL, France, see: www.oktal.fr, 13 December 2019), a road and environment creation software for automotive simulation. This software was responsible for delivering the raw sensor data to the control interface and for transferring the control commands (steer, brake, acceleration) to the simulator engine for execution through a dedicated application program interface function. The driving performance data were recorded at a frequency of 20 Hz.

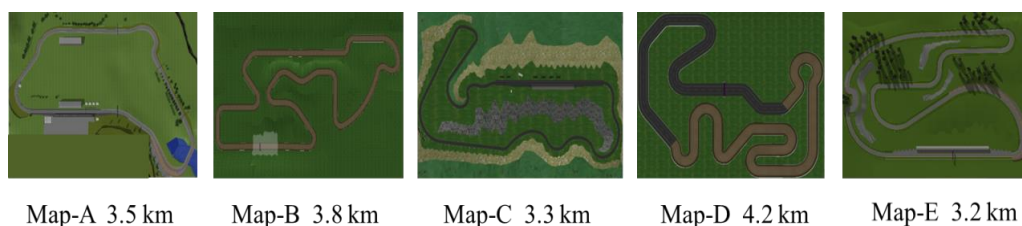


Figure 10. Trajectory virtual scene road maps.

For a specific traffic environment, we aimed to provide the path tracking controller with a reference trajectory to follow. We invited an experienced driver to operate a car with a steering wheel and pedals on the different urban road maps to generate the corresponding reference trajectories. The collected data included the vehicle world location, heading angles, and velocities in the x -direction, thereby providing environmental states for training and test evaluations, based on the vehicle sensor IMU information and GPS data. At every time step, the path tracking controller calculated the reference error based on the simplified vehicle models. The vehicle's location, relative to the specific reference coordinate system, was denoted as (x, y, φ) , where x and y were the coordinates of the midpoint of the vehicle's center of gravity and φ was the orientation angle of the vehicle's body.

b. Realistic autobus experiment platform

The realistic autobus experiment platform provided radar, GPS, and IMU data, and we could parse out obstacle distance and vehicle attitude information as well as genuine road indicator values. The autobus platform could also execute control commands (steer, brake, and acceleration) received from the path tracking controller through the vehicle's controller area network bus. The self-optimizing controller, based on the RL framework, ran on NVIDIA's computing unit Xavier and comprised two submodules. First, the actor-critic network architecture mode obtained the proportional and derivative gain values by training the network using the DDPG algorithm. Second, the self-optimizing PID controller module received the gain values and calculated the real-time control commands for acting on the vehicle steering wheel. A schematic diagram of the realistic autobus experiment platform is shown in Figure 11, and the function description and precision of each sensor are shown in Table 3.

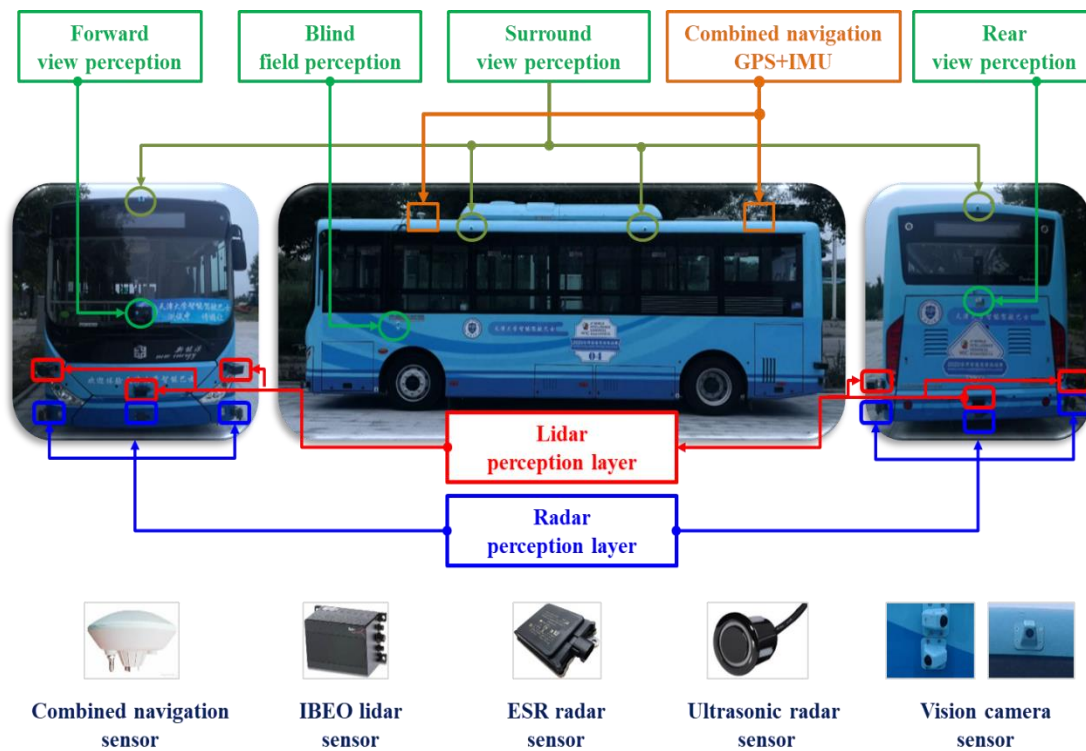


Figure 11. Schematic diagram of the realistic autobus experiment platform.

Table 3. Vehicle sensor configuration scheme. (* indicates the number of sensors).

Sensors	Position	Function Description	Precision
GPS+IMU *1	Top	Precise location of the vehicle.	Positioning accuracy: 5 cm
IBEO Lidar *6	Front, Rear	1. Vehicle, pedestrian detection. 2. Relative distance, speed, angle	Detection accuracy: 90% Effective distance: 80 m
ESR Radar *6	Front, Rear	1. Long-distance obstacle detection. 2. Road edge detection.	Detection accuracy: 90% Effective distance: 120 m
Vision Camera *12	Front, Rear Top sides	1. Traffic light status detection. 2. Lane line detection.	Detection accuracy: 95% Effective angle: 178°
Ultrasonic radar *8	Front, Rear, Both sides	1. Short-distance obstacle detection. 2. Blind field detection.	Detection accuracy: 90% 360° coverage

Our real-world driving experiment mimicked those conducted in simulations in many ways. However, executing this experiment in the real world was significantly more challenging, as the system could not automatically reset the starting state. In addition, the RL agent evaluates and improves its control policy in a trial-and-error manner; thus, it would have been dangerous and costly to train an agent on a real vehicle; moreover, particularly with dynamic balancing systems, such an approach would be complicated and expensive. We were motivated by the steady ability of the traditional PID controller and learning mechanisms that interacted with the environment. As noted above, in this paper, a self-optimizing PID path tracking controller, based on an RL framework, was proposed for use with a realistic autonomous platform. The design of the controller had three advantages, as mentioned above (reducing the exploration space of the RL, introducing an online self-learning mechanism into the traditional controller design, and using RL for practical engineering control). To the best of our knowledge, this was the first demonstration of a deep RL agent driving a real autobus. We conducted our experiment using a wire-controlled autobus (“New Energy Electric Bus”) (see Table 4 for the specific vehicle parameters).


Table 4. Specific parameter information of the wire-controlled autobus.

Vehicle Information Parameters			
Length (mm)	8010	Maximum Total Mass (kg)	13000
Width (mm)	2390	Front Suspension/Rear Suspension (mm)	1820/1690
Height (mm)	3090	Approach Angle/Departure Angle (°)	8/12
Wheelbase (mm)	4500	Maximum Speed (km/h)	69
Turning Radius (mm)	9000	Tire Size × Number	245/70R19.5 × 4

c. Software version and hardware computing platform

In this project, we used NVIDIA’s Jetson AGX Xavier computing module to run software algorithms on virtual and realistic autonomous platforms. Thus, it was possible to implement an autonomous machine domain controller using artificial intelligence (AI) technology, which was sufficient for completing the following tasks: sensor fusion, high-precision positioning, path planning, and executing tracking algorithms. The kit benefited from NVIDIA’s rich set of AI tools and workflows, which can be used to quickly train and deploy neural networks. Table 5 presents the path tracking controller software and computing hardware environments that the agents relied on in the training and testing processes.

Table 5. Software and hardware technical parameters.

Software and Hardware Technical Parameters of the On-Board Computing Unit		
	GPU	512-core Volta GPU with Tensor Core
	CPU	8-core ARM 64-bit CPU
	RAM	32 GB
	Compute DL-TOPs	30 TOPs
	Operating system	Ubuntu 18.04
	RL framework	Tensorflow-1.14

4.2. Performance Verification and Results Analysis

a. Simulation experiment setup and performance during training process

We trained our path tracking controller on four maps (Figure 10 Map-A~Map-D). Map-A was relatively simple and was used for the first stage of training, in which the vehicle learned a basic reference trajectory-tracking task, such as on long straight roads and/or some simple corners. Map-B, Map-C, and Map-D had different levels of difficulty, with diverse corner shapes. Map-E had the most complicated trajectory and was used for testing based on the pre-trained weights from Map-A to Map-D to evaluate the control performance and generalization ability of the controller. The training rewards of the different tracks, based on the RL controller, are illustrated in Figure 12.

In the path tracking control experiments, we trained an optimal policy to achieve continuous action control in the simulation platform. During the entire training process, the ego vehicle (also referred to as host vehicle) was driven at a fixed speed of 30 km/h and the experimental frequency was 20 Hz. If the vehicle was driven out of the lane or collided and/or the vehicle speed dropped to 0, we penalized the model, and the current episode was terminated. In Figure 12, which illustrates the total rewards against the number of episodes, we can see that, as the training continues, the total reward in one episode increases, because the model gradually finds the optimal control policy. In addition, the complexity of the reference trajectory also directly affects the training time and number of episodes. With the goal of completing a lap driving task, we recorded the number of episodes, iteration steps, driving distances, and training times, and the results are shown in Table 6.

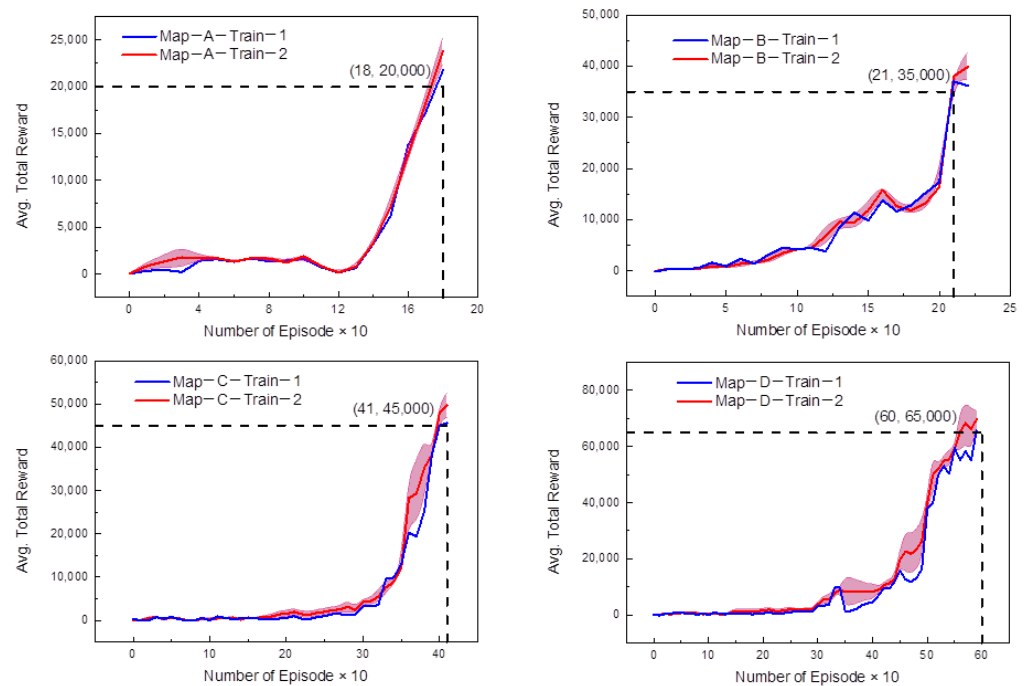


Figure 12. Average total reward value with the RL training episode.

Table 6. Reinforcement learning (RL) training result statistics.

	Number Episode	Iteration Step	Drive Distance	Training Time
Map-A	180	8858	2387.64	0.75 h
Map-B	210	16,139	3242.83	1.2 h
Map-C	410	38,926	2935.72	2.3 h
Map-D	600	61,538	6470.38	3.6 h

During this study, although the RL agent learned the optimal control strategy, the training process took a very long time; the training time reached 3.6 h during the training process of Map-D, and the numbers of episodes and iteration steps reached 600 and 61,538, respectively. The actor–critic neural network sometimes did not converge (overfitting), causing the reward to drop sharply; the pink band represents the standard deviation of the total reward, which fluctuated greatly during the training process on the roads of Map-C and Map-D. For the training result of MSE, the loss value on the road of Map-D illustrates that, at the 140th iteration step, the value converged to 4500, as shown in Figure 13.

The possible reasons for these results include the fact that the essence of the DDPG algorithm is a trial-and-error method, which is based on random sampling during the training process; thus, it can easily encounter excessive training time and overfitting problems. The key is to properly balance exploration and utilization. In this paper, the design of the controller reduces the exploration space of the RL to find the optimal control parameters, which will greatly improve the learning efficiency. The training rewards for the self-optimizing path tracking controller, based on the RL framework, are illustrated in Figure 14.

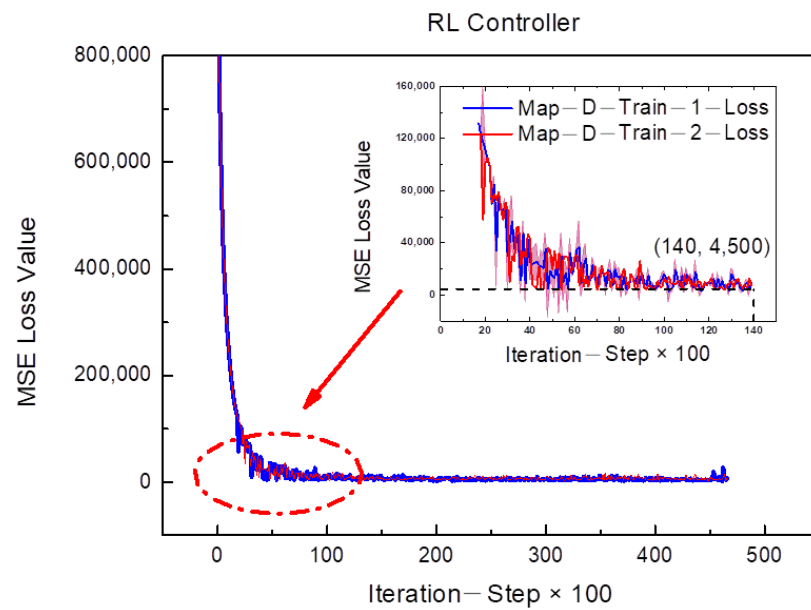


Figure 13. Mean squared error (MSE) loss value with the RL training step number.

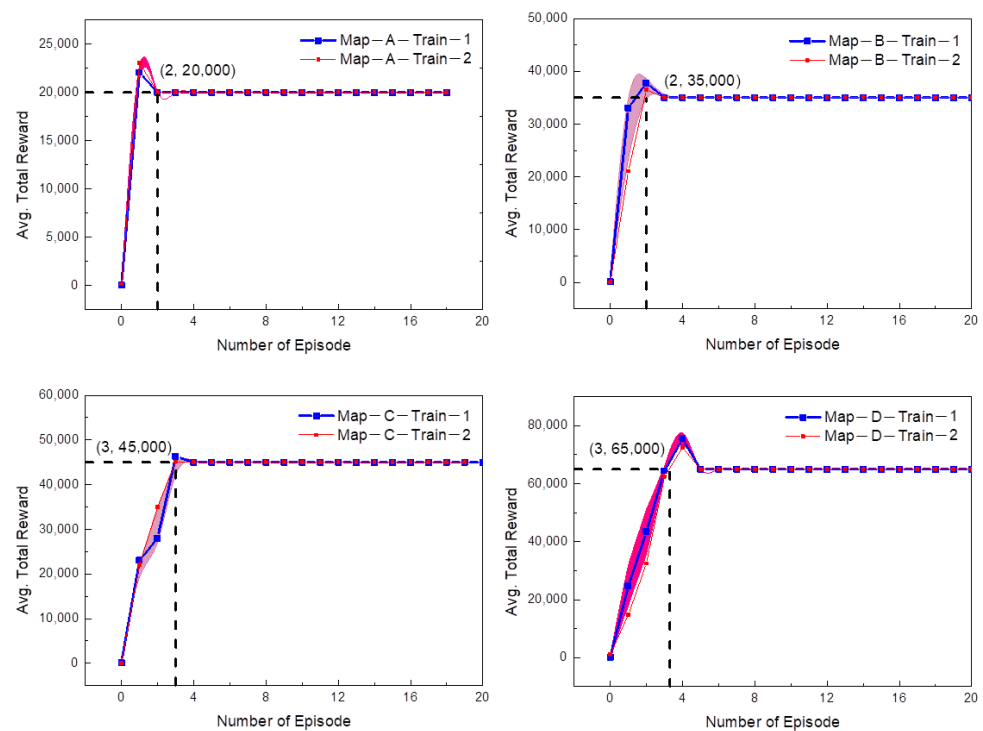


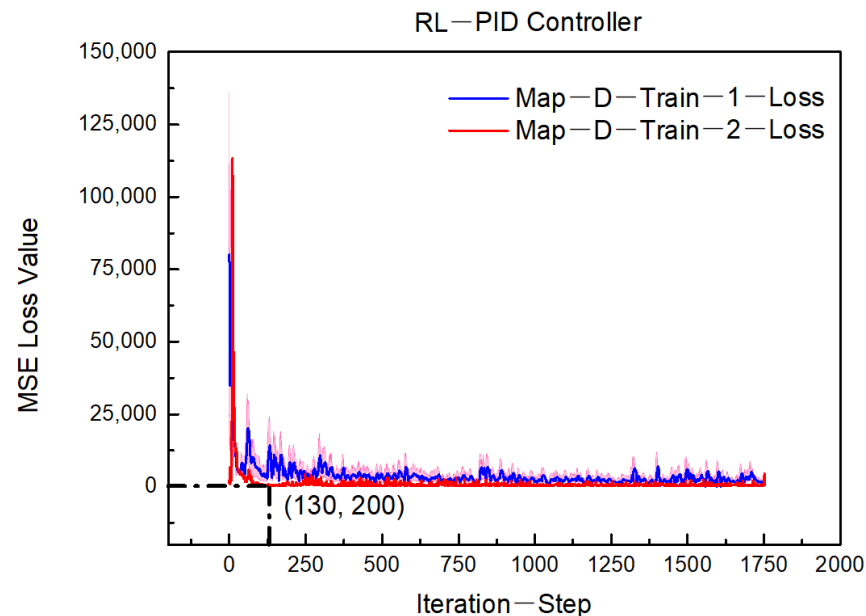
Figure 14. Average total reward value with the PID-RL training episode.

As training continues, the total reward in one episode increases linearly. Owing to the introduction of the traditional PID control parameters (as constrained by experience knowledge), the exploration space for the RL is greatly reduced, meaning that the optimal control policy can be learned quickly. Based on the goal of completing a lap driving task, the results for the number of episodes, iteration steps, driving distances, and training times are shown in Table 7.

Table 7. RL proportional–integral–derivative (PID–RL) training result statistics.

	Number Episode	Iteration Step	Drive Distance	Training Time
Map-A	2	3858	2987.4	9.8 min
Map-B	2	5139	3642.3	11.2 min
Map-C	3	6926	4732.8	13.1 min
Map-D	3	8738	6870.2	18.3 min

We can see that the self-optimizing PID path controller, based on RL (online framework), enables us to obtain an acceptable control policy quickly. The ego vehicle can pass Map-A and Map-B at the 2nd episode and Map-C and Map-D at the 3rd episode, respectively. Simultaneously, it effectively solves the problem of network overfitting, allowing for stable convergence (as shown in Figure 15). With regard to the training result for the loss value in Map-D, the result illustrates that, at the 130th iteration step, the network converges to 200. After 130 iterations, a highly effective path tracking control policy will have been learned.

**Figure 15.** MSE loss value with the PID–RL training step number.

Based on a comparative analysis of the above results (Figure 12 vs. Figure 14, Figure 13 vs. Figure 15, Table 6 vs. Table 7), we can verify that the control policy, as constrained by prior experience, can help the RL agent learn relatively quickly.

b. Evaluating the performance of self-optimizing proportional–integral–derivative (PID) controller, based on RL framework

The purpose of the evaluation is to verify the adaptive ability of the proposed controller algorithm under complex reference trajectories and various driving speeds, as well as different dynamic models. Additionally, for the further analysis of the proposed controller, reference results from an experienced driver are presented for comparison in order to verify whether the self-optimizing controller can learn a better control policy. This paper adopted five indicators for measuring the performance of the self-optimizing path tracking controller:

- **The smoothness** indicator represents the comfort resulting from the path-following control. In this paper, the vibration amplitude of the steering wheel was used to represent the smoothness indicator.
- **The lateral track error, e , and heading angle error, $\Delta\varphi$,** evaluate the effects of the path tracking.

- **The maximum speed and average speed** indicators characterize the driving efficiency.

A traditional PID controller uses a more intuitive steering control law, where RL is a more advanced self-optimal learning controller. Hence, in this study, the self-optimizing path tracking controller proposed is based on the RL framework and combines traditional PID control algorithms and RL mechanisms. The figure below shows a performance comparison between a fixed-parameter PID controller, an RL controller, and the proposed PID–RL self-optimizing controller, in a path-following control process at 30 km/h. It can be observed that the self-optimizing controller learns a better control policy.

From Figure 16 and Table 8, it can be seen that the amplitude (min–max), mean, and standard deviation are all reduced with the self-optimizing path controller, which can quickly realize stable control and overcome the overshoots caused by PID control and the unstable characteristics of the RL controller. In particular, the standard deviation value is significantly reduced, indicating the smoothness of the steering wheel rotation and the driving comfort.

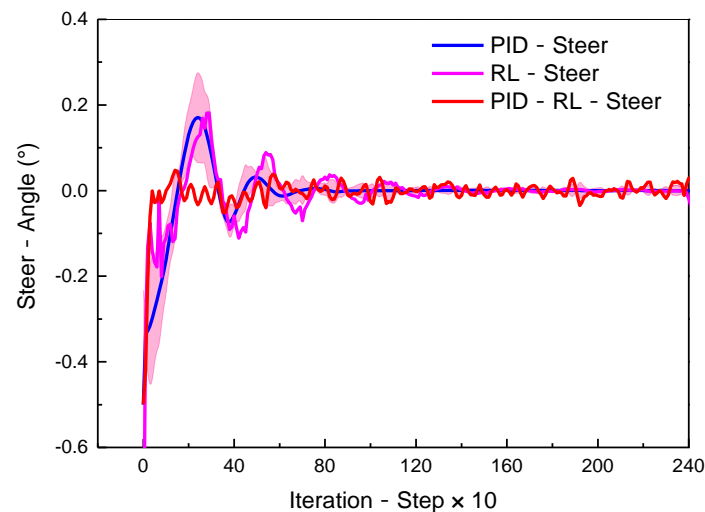


Figure 16. Steer angle of the path tracking controller with the iteration step.

Table 8. Mathematical statistics of steering wheel angles.

	Standard Deviation	Minimum	Maximum
PID–Steer	0.11785	−0.5	0.17068
RL–Steer	0.13907	−0.96124	0.18131
PID–RL–Steer	0.04705 ↓	−0.49881 ↓	0.07665 ↓

From Figure 17 and Table 9, it can be seen that the standard deviations of the lateral error for the two controllers are almost identical, whereas the amplitude of the self-optimizing path controller is the lowest, indicating that its control performance is more stable than that of the other two controllers.

Table 9. Mathematical statistics of lateral errors.

	Standard Deviation	Minimum	Maximum
PID–cross–track error (CTE)	0.1616	−0.12305	0.33338
RL–CTE	0.1220	−0.17023	0.34481
PID–RL–CTE	0.0915 ↓	−0.0092 ↓	0.29207 ↓

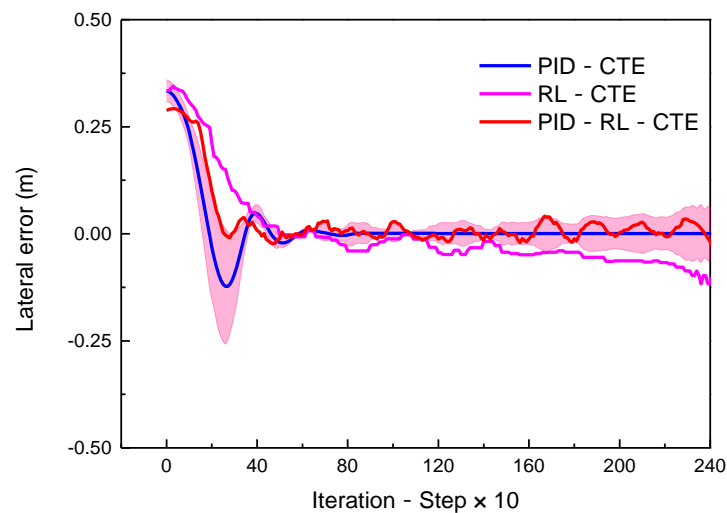


Figure 17. Lateral error of path tracking controller with the iteration step.

Another important evaluation indicator for path tracking is the heading angle error; from Figure 18 and Table 10, it can be seen that the standard deviation value is significantly reduced with the self-optimizing path controller. Considering the smoothness indicator, the lateral track error, e , and the heading angle error, $\Delta\varphi$, the controller performance can be expressed as follows.

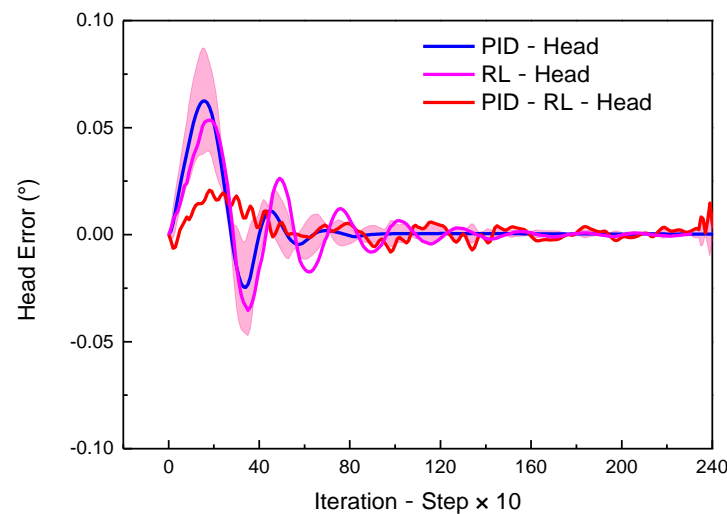


Figure 18. Heading error of path tracking controller with the iteration step.

Table 10. Mathematical statistics of heading errors.

	Standard Deviation	Minimum	Maximum
PID-Head	0.0343	−0.0247	0.0625
RL-Head	0.0349	−0.0353	0.0534
PID-RL-Head	0.0073 ↓	−0.0061 ↓	0.0208 ↓

Another important evaluation indicator for path tracking is the heading angle error; from Figure 18 and Table 10, it can be seen that the standard deviation value is significantly reduced with the self-optimizing path controller. Considering the smoothness indicator, the lateral track error, e , and the heading angle error, $\Delta\varphi$, the controller performance can be expressed as follows:

self-optimizing controller > PID controller > RL controller

The self-optimizing PID controller based on the RL framework can automatically adjust the control parameters to realize the dynamic compensation of the control system online and ultimately obtain a better path tracking control performance. The box chart distribution and mathematical statistics of the four control parameters are shown in Figure 19 and Table 11, respectively.

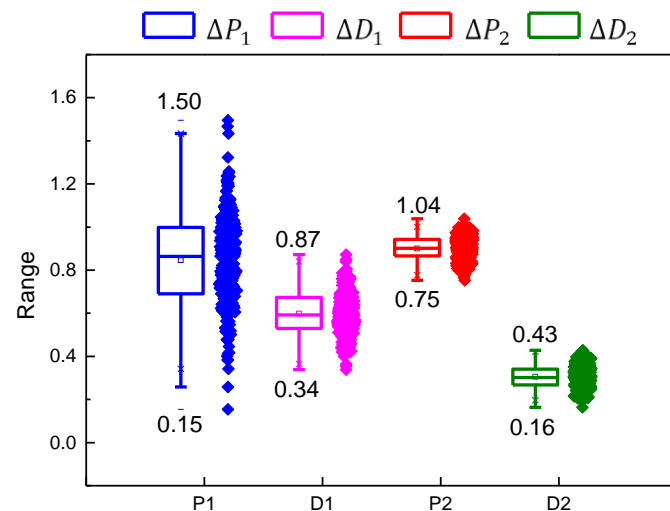


Figure 19. Data distribution of path tracking control parameters.

Table 11. Mathematical statistics of path tracking control parameters.

	Mean	Standard Deviation	Sum	Minimum	Median	Maximum
ΔP_1	0.84623	0.21836	206.48024	0.15364	0.86328	1.49518
ΔD_1	0.59826	0.10214	145.97583	0.33797	0.59185	0.87246
ΔP_2	0.90078	0.05179	219.78983	0.75222	0.90156	1.0384
ΔD_2	0.30479	0.05151	74.36867	0.16412	0.30233	0.42815

According to the fluctuation results for the standard deviation, the lateral deviation and its rate of change reached 0.2183 and 0.10214, respectively. It can be seen that when the actual trajectory is far away from the reference trajectory, the proportional adjustment of the lateral control parameters is the main factor. When approaching the reference trajectory, the heading angle deviation and its rate of change have a greater impact, with average values of 0.90078 and 0.30479, respectively. The controller can realize the self-optimization tuning of the PID control parameters based on the RL algorithm, which can then be used for online learning and the optimization of complex path tracking control.

It can be seen from the experimental results that RL can automatically optimize the PID control parameters according to the objective reward function and offer real-time online learning capabilities, providing a new solution to controller optimization problems of complex and uncertain systems.

To further verify the dynamic compensation performance of the self-optimizing controller, based on the RL framework, its performance is compared with that of an active disturbance rejection controller (ADRC). As described in References [49–51], the ADRC controller effectively alleviates the problem of vehicle jitter caused by road curvature changes under the conditions of a complex trajectory by observing internal and external disturbances of the system. This paper compared the lateral track error, e , of the two controllers on Map-E, and the speed conditions were set to 50 km/h and 60 km/h, respectively. The comparison results for the two controllers on the four corner types are shown in Figures 20 and 21, and the statistical analysis results are shown in Table 12.

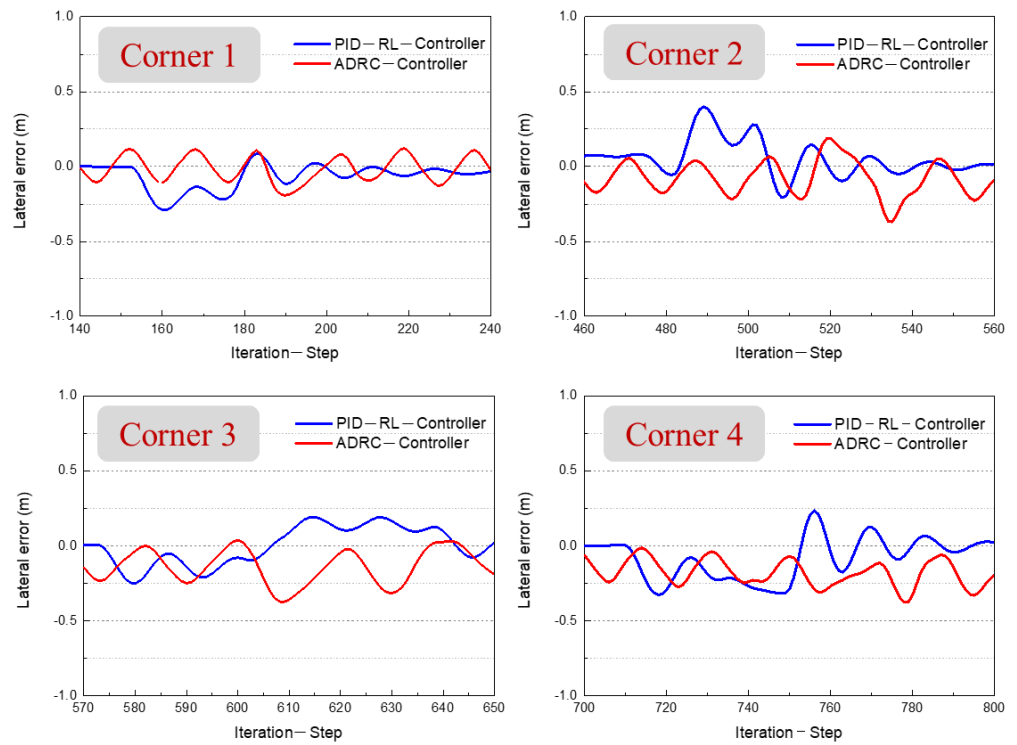


Figure 20. Self-optimizing controller and ADRC testing on road of Map-E under 50 km/h driving conditions.

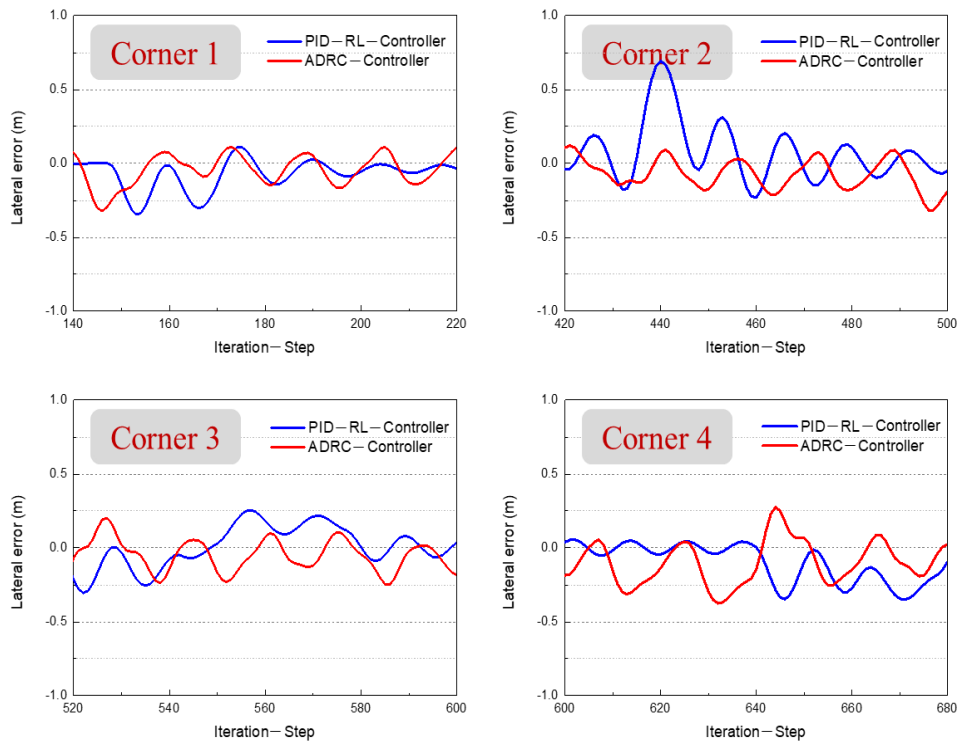


Figure 21. Self-optimizing controller and ADRC testing on road of Map-E under 60 km/h driving conditions.

Table 12. Mathematical statistics of lateral error for self-optimizing controller and ADRC controller on the road of Map-E.

50 km/h Driving Condition on the Road of Map-E				
	Mean	Standard Deviation	Minimum	Maximum
PID-RL Controller	-0.01017 ↓	0.09325	-0.32759	0.39932 ↑
ADRC-Controller	-0.06622	0.10941	-0.37658	0.18714
60 km/h Driving Condition on Road of Map-E				
	Mean	Standard Deviation	Minimum	Maximum
PID-RL Controller	-0.00823 ↓	0.10994	-0.35013	0.6918 ↑
ADRC-Controller	-0.04492	0.10508	-0.37490	0.27655

From the results, we can conclude that the path tracking performances of the self-optimizing controller and ADRC controller are almost unanimous. As shown in Table 12, both controllers can achieve a stable control performance under 50 km/h and 60 km/h driving conditions in terms of the standard deviation value of the lateral error. The ADRC controller observes the disturbances during the operation of the vehicle (see Figure 22), whereas the controller proposed in this study uses real-time online adjustment of controller parameters to achieve dynamic compensation during the path tracking control process (see Figure 23).

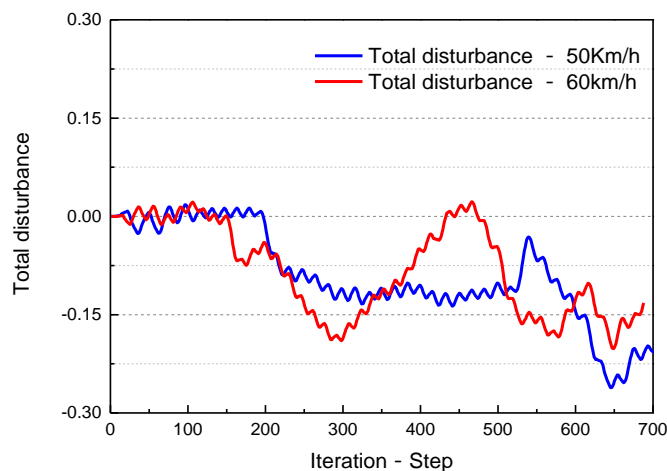


Figure 22. Total disturbance observed by ADRC.

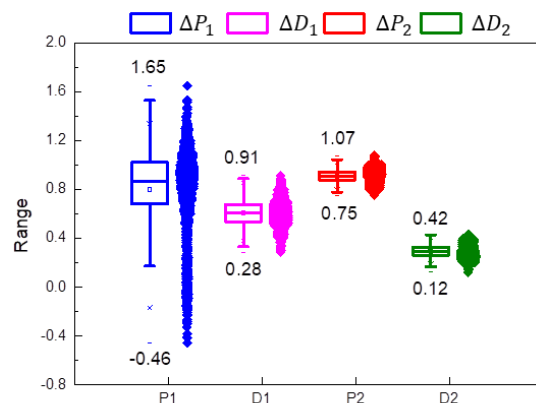


Figure 23. Distribution of control parameter.

c. Generalization of self-optimizing PID controller based on the RL framework

To test the generalization ability of the proposed self-optimizing PID controller, based on the RL framework, we evaluated it with complex trajectories and variable speed conditions on the road in Map-E. The different corner track types are shown in Figure 24 and include an arc curve track, a right-angle curve track, an S-curve track, and a U-turn track.

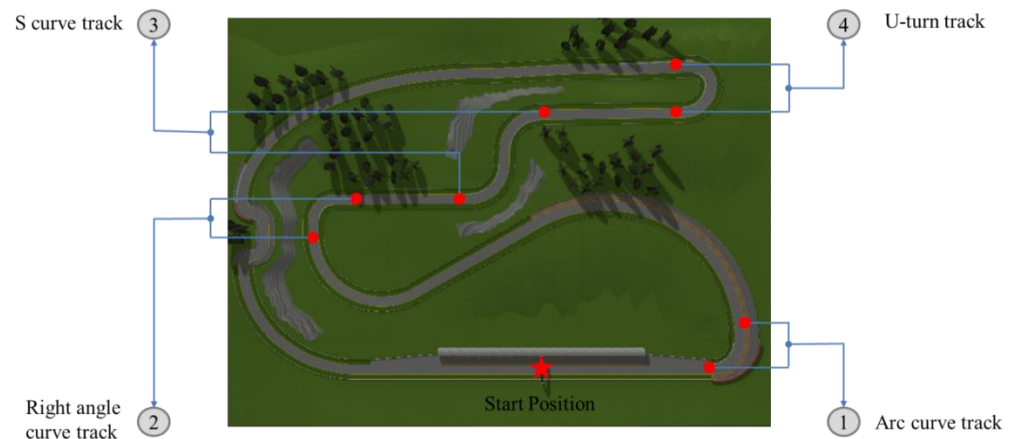


Figure 24. Complex corner track types of the road of Map-E.

The self-optimizing path tracking controller is further tested under variable speed conditions on the road of Map-E to evaluate whether the proposed controller can be implemented in a complex corner scenario at a high speed, based on using the behavioral data of professional drivers as a baseline for comparative analysis.

From Figures 25–27, and the analysis of the statistical results in Table 13, it can be seen that the max absolute value of the PID controller’s lateral error reached 1.11 m, the max driving speed was 54 km/h, and the average cornering speed was concentrated in the range of 36–42 km/h (as shown in Figure 25). The analysis results show that the controller with constant control parameters found it difficult to adapt to speed changes and large curvature road trajectories. As for the self-optimizing controller, the above problems can be overcome; the max absolute value of the PID–RL Controller’s lateral error was 0.66m, the max driving speed was more than 100 km/h, and the average cornering speed was concentrated in the range of 63–76 km/h (as shown in Figure 26). Furthermore, we conducted a comparative analysis with the human driver’s behavior data, the max absolute value of the human driver’s lateral error was 0.61 m, the lateral path track error of the method proposed in this paper is almost consistent with it. The maximum driving speed was 71 km/h; to avoid leaving the curve track, the drivers were forced to brake early before entering the corner, and the average cornering speed was concentrated in the range of 44–53 km/h (as shown in Figure 27), while the average cornering speed of the self-optimizing controller was better than that of the human driver.

Table 13. Statistical results of the generalization ability test of the self-optimizing controller.

	Indicators	Mean	Standard Deviation	Minimum	Maximum
PID Controller	Speed	41.0455	8.4158	0	55
	Lateral error	−0.1030	0.3573	−1.1125	0.9966
PID–RL Controller	Speed	84.6887	16.9462	0	101
	Lateral error	−0.0137	0.1089	−0.3844	0.6640
Human Driver	Speed	51.5378	11.8857	0	71
	Lateral error	−0.0014	0.1135	−0.6068	0.5521

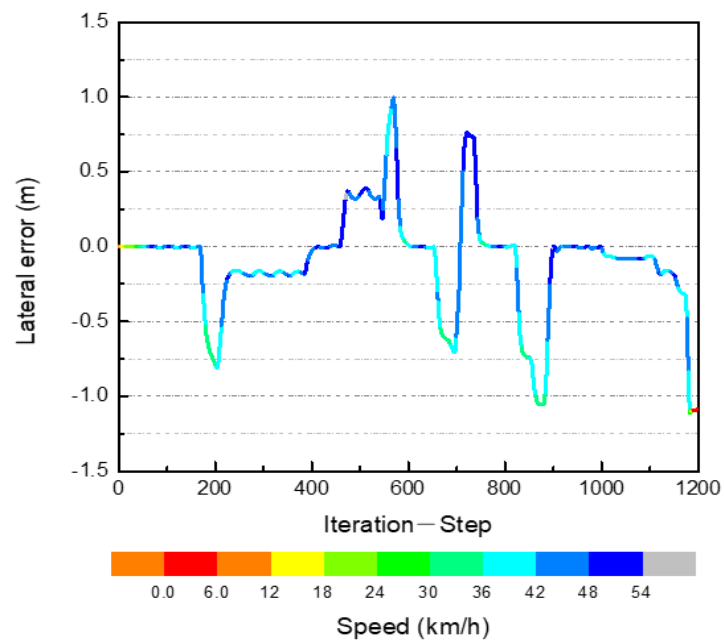


Figure 25. Lateral error of the traditional PID controller on the road of Map-E.

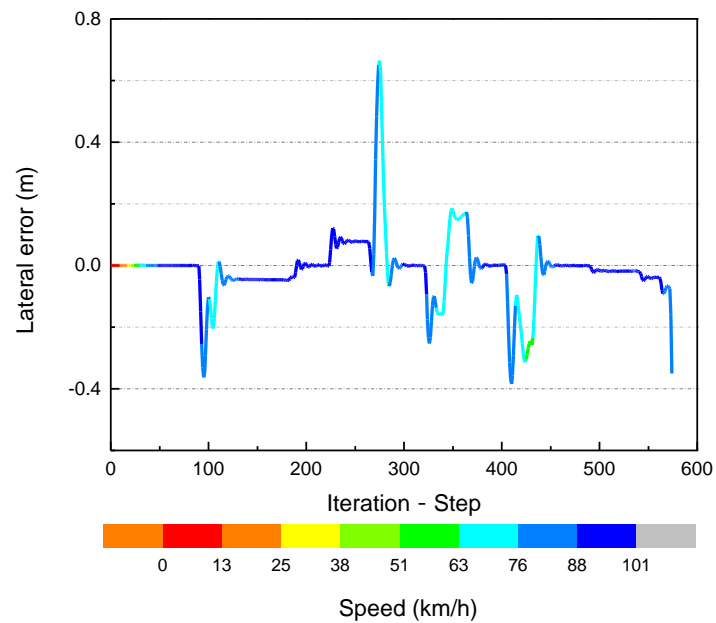


Figure 26. Lateral error of PID-RL Controller on the road of Map-E.

In summary, we can draw the conclusion that our proposed path tracking approach has the adaptability to cope with complex trajectory conditions and variation of speed by optimizing the PID controller parameters in real time.

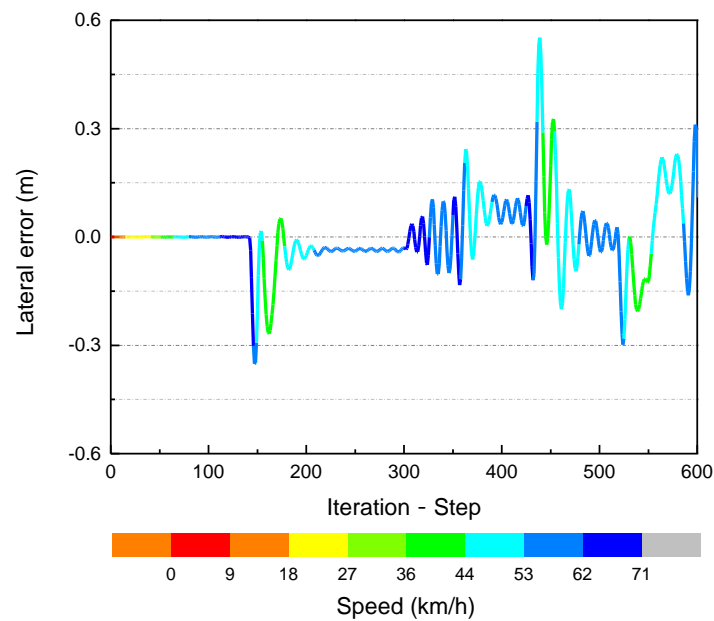


Figure 27. Lateral error of human-driver control on the road of Map-E.

d. Steering of a realistic autobus platform, based on the self-optimizing PID controller

Considering the achievements of self-optimizing controllers, based on the RL framework in the simulation environment, a natural question is whether these learned control policies can be deployed in real physical systems. The essence of the RL algorithm is a trial-and-error method, based on random sampling during the training process; thus, it would be dangerous to train an agent on a real vehicle. Furthermore, unlike the training process of a control policy in a simulated environment, the initial state of the controlled object cannot reset the state automatically between episodes in a real environment. The authors of [25] required a human driver to reset the vehicle to a valid starting position and initial state when the training episode terminated; however, this requires significant human labor costs. In summary, the simulation-to-reality transfer is hindered by the reality gap, in terms of how to effectively reset the initial state to ensure the stability of the control system, and how to reduce the time spent during the training process. To solve the above-mentioned problems, our real-world driving experiments imitated (in many aspects) those conducted in simulations and in steering realistic autobuses, based on a self-optimizing PID controller. This section describes in detail the deployment of the physically realistic autobus system and training process. An overview of our training method is presented in Figure 28.

For both the simulations and the real-world experiments, we realized the symmetric migration from a virtual simulation scene to a real vehicle platform, which resolved the limitation that reinforcement learning can only be used in simulation scene. We used the same actor–critic architecture and the same hyper-parameters that were found to be effective in the simulation. The common training procedures required adjustments in order to be deployed for a RL algorithm on a physical vehicle, running in a real-world environment. To account for both effectively resetting the initial state to ensure the stability of the control system and reducing the time spent during the training process, we created an architecture for the training procedures, comprising a simple state machine, as presented in Figure 28. It included five sub-modules: state initialization, model training, state automatic reset, driver takeover, and training task termination.

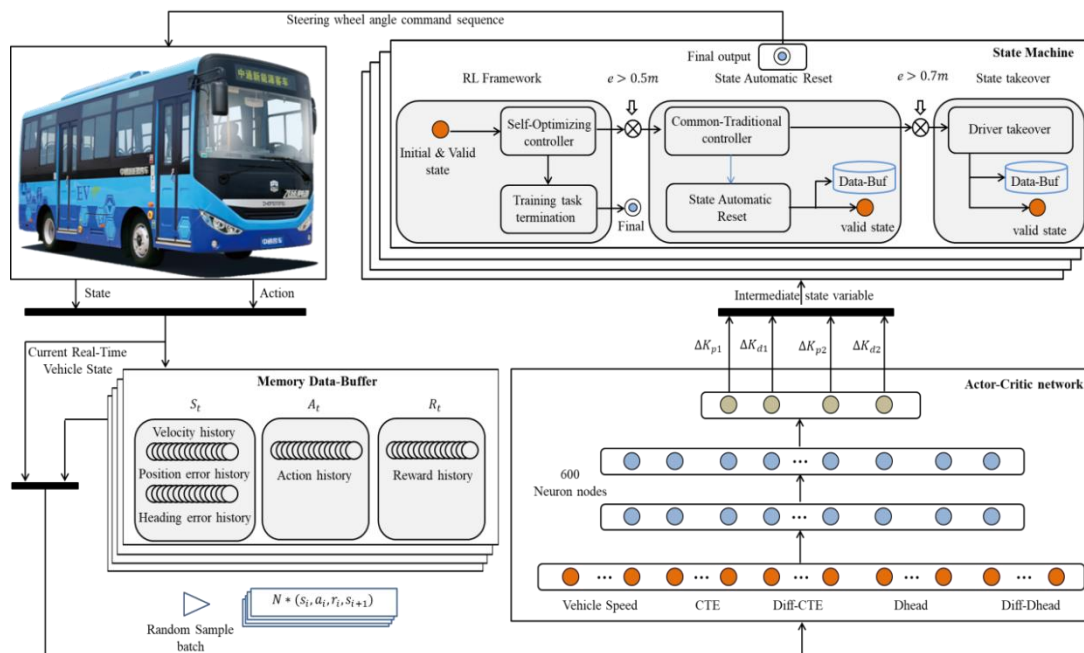


Figure 28. Architecture diagram of self-optimizing path tracking controller on a realistic autobus platform.

In fact, many environmental factors affect the training process; therefore, real-time safety and state machine monitoring mechanisms must be implemented in a physical vehicle control system. For these experiments, the vehicle was initialized at the starting position of the road during preparation for training. However, when the distance of the car from the center of the lane reached 0.5 m, the training episode was terminated and the process entered the state automatic reset module; simultaneously, the common traditional controller was used to control the vehicle, to revert to a valid state. At this point, the training process was executed. When the autobus deviated from the center line of the lane over a pre-set value ($e > 0.7$ m) and entered an unrecoverable state, a safety driver took over and steered the vehicle to return to the center of the lane—that is, to the valid state. Then, the next episode of training was begun. The introduction of the state machine effectively reset the initial state to ensure the stability of the control system.

In addition, we built a cache buffer of driver's behavior data. During the driver's driving process, the controller symmetrically learned the controllability of the human driver, in order to obtain the optimal control parameters. The memory data buffer was used to record the historical state and the action information of the vehicle. During the training process, random batches of N historical data were sampled for the online training of the network, and the actor-critic network mapped the vehicle state history to the intermediate state variable, which was used to calculate the increment of the steering wheel control sequence. Notably, if the data buffer stored positive sample data—that is, experience data from excellent human drivers—then it would contribute to the actor-critic network by quickly learning effective control policies, thus reducing the time spent on network training.

In general, the proposed self-optimizing controller learns the control policy by directly interacting with a realistic vehicle operating environment. The observation state space used by our method should be directly observable on a real vehicle equipped with sensors. The GPS allows for the determination of its relative coordinates, based on the current track of the road, and calculates the current lateral deviation error of the autobus from the given track. The heading angle deviation is obtained using the IMU sensor, which measures the difference in the change in the yaw angle with respect to the road track curvature. The generated intermediate quantity is the gain parameter of the self-optimizing controller, and its final output is the steering wheel angle control sequence data expanded over time, which directly acts on the autonomous platform. It should be emphasized that the final output

calculation result depends on the working mode selected by the state machine, as discussed in the second section of this paper. The calculation method used for the self-optimizing controller, based on the RL framework, is shown in Equation (18); whereas, the calculation method for the common traditional controller, based on the PID paradigm, is shown in Equation (16).

The path tracking task description follows a given reference trajectory by controlling the steering angle of the steering wheel. The sensory inputs are the pose information of the autobus (provided by the GPS and IMU systems) and the vehicle's speed. The path tracking controller output is the desired angle of the steering wheel in the range of $\pm 620^\circ$. The controller acts at 20 Hz, corresponding to a control interval of 50 ms. The autobus' drive-by-wire system will automatically disengage if the safety driver takes over, either by lightly stepping on the accelerator or brake pedal or by turning the steering wheel.

The path tracking controller of the vehicle system was tested on an autobus with different types of trajectories. Figure 29 is a screenshot from Google Maps, showing the curve case, the straight case, the round island case, the corner angle case, and the lane change case, etc. These were driven at speeds of 8 km/h and 10 km/h, with step speeds of 0–20 km/h. The desired trajectory was reconstructed from the data points with a 0.5 m spacing, as recorded from the GPS+IMU sensor. The 2-DOF bicycle model in vehicle dynamics was used to describe the basic motion law of the intelligent autonomous system, and the desired steering angle was determined by calculating the lateral error and the heading angle error, according to Equations (16) and (18).

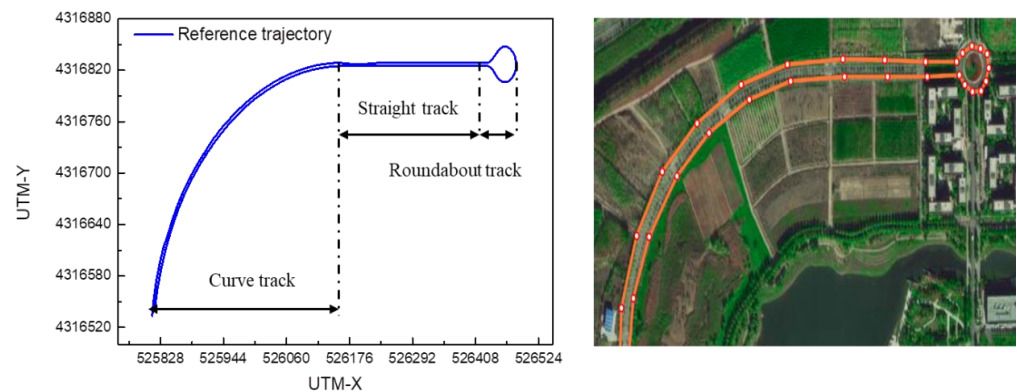


Figure 29. Schematic diagram of the test road map in the real world.

The evaluation indicators were the same as those tested on the simulation platform, to verify the path tracking control performance of the self-optimizing controller on a realistic physical vehicle platform. The resulting self-optimizing controller, based on the RL framework, showed much better behavior than the traditional controller—e.g., with respect to smoother steering wheel control sequence data, as illustrated in Figure 30.

From the experimental results in Figure 30 and Table 14, it can clearly be concluded that the human driver can complete the lateral motion control of the vehicle with a lower vibration amplitude (69.0577) in terms of the performance of the smoothness of the steering wheel control. Compared with the PID controller, the self-optimized RL controller can obtain better control stability: the vibration amplitudes of the steering wheels were 88.8032 and 80.6986, respectively. The above results verify the effectiveness of the self-optimizing controller, based on reinforcement learning, in real vehicles.

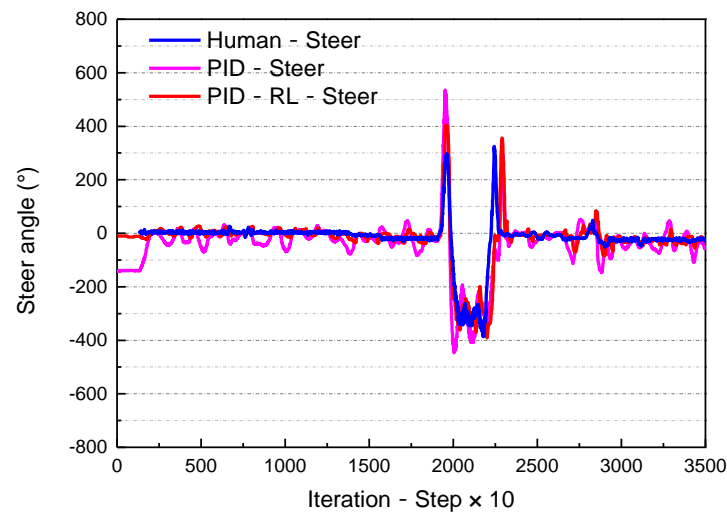


Figure 30. Steer angle of the path tracking controller with the iteration step.

Table 14. Mathematical statistics of steering wheel angles.

	Mean	Standard Deviation	Minimum	Maximum
Human—steer	−22.15445	69.05777	−386	325
PID—Steer	−42.81777	88.80323	−447	536
PID—RL—Steer	−22.06958 ↓	80.69866 ↓	−390.8 ↓	403.2 ↓

With respect to both the path tracking accuracy and efficiency, the self-optimizing controller can control the steering wheel angle with small lateral distance and heading angle deviations, to keep the vehicle driving in the center of the lane. As shown in Figures 31 and 32, the vehicle follows the reference trajectory quite satisfactorily with the self-optimizing controller on the curved track, the straight track, and the roundabout track, and its control performance was better than that of the traditional control method.

The memory data buffer stores the driver's experience behavior data, including the vehicle status information and operating sequences. Therefore, during the training process, the amount of positive sample data required for network training increases, meaning that the reward value of the controller shows an increasing trend (see Figure 33). The test results prove that after 35,000 iteration steps, the actor–critic network has learned an excellent control policy.

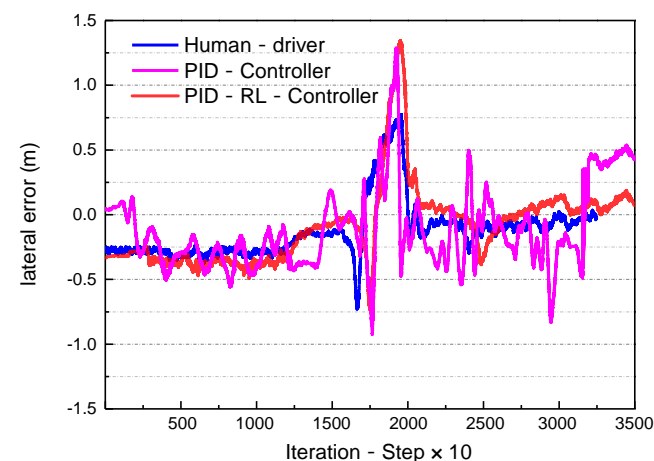


Figure 31. Lateral error of path tracking controller on a test road map.

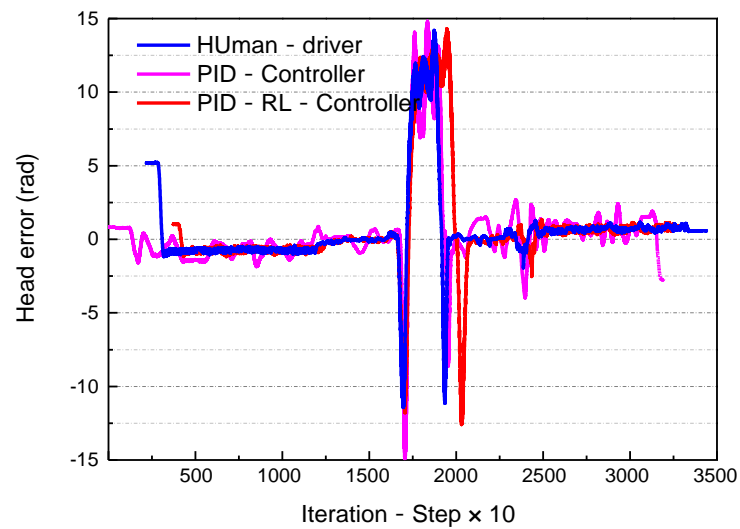


Figure 32. Heading error of path tracking controller on a test road map.

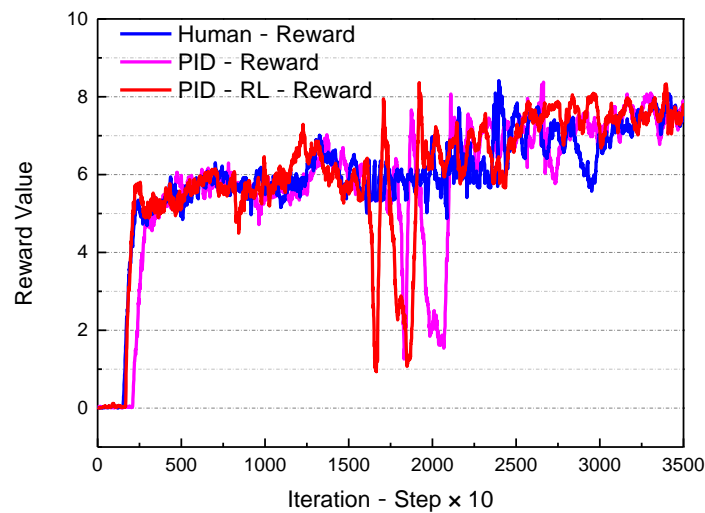


Figure 33. Single-step reward value of path tracking controller on the test road map.

The intermediate state variables generated during the training process are the gain parameters of the controller, which are used to compensate for the dynamic error of the system to keep the vehicle always driving along the center of the lane, under different types of road trajectories. The data distribution and statistical results are shown in Figure 34 and Table 15, respectively.

Table 15. Mathematical statistics of path tracking control parameters.

	Mean	Standard Deviation	Minimum	Median	Maximum
ΔP_1	53.84405	10.34426	16.2258	56.0301	79.3932
ΔD_1	92.10457	2.05332	85.6766	92.0764	98.2066
ΔP_2	39.07703	0.49852	37.5222	39.0849	40.747
ΔD_2	6.46417	0.24382	5.6081	6.46435	7.1453

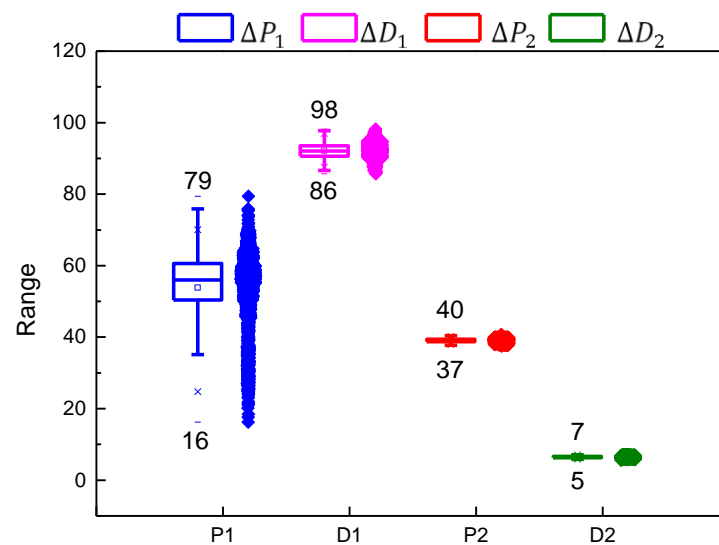


Figure 34. Data distribution of path tracking control parameters.

It can be seen that, when the actual trajectory is far away from the reference trajectory—such as on the curve and the roundabout tracks—the percentage gain, ΔP_1 , increases with the curvature of the reference track. The maximum value can reach 79; when the vehicle is travelling in a straight line, the minimum value reaches 16. Therefore, the standard deviation reaches a value of 10.34426. Thus, the controller responds faster to a decrease in the lateral deviation. When approaching the reference trajectory, the heading angle deviation and its rate of change have a greater impact, with average values reaching 39.07703 and 6.46417, respectively.

We can conclude that, in the absence of prior knowledge of the dynamic characteristics of the vehicle's physical model, the optimization problem of the path tracking controller can be solved based on real-time interactive learning with the operating environment of a real vehicle; moreover, the controller can realize the self-optimized tuning of the PID control parameters, based on the RL algorithm, which can be used for online learning and the optimization of complex path tracking control.

5. Conclusions

In this paper, we propose a self-optimized path tracking controller to simultaneously track a predefined path with high accuracy and a well ride comfort experience. For the lateral control of the vehicle, a steering method, based on the fusion of the reinforcement learning with traditional PID controller, is designed to adapt to various tracking scenarios. According to the pre-defined path geometry and the real-time status of the vehicle, combined with the environment interactive learning mechanism, based on the RL framework, the optimization of the PID control parameters can be realized. The adaptive performance of velocity changes was also considered in the tracking process. Both the driving simulator and the on-site vehicle experiments have verified the effects of our proposed self-optimization controller. Nevertheless, there remains a gap between simulation and real scenes; a transfer learning (sim-to-real) strategy can better adapt to controllers to real vehicles, which should be emphasized in our further research.

6. Discussion of Limitations and Future Work

One challenge for the self-optimizing path tracking controller, based on RL, is the question of how to design an accurate reward function and effectively balance exploration and utilization, in order to avoid the network training falling into local optimality. In addition, although the simulations can provide large amounts of cheap data for the training and testing of the RL agent, the gap between simulation and reality is also the main reason that these approaches are difficult to popularize and apply in real-world engineering

problems. In future research, we will focus on the application of transfer learning in the sim-to-real domain. The full name of sim-to-real is simulation to reality, which is a branch of reinforcement learning and a kind of transfer learning [52]. In the field of robotics or autonomous driving, the main problem that transfer learning solves is that of how to directly allow the autonomous systems or agents to interact with the virtual environment and the real environment [53,54]. Reinforcement learning is considered as a promising direction for driving policy learning. However, training autonomous driving vehicle with reinforcement learning in real environment involves non-affordable trial-and-error research methods [55]. It is more desirable to first train in a virtual environment and then transfer to the real environment.

Author Contributions: Data curation, J.M.; formal analysis, J.M.; funding acquisition, H.X.; investigation, J.M.; project administration, H.X.; funding acquisition, H.X. and K.S.; software, J.M.; supervision, H.X.; validation, H.L.; visualization, H.L.; writing—original draft, J.M.; Writing—review and editing, K.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Tianjin Science and Technology Planning Project (2019): Research and Application of Deep Reinforcement Learning Control Algorithm for Intelligent Unmanned System (award number: 19ZXZNGX00050).

Conflicts of Interest: There are no conflict of interest to declare.

References

1. Visioli, A. *Practical PID Control*; Springer: Berlin/Heidelberg, Germany, 2006.
2. Jeffrey, S.; Wit, J.; Crane, C.D., III; Armstrong, D. *Autonomous Ground Vehicle Path Tracking*; University of Florida: Gainesville, FL, USA, 2000.
3. Johary, N.M. Path Tracking Algorithm for An Autonomous Ground Robot. Ph.D. Thesis, Universiti Tun Hussein Onn Malaysia, Batu Pahat, Malaysia, 2014.
4. Goh, J.Y.; Goel, T.; Gerdes, J.C. A controller for automated drifting along complex trajectories. In Proceedings of the 14th International Symposium on Advanced Vehicle Control (AVEC 2018), Beijing, China, 16–20 July 2018.
5. Goh, J.Y.; Gerdes, J.C. Simultaneous stabilization and tracking of basic automobile drifting trajectories. In Proceedings of the 2016 IEEE Intelligent Vehicles Symposium (IV), Los Angeles, CA, USA, 11–14 June 2017; pp. 597–602.
6. Hindiyeh, R.Y.; Gerdes, J.C. A controller framework for autonomous drifting: Design, stability, and experimental validation. *J. Dyn. Syst. Meas. Control.* **2014**, *136*, 051015. [[CrossRef](#)]
7. Kim, D.; Yi, K. Design of a Path for Collision Avoidance and Path Tracking Scheme for Autonomous Vehicles. *IFAC Proc. Vol.* **2009**, *42*, 391–398. [[CrossRef](#)]
8. Chen, S.-P.; Xiong, G.-M.; Chen, H.-Y.; Negrut, D. MPC-based path tracking with PID speed control for high-speed autonomous vehicles considering time-optimal travel. *J. Central South Univ.* **2020**, *27*, 3702–3720. [[CrossRef](#)]
9. Wang, H.; Liu, B.; Ping, X.; An, Q. Path Tracking Control for Autonomous Vehicles Based on an Improved MPC. *IEEE Access* **2019**, *7*, 161064–161073. [[CrossRef](#)]
10. Kim, D.; Kang, J.; Yi, K. Control strategy for high-speed autonomous driving in structured road. In Proceedings of the 2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC), Washington, DC, USA, 5–7 October 2011.
11. Vivek, K.; Sheta, M.A.; Gumtapure, V. A Comparative Study of Stanley, LQR and MPC Controllers for Path Tracking Application (ADAS/AD). In Proceedings of the 2019 IEEE International Conference on Intelligent Systems and Green Technology (ICISGT), Visakhapatnam, India, 29–30 June 2019.
12. Tiep, D.K.; Lee, K.; Im, D.-Y.; Kwak, B.; Ryoo, Y.-J. Design of Fuzzy-PID Controller for Path Tracking of Mobile Robot with Differential Drive. *Int. J. Fuzzy Log. Intell. Syst.* **2018**, *18*, 220–228. [[CrossRef](#)]
13. El Hamidi, K.; Mjahed, M.; El Kari, A.; Ayad, H. Neural Network and Fuzzy-logic-based Self-tuning PID Control for Quadcopter Path Tracking. *Stud. Inform. Control* **2019**, *28*, 401–412. [[CrossRef](#)]
14. Liang, X.; Zhang, W.; Wu, Y. Automatic Collimation of Optical Path Based on BP-PID Control. In Proceedings of the 2017 10th International Conference on Intelligent Computation Technology and Automation (ICICTA), Changsha, China, 9–10 October 2017.
15. Ma, L.; Yao, Y.; Wang, M. The Optimizing Design of Wheeled Robot Tracking System by PID Control Algorithm Based on BP Neural Network. In Proceedings of the 2016 International Conference on Industrial Informatics-Computing Technology, Wuhan, China, 3–4 December 2016.
16. El Sallab, A.; Abdou, M.; Perot, E.; Yogamani, S. Deep Reinforcement Learning framework for Autonomous Driving. *Electron. Imaging* **2017**, *2017*, 70–76. [[CrossRef](#)]
17. Wang, S.; Jia, D.; Weng, X. Deep Reinforcement Learning for Autonomous Driving. *arXiv* **2018**, arXiv:1811.11329.
18. Dong, L.; Zhao, D.; Zhang, Q.; Chen, Y. Reinforcement Learning and Deep Learning based Lateral Control for Autonomous Driving. *arXiv* **2018**, arXiv:1810.12778.

19. Wymann, B.; Espié, E.; Guionneau, C.; Dimitrakakis, C.; Coulom, R.; Sumner, A. TORCS, The Open Racing Car Simulator, v1.3.5. 2013. Available online: <http://torcs.sourceforge.net/> (accessed on 10 December 2019).
20. Ingram, A. Gran Turismo Sport—Exploring Its Impact on Real-World Racing with Kazunori. 2019. Available online: Yamauchi.evo.co.uk (accessed on 1 June 2020).
21. Fuchs, F.; Song, Y.; Kaufmann, E.; Scaramuzza, D.; Dürr, P. Super-Human Performance in Gran Turismo Sport Using Deep Reinforcement Learning. *arXiv* **2020**, arXiv:2008.07971. [[CrossRef](#)]
22. Cai, P.; Mei, X.; Tai, L.; Sun, Y.; Liu, M. High-Speed Autonomous Drifting With Deep Reinforcement Learning. *IEEE Robot. Autom. Lett.* **2020**, *5*, 1247–1254. [[CrossRef](#)]
23. Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; Koltun, V. Carla: An open urban driving simulator. *arXiv* **2017**, arXiv:1711.03938.
24. Gao, X.; Gao, R.; Liang, P.; Zhang, Q.; Deng, R.; Zhu, W. A Hybrid Tracking Control Strategy for Nonholonomic Wheeled Mobile Robot Incorporating Deep Reinforcement Learning Approach. *IEEE Access* **2021**, *9*, 15592–15602. [[CrossRef](#)]
25. Zhang, Y.; Zhang, Y.; Yu, Z. Path Following Control for UAV Using Deep Reinforcement Learning Approach. *Guid. Navig. Control* **2021**, *1*, 2150005. [[CrossRef](#)]
26. Duan, K.; Fong, S.; Chen, C.P. Reinforcement Learning Based Model-free Optimized Trajectory Tracking Strategy Design for an AUV. *Neurocomputing* **2022**, *469*, 289–297. [[CrossRef](#)]
27. Li, B.; Wu, Y. Path Planning for UAV Ground Target Tracking via Deep Reinforcement Learning. *IEEE Access* **2020**, *8*, 29064–29074. [[CrossRef](#)]
28. Wang, S.; Yin, X.; Li, P.; Zhang, M.; Wang, X. Trajectory Tracking Control for Mobile Robots Using Reinforcement Learning and PID. *Iran. J. Sci. Technol. Trans. Electr. Eng.* **2020**, *44*, 1059–1068. [[CrossRef](#)]
29. Xiao, J.; Li, L.; Zou, Y.; Zhang, T. Reinforcement Learning for Robotic Time-optimal Path Tracking Using Prior Knowledge. *arXiv* **2019**, arXiv:1907.00388.
30. Zhang, S.; Wang, W. Tracking Control for Mobile Robot Based on Deep Reinforcement Learning. In Proceedings of the 2019 2nd International Conference on Intelligent Autonomous Systems (ICoIAS), Singapore, 28 February–2 March 2019.
31. Arroyo, M.A.; Giraldo, L.F. Data-driven Outer-Loop Control Using Deep Reinforcement Learning for Trajectory Tracking. *arXiv* **2020**, arXiv:2008.13732.
32. Shan, Y.; Zheng, B.; Chen, L.; Chen, L.; Chen, D. A Reinforcement Learning-Based Adaptive Path Tracking Approach for Autonomous Driving. *IEEE Trans. Veh. Technol.* **2020**, *69*, 10581–10595. [[CrossRef](#)]
33. Puccetti, L.; Köpf, F.; Rathgeber, C.; Hohmann, S. Speed Tracking Control using Online Reinforcement Learning in a Real Car. In Proceedings of the 6th IEEE International Conference on Control, Automation and Robotics (ICCAR), Singapore, 20–23 April 2020.
34. Wang, N.; Gao, Y.; Yang, C.; Zhang, X. Reinforcement Learning-based Finite-time Tracking Control of an Unknown Unmanned Surface Vehicle with Input Constraints. *Neurocomputing* **2021**. Available online: <https://www.sciencedirect.com/science/article/abs/pii/S0925231221015733> (accessed on 10 June 2021). [[CrossRef](#)]
35. Jiang, L.; Wang, Y.; Wang, L.; Wu, J. Path tracking control based on Deep reinforcement learning in Autonomous driving. In Proceedings of the 2019 3rd Conference on Vehicle Control and Intelligence (CVCI), Hefei, China, 21–22 September 2019.
36. Kamran, D.; Zhu, J.; Lauer, M. Learning Path Tracking for Real Car-like Mobile Robots From Simulation. In Proceedings of the 2019 European Conference on Mobile Robots (ECMR), Prague, Czech Republic, 4–6 September 2019.
37. Riedmiller, M.; Montemerlo, M.; Dahlkamp, H. Learning to Drive a Real Car in 20 Minutes. In Proceedings of the Frontiers in the Convergence of Bioscience & Information Technologies IEEE Computer Society, Jeju City, Korea, 11–13 October 2007.
38. Kendall, A.; Hawke, J.; Janz, D.; Mazur, P.; Reda, D.; Allen, J.-M.; Lam, V.-D.; Bewley, A.; Shah, A. Learning to Drive in a Day. *arXiv* **2018**, arXiv:1807.00412.
39. Rajamani, R. *Vehicle Dynamics and Control*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2011.
40. Kong, J.; Pfeiffer, M.; Schildbach, G.; Borrelli, F. Kinematic and dynamic vehicle models for autonomous driving control design. In Proceedings of the 2015 IEEE Intelligent Vehicles Symposium (IV), Seoul, Korea, 29 June–1 July 2015.
41. Zhu, M.; Wang, X.; Wang, Y. Human-like autonomous car-following model with deep reinforcement learning. *Transp. Res. Part C Emerg. Technol.* **2018**, *97*, 348–368. [[CrossRef](#)]
42. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
43. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.
44. Yu, A.; Palefsky-Smith, R.; Bedi, R. Course Project Reports: Deep Reinforcement Learning for Simulated Autonomous Vehicle Control. *Course Proj. Rep. Winter* **2016**. Available online: http://cs231n.stanford.edu/reports/2016/pdfs/112_Report.pdf (accessed on 10 June 2021).
45. Yu, R.; Shi, Z.; Huang, C.; Li, T.; Ma, Q. Deep reinforcement learning based optimal trajectory tracking control of autonomous underwater vehicle. In Proceedings of the 2017 36th Chinese Control Conference (CCC), Dalian, China, 26–28 July 2017.
46. Monahan, G.E. A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms. *Manag. Sci.* **1982**, *28*, 1–16. [[CrossRef](#)]
47. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
48. Konda, V.R.; Tsitsiklis, J.N. Actor-critic algorithms. *SIAM J. Control Optim.* **2002**, *42*, 1143–1166. [[CrossRef](#)]

49. Yan, Z.; Zhuang, J. Active Disturbance Rejection Algorithm Applied to Path Tracking in Autonomous Vehicles. *J. Chongqing Univ. Technol. Nat. Sci.* **2020**, 1–10. (In Chinese). Available online: <http://kns.cnki.net/kcms/detail/50.1205.T.20200522.1459.004.html> (accessed on 10 June 2021).
50. Chao, C.; Gao, H.; Ding, L.; Li, W.; Yu, H.; Deng, Z. Trajectory tracking control of wms with lateral and longitudinal slippage based on active disturbance rejection control. *Robot. Auton. Syst.* **2018**, *107*, 236–245.
51. Gao, Y.; Xia, Y. Lateral path tracking control of autonomous land vehicle based on active disturbance rejection control. In Proceedings of the 32nd Chinese Control Conference, Xian, China, 26–28 July 2013.
52. Pan, X.; You, Y.; Wang, Z.; Lu, C. Virtual to Real Reinforcement Learning for Autonomous Driving. In Proceedings of the 2017 British Machine Vision Conference, London, UK, 4–7 September 2017.
53. Hu, H.; Zhang, K.; Tan, A.H.; Ruan, M.; Agia, C.; Nejat, G. A Sim-to-Real Pipeline for Deep Reinforcement Learning for Autonomous Robot Navigation in Cluttered Rough Terrain. *IEEE Robot. Autom. Lett.* **2021**, *6*, 6569–6576. [[CrossRef](#)]
54. Chaffre, T.; Moras, J.; Chan-Hon-Tong, A.; Marzat, J. Sim-to-Real Transfer with Incremental Environment Complexity for Reinforcement Learning of Depth-based Robot Navigation. In Proceedings of the 17th International Conference on Informatics in Control, Automation and Robotics, Paris, France, 7–9 July 2020.
55. Suenaga, R.; Morioka, K. Development of a Web-Based Education System for Deep Reinforcement Learning-Based Autonomous Mobile Robot Navigation in Real World. In Proceedings of the 2020 IEEE/SICE International Symposium on System Integration (SII), Honolulu, HA, USA, 12–15 January 2020.