


Article

# Intelligent Land-Vehicle Model Transfer Trajectory Planning Method Based on Deep Reinforcement Learning

Lingli Yu <sup>1,2,3</sup> , Xuanya Shao <sup>1,\*</sup>, Yadong Wei <sup>1</sup> and Kaijun Zhou <sup>4</sup>

<sup>1</sup> School of Information Science and Engineering, Central South University, Changsha 410083, China; llyu@csu.edu.cn (L.Y.); 13477011934@163.com (Y.W.)

<sup>2</sup> State Key Laboratory of Robotics and System, Harbin Institute of Technology, Haerbin 150001, China

<sup>3</sup> State Key Laboratory of Mechanical Transmissions, Chongqing University, Chongqing 400044, China

<sup>4</sup> School of Computer and Information Engineering, Hunan University of Commerce, Changsha 410205, China; alpha218@126.com

\* Correspondence: 174611083@csu.edu.cn; Tel.: +86-130-5516-6724

Received: 1 August 2018; Accepted: 29 August 2018; Published: 1 September 2018



**Abstract:** To address the problem of model error and tracking dependence in the process of intelligent vehicle motion planning, an intelligent vehicle model transfer trajectory planning method based on deep reinforcement learning is proposed, which is able to obtain an effective control action sequence directly. Firstly, an abstract model of the real environment is extracted. On this basis, a deep deterministic policy gradient (DDPG) and a vehicle dynamic model are adopted to jointly train a reinforcement learning model, and to decide the optimal intelligent driving maneuver. Secondly, the actual scene is transferred to an equivalent virtual abstract scene using a transfer model. Furthermore, the control action and trajectory sequences are calculated according to the trained deep reinforcement learning model. Thirdly, the optimal trajectory sequence is selected according to an evaluation function in the real environment. Finally, the results demonstrate that the proposed method can deal with the problem of intelligent vehicle trajectory planning for continuous input and continuous output. The model transfer method improves the model's generalization performance. Compared with traditional trajectory planning, the proposed method outputs continuous rotation-angle control sequences. Moreover, the lateral control errors are also reduced.

**Keywords:** intelligent driving vehicle; trajectory planning; end-to-end; deep reinforcement learning; model transfer

## 1. Introduction

Although intelligent driving technology is developing rapidly, some new problems are emerging during development. In 2016, the first major accident of Tesla happened in the field of automatic driving. Meanwhile, Uber suffered an incident of automation driving hitting pedestrians on 28 March 2018. These problems greatly aroused worldwide attention on the safety of intelligent driving. Therefore, there is still a long way for intelligent driving to improve its innovative and stable safety. As the key to its technology, trajectory planning technology is attracting more and more attention and exploration by researchers at home and abroad.

Trajectory planning is not only applied to intelligent vehicles, but also widely used in the field of robotics and unmanned aerial vehicles [1,2]. There are various ways of trajectory generation in trajectory planning, including the Nelson polynomial, spiral curve equation, spline curve, Bezier curve, etc. [3]. For example, a fourth-order polynomial and dynamic bicycle model were utilized to describe a vehicles

kinematics model [4], considering the overtaking and chasing behavior of different cost functions in each case. However, it assumes that the vehicle velocity is a constant, which conflicts with most actual situations. Yu, L et al. [5] put forward a technique of trajectory smoothing and stitching based on a Bezier Curve. Sahingoz, O.K. [6] proposed trajectory planning based on a Bezier Curve that takes into consideration the kinematics constraint, initial state constraint, target state constraint, and curvature continuous constraint.

With the rapid development of deep learning, vision-based control methods acquired great achievements [7]. Hotz [8] adopted a variational auto-encoder (VAE) and a generative adversarial network (GAN) to achieve image coding, road tracking, and intelligent driving vehicle potential space decoding. Low-level control strategy and advanced prior action were learned through a neural network, and multi-level strategies were taken as heuristic search algorithms to realize complex motion planning tasks [9]. Deep learning models were adopted to establish the mapping relationship between lidar distance, target position, and control instruction [10]. To realize the motion planning of an intelligent vehicle, Liu, W et al. [11] and Lin, Y.L et al. [12] proposed deep learning to establish the mapping relationship between the control sequence and the corresponding trajectory.

In recent years, reinforcement learning was applied to robot control tasks. A deep Q network (DQN) was proposed to deal with a discrete action continuous state, which aimed to combine a deep neural network with reinforcement learning [13]. Subsequently, Lillicrap, T.P et al. [14] and Gu, S et al. [15] offered an offline depth reinforcement learning algorithm based on a deep Q network and extended it to continuous high-dimensional state space. Schaul, T et al. [16] and Metz, L et al. [17] made it possible to achieve multiple targets by extending the DQN. Schaul, T et al. [18] proposed prior experience replay technology to improve the performance of the DQN. Andrychowicz, M et al. [19] explicated experience to improve sample collection efficiency. The Ornstein–Uhlenbeck (OU) process [20] was used to add noise after an action strategy to improve network exploration ability. Plappert, M et al. [21] introduced network parameter hierarchy with noise to improve network performance. Gu, S et al. [22] showed the possibility of learning complex manipulation strategies without demonstrations. Genders, W et al. [23] adopted a deep reinforcement learning model to establish a traffic signal agent, while Isele, D et al. [24] solved the problem of a complex traffic intersection without traffic signals. Tai, L et al. [25] proposed a motion planning method without a map. The sparse sensor ranging information and target position were utilized as input, while the continuous steering command was taken as output, and verification was conducted in practical experiments. Data efficiency and task performance were improved by addressing the problem of maximizing cumulative rewards for reinforcement learning (RL), and considering supervised/unsupervised learning styles, so as to achieve navigational capabilities [26].

When the model is known, a strategy iteration and value iterative algorithm based on dynamic programming (DP) [27] is able to update value functions after each step of the strategy, which is efficient. The intelligent vehicle driving problem is a model-free problem. The Monte Carlo (MC) [28] reinforcement learning algorithm overcomes the difficulties caused by the unknown model estimation by considering the sampling trajectories. This algorithm updates the value estimate of the strategy after completing a sampling trajectory, which is inefficient compared to the algorithm based on dynamic programming. Temporal difference (TD) learning [29] combines dynamic programming and Monte Carlo reinforcement learning for more efficient model-free learning.

As known, Q-learning solves the low-dimensional problem of discrete space, which is a classic case of temporal difference learning. The DQN improves the processing ability of high-dimensional state space, but it is still unable to cope with high-dimensional continuous action space. The Actor-Critic (AC) method [30] is able to handle continuous action space, but the randomness strategy makes it difficult for the network to converge. To this end, the deep deterministic policy gradient (DDPG) [31] adopts the Actor-Critic framework to combine the advantages of DQN to solve the problem of continuous state space and continuous action space. Moreover, it adopts a deterministic policy to ensure the network is more convergent. Since traditional motion cannot eliminate the model error, an end-to-end model transfer trajectory planning method based on depth reinforcement learning is proposed in this study.

Furthermore, DDPG is a deep reinforcement learning method, and it is utilized to train the model in a simple virtual environment which is constructed independently, thereby reducing its dependence on the sample data. Additionally, it can deal with the model training of continuous input and continuous output, thus directly outputting the control action and trajectory sequence. The complexity is lower than that of the optimal control calculation, and the model transfer method was applied to improve the model's generalization performance. Compared with traditional planning and end-to-end planning, the proposed method has more continuous corner control sequences and smaller lateral errors while the vehicle is driving.

## 2. Reinforcement Learning and Description of the Driving Environment

### 2.1. Reinforcement Learning Method

The basic principle of reinforcement learning is presented in Figure 1. When the agent is required to achieve a task, it first interacts with environment (*Env*) via action (*a*); then, the impact of the action on the environment brings the agent into a new state (*s*). At the same time, the agent receives reward feedback (*Reward*) from the environment. The agent and environment generate a large amount of data through a continuous loop and interaction. Reinforcement learning utilizes these sample data to adjust the strategy  $\pi$ . Afterward, it interacts with *Env* to enter a new *state*, generating new  $data = (s_t, a_t, r_t, s_{t+1})$ . Subsequently, the new samples are adopted to modify the strategy  $\pi$  for several iterations. After a great deal of iterative learning, the agent finally learns the optimal strategy  $\pi^*$  to complete the corresponding task.

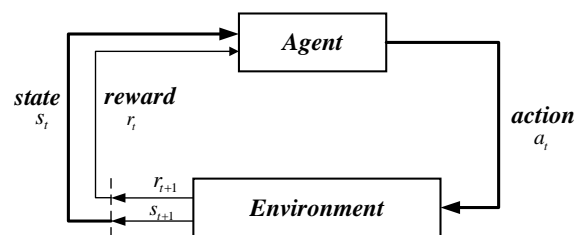


Figure 1. The principle of reinforcement learning.

Strategy  $\pi$  refers to the agent's selection of action *a* under state *s*, which is the key problem in reinforcement learning. Strategy  $\pi$  is a map from the agent, which is aware of environmental state *s* to action *a*. The random strategy selects the corresponding action according to the probability  $\pi(a|s)$  of each action, while deterministic policy selects action  $a = \pi(s)$  directly according to *s*.

$$\begin{aligned} \text{stochastic Policy} : \quad & \sum \pi(a|s) = 1 \\ \text{deterministic Policy} : \quad & \pi(s) : S \rightarrow A \end{aligned} \quad (1)$$

Cumulative rewards or returns are calculated when a strategy  $\pi$  is given. The definition of cumulative returns is as follows:

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \quad (2)$$

where  $0 < \gamma < 1$  is the discount factor of long-term income. The state function is defined as the cumulative return benefit corresponding to state *s* under *a* strategy  $\pi$ :

$$v_{\pi}(s) = E_{\pi} \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right]. \quad (3)$$

The corresponding state-action value function is defined as

$$q_{\pi}(s, a) = E_{\pi} \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right]. \quad (4)$$

## 2.2. Virtual Driving Environment Model Design

According to the description of the intelligent vehicle driving scene, intelligent driving behavior decision tasks include normal driving, changing/overtaking, curve/ramp driving, and so on. Here, the environment model was built as a circular map with three lanes, as shown in Figure 2. Specifically, the green area and outer lane boundary are deemed insurmountable obstacles, while the others are free travel space. The light-blue lines indicate the desired path with rewards,  $path_d = (X_d, Y_d, \phi_d)$ . The intelligent vehicle,  $Car = (x_c, y_c, \phi_c, v)$ , drives in a circular map and learns intelligent driving maneuvers, including straight, changing, and curving driving behavior. Lastly,  $x_c, y_c, \phi_c$  represents the current position and posture information of the intelligent vehicle.

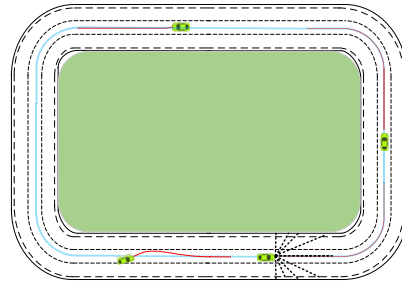


Figure 2. Virtual environment abstract model.

To simplify the environment model *Env*, the intelligent vehicle has  $n$  ranging beams *Sensor*. Furthermore, the farthest distance  $d_{Max}$  of each ranging beam is the same. Each ranging beam supports feedback information  $Sensor_i = (d_i, x_{end}, y_{end})$  to the intelligent vehicle when it encounters obstacles. Here,  $d_i$  is the length that the ranging beam is blocked by obstacles or boundaries, and  $x_{end}, y_{end}$  are the position coordinates of the beam in contact with obstacles or boundaries. The intelligent vehicle speed keeps a constant  $v$ , and the angle control output is  $-\delta_{min} \leq \delta_t \leq \delta_{max}$ . Thus, the key return function *Reward* for the environment model *Env* is as follows:

$$Reward = \begin{cases} -1 & \text{when contacts with obstacles or boundaries} \\ R_{action} + R_{money} & \text{else} \end{cases}, \quad (5)$$

$$\begin{cases} R_{action} = -\lambda_1 \times \|\delta_{old} - \delta\|^2 \\ R_{money} = \begin{cases} 0 & \text{get\_money} = \text{False} \\ 0.1 & \text{get\_money} = \text{True} \end{cases} \end{cases}, \quad (6)$$

$$get\_money = \begin{cases} \text{True} & \text{if } |\Delta x| \leq \varepsilon_1 \ \& \ |\Delta y| \leq \varepsilon_2 \ \& \ |\Delta \phi| \leq \varepsilon_3 \\ \text{False} & \text{other else} \end{cases}, \quad (7)$$

where  $\lambda_1$  is the positive penalty coefficient, and  $R_{action}$  represents the difference penalty between the front and rear successive front wheel angles  $\delta_{old}, \delta$  of an intelligent vehicle. The smaller the change is between successive actions, the smaller the penalty.  $R_{money}$  represents the reward for an intelligent vehicle driving on the desired path,  $(\Delta x, \Delta y, \Delta \phi)$  is the difference between the current posture  $(x_c, y_c, \phi_c)$  and the desired path  $path_d = (X_d, Y_d, \phi_d)$ , and  $\varepsilon_1, \varepsilon_2, \varepsilon_3$  is the fault-tolerant error.

Intelligent vehicles randomize their initial position  $(x_0, y_0, \phi_0)$  according to the given policies  $\pi_{reset}$  to ensure a more adequate exploration of the environment and the stability of the result. Intelligent vehicle termination conditions at each epoch include 1) contact with obstacles or boundaries; 2) meeting

the maximum number of driving steps,  $n_{step} = Num_{max}$ . The optimal intelligent driving maneuver for intelligent vehicle learning is  $\pi$ ; therefore, the strategy space of the model is  $\pi_{all} = \{\pi_{reset}, \pi\}$ .

$$\pi_{reset} : \begin{cases} x_0 \in [X_{min}, X_{max}] \\ y_0 \in [Y_{min}, Y_{max}] \\ \varphi_0 \in [\phi_{min}, \phi_{max}] \end{cases} . \quad (8)$$

State space is assumed as  $\Sigma(Sensor) = \{d_0, d_1, \dots, d_n\}$ , and motion space is assumed as  $\Sigma(\delta) = \{\delta_{new}\}$ . For the intelligent vehicle and environment model *Env*, an abstract model *M* is constructed.

$$M = \{Env, Car, \Sigma(Sensor), \Sigma(\delta), \pi_{all}, Reward\}.$$

### 3. Model Transfer Trajectory Planning Based on Deep Reinforcement Learning (DRL-MTTP)

#### 3.1. DDPG Network Structure and Algorithm Flow

For complex continuous state space  $\Sigma(Sensor)$  and continuous action space  $\Sigma(\delta)$ , it is necessary to train the deep reinforcement learning model  $M_\theta$  in a virtual environment *M* using the DDPG algorithm. The DDPG algorithm consists of an Actor policy network and a Critic evaluation network, as shown in Figure 3.

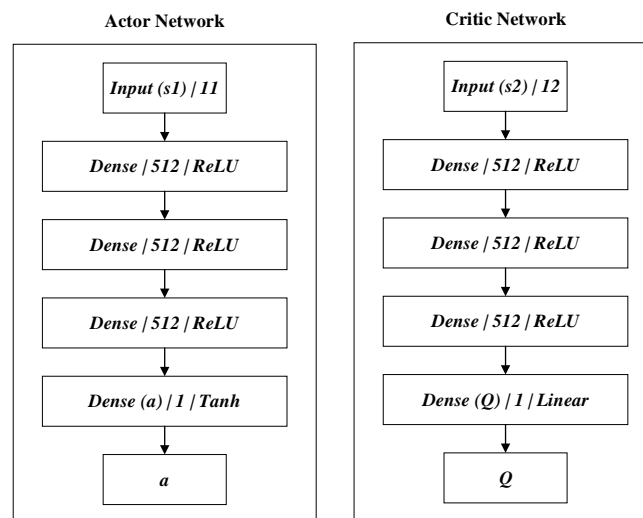


Figure 3. Deep deterministic policy gradient (DDPG) network structure.

The state  $s_{sensor} \in \Sigma(Sensor)$ , speed  $v$ , and action  $\delta_{old}$  of the final moment are combined as  $s_a = (s_{sensor}, v, \delta_{old})$ . They are adopted as the input to the Actor policy network; therefore, the number of Actor policy network input-layer neurons is 11. Meanwhile, the policy network's hidden layer utilizes three full connected networks; each layer contains 512 neurons. The fully connected layer is followed by batch normalization (BN), before the *ReLU* (a type of activation function) is adopted as the activation function. At the same time, the last layer of the network chooses *tanh* as the activation function to map the network output between the interval  $[-1, 1]$ . The network output is action  $\delta \in \Sigma(\delta)$ . After the state  $s_a$  and action  $\delta$  are merged as  $s_c = (s_a, \delta)$ , then they become the input to the Critic evaluation network. The number of Critic policy network input-layer neurons is 12. Meanwhile, the policy network's hidden layer utilizes three full connected networks; each layer contains 512 neurons. The fully connected layer is followed by BN. Although the hidden layer of the evaluation network and the policy network hold the same structure, its last layer is activated by a linear function such as that in Equation (9). Thus, the network output is the corresponding Q-value,  $Q(s_a, \delta)$ , of  $s_c$ .

$$y = kx + b, \quad (9)$$

where  $x$  is the input of the last layer,  $y$  is the predicted Q-value, and  $k, b$  are the weight and bias for network training.

DDPG adopts the Actor-Critic framework, including the Actor and Critic structure. Here, the Actor part includes the online policy network and the target policy network, which adopt the deterministic policy to get a definite action from the current state. The Critic part includes an online Q network and a target Q network, in which the Bellman equation of the action–state function Q is utilized to measure the quality of action. The pseudo code of the DDPG algorithm is shown in Algorithm 1 and the DDPG algorithm flow is shown in Figure 4. The input state  $(d_1, \dots, d_9, v, \delta_{old})$  and output action  $\delta$  are represented accordingly. The DDPG algorithm adopts a deterministic policy, and the policy output is an action. Therefore, it needs less sampled data to maximize efficiency. However, this results in the environment not being explored. In order to improve the algorithm’s exploration ability, the OU stochastic process was added to the deterministic policy action. Furthermore, the environmental execution was carried out after sampling from a random process of the action. Due to its good correlation over a time series, the OU stochastic process is able to explore environments with momentum properties by the agent.

---

**Algorithm 1. Pseudo code of the deep deterministic policy gradient (DDPG) algorithm. OU—Ornstein–Uhlenbeck process.**

---

1. Randomly initialize Critic online Q network parameters  $\theta^Q$  and Actor’s online policy network parameters  $\theta^\mu$ .
  2. Initialize Critic target Q network parameters  $\theta^{Q'} \leftarrow \theta^Q$  and Actor’s target policy network parameters  $\theta^{\mu'} \leftarrow \theta^\mu$ .
  3. Initialize experience replay memory (R).
  4. for *episode* = 1, M do
  5.     Initialize the OU random process  $D$  for the exploration of action.
  6.     Input initial observation state  $s_1$ .
  7.     for  $t = 1, T$  do
  8.         Choose action  $a_t$  based on current strategy  $\mu(s_t)$  and exploring noise  $D_t$ :  

$$a_t = \mu(s_t) + D_t$$
  9.         Perform the action  $a_t$ , get the reward  $r_t$ , and observe the new state  $s_{t+1}$ .
  10.         Store the process  $(s_t, a_t, r_t, s_{t+1})$  in R.
  11.         Sampling from R to get the process  $(s_i, a_i, r_i, s_{i+1})$  of batch  $N$ .
  12.         Set  $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} | \theta^{\mu'})) | \theta^{Q'}$  //  $Q'$  is the state–action value calculated by the target Q network, and  $\mu'$  is the current strategy obtained by the target policy network.
  13.         Update Critic’s online Q network by minimizing the loss function:  

$$L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i | \theta^Q))^2$$
  14.         Update the Actor’s online policy network with sampling gradient:  

$$\nabla_{\theta^\mu} \mu | s_i \approx \frac{1}{N} \sum_i \nabla_a Q(s, a | \theta^Q) |_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s | \theta^\mu) | s_i$$
  15.         Update Critic’s target Q network :  $\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$
  16.         Update Actor’s target policy network :  $\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$
  17.     end for
  18. end for
-

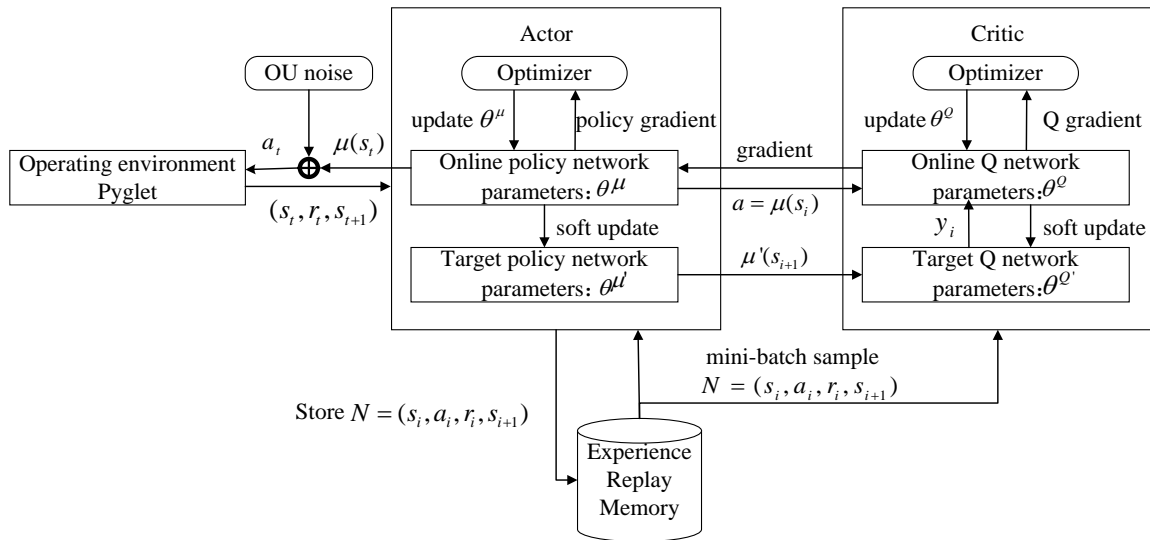


Figure 4. DDPG algorithm flow chart. OU—Ornstein–Uhlenbeck process.

### 3.2. Model Transfer Strategy

The intelligent vehicle *Car* obtains the optimal intelligent driving strategy  $\pi$  from the environment model  $M$  through deep reinforcement learning training. The real environment  $M^*$  is obviously different from the virtual environment model  $M$ ; the former usually tends to be more complex and time-varying. Therefore, if the training model of virtual environment  $M$  is directly applied to real environment  $M^*$ , it brings numerous predictable or unpredictable problems.

The decision tasks of intelligent vehicles include driving straight, changing lanes, crossing corners, ramp driving, etc. Thus, the model of intelligent driving tasks is abstracted from the real environment  $M^*$  and migrated to the virtual environment  $M$ , which maps onto a location area corresponding to the ring map in  $M$ . Then, the optimal driving strategy  $\pi$  is adopted to plan the control-trajectory sequence  $C = \{\delta, \zeta\}$  to achieve the driving task, including the control sequence  $\delta = \{\delta_1, \delta_2, \dots, \delta_t\}$  and its corresponding trajectory  $\zeta = \{p_1, p_2, \dots, p_t\}$ . Finally, the intelligent vehicles carry out the task  $\delta$  to complete the driving task in the real environment  $M^*$ .

According to a different sub-task  $\mathbb{Z}$ , such as lane keeping, lane changing, and overtaking, the fixed reference points  $P_{ref} | \mathbb{Z} = (x_{ref}, y_{ref}, \varphi_{ref})$  in  $M$  are set as a reference target or local tasks. In terms of the driving task's endpoint goal  $P_{tar} = (x_{tar}, y_{tar}, \varphi_{tar})$  of path planning and intelligent vehicle posture  $Car_{M^*} = (x_{c|M^*}, y_{c|M^*}, \varphi_{c|M^*})$  in the real environment  $M^*$ , the model transfer strategy  $\mathfrak{R}$  is mapped to  $M$  using Equation (10). Finally, the intelligent parking posture is obtained as  $Car = (x_c, y_c, \varphi_c)$ .

$$\mathfrak{R} : \begin{cases} \theta = \varphi_{tar} - \varphi_{ref} \\ (x', y', \varphi') = (x_{c|M^*} \cos \theta - y_{c|M^*} \sin \theta, x_{c|M^*} \sin \theta + y_{c|M^*} \cos \theta, \varphi_{c|M^*} - \theta) \\ (x_{tar}', y_{tar}', \varphi_{tar}') = (x_{tar} \cos \theta - y_{tar} \sin \theta, x_{tar} \sin \theta + y_{tar} \cos \theta, \varphi_{ref}) \\ (\Delta x, \Delta y, \Delta \varphi) = (x' - x_{tar}', y' - y_{tar}', \theta) \\ (x_c, y_c, \varphi_c) = (x_{ref} + \Delta x, y_{ref} + \Delta y, \varphi_{ref} + \Delta \varphi) \end{cases}, \quad (10)$$

where  $\theta$  is the difference between target heading angle  $\varphi_{tar}$  of the vehicle's driving destination and heading angle  $\varphi_{ref}$  of the reference point. In order to keep the heading angle of the target point  $P_{tar}$  in the real environment coinciding with the heading angle of reference point  $P_{ref}$  in the virtual environment, the real environment coordinate system is rotated by  $(x', y', \varphi')$ , which is the pose corresponding to the ego vehicle in the rotated coordinate system, and  $(x_{tar}', y_{tar}', \varphi_{tar}')$ , which is the pose corresponding to the target point  $P$  in the rotated coordinate system.  $(\Delta x, \Delta y, \Delta \varphi)$  is the difference



in pose between the ego vehicle and the target point in the rotated coordinate system, and  $(x_c, y_c, \varphi_c)$  is the corresponding pose of the ego vehicle in the virtual environment after the model transfer.

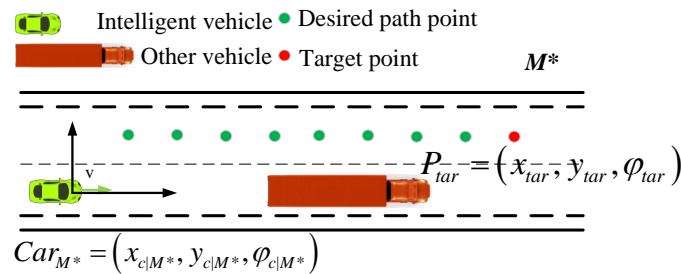
From Figure 2, the virtual environment seems to contain only two curves, but this is inaccurate. After the training is completed, the vehicle can not only complete straight and turning tasks, but also complete lane-changing operations. Trajectories generated in the lane-changing phase contain different curvatures; thus, mapping ramps to the lane-changing phase in stages can solve the problem of ramp driving.

However, the virtual simulation environment does not consider turning around; thus, when the actual road curvature is very large, the proposed method is not applicable.

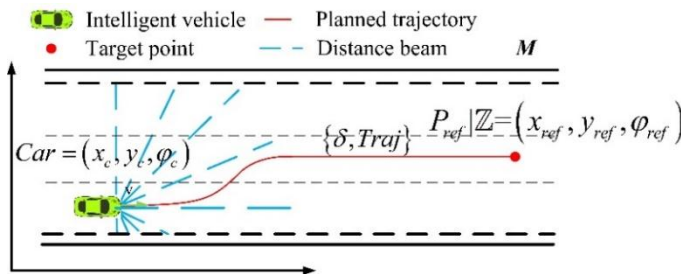
Figure 5a–c show the model transfer process of a lane change. Figure 5a shows the real-world  $M^*$  scene. The road condition information is composed of invariants and variables. The invariants include the number of lanes and the width of lanes. The variables are the information regarding obstacles, which is fed back to the distance beam of the intelligent vehicle. The initial planning process is to travel along the current driving lane. When it is detected that there is a vehicle with lower speed than the ego vehicle ahead of it in the current lane, the left lane is taken as a desired path. The driving task is to switch to the left lane based on behavioral decision planning.

Figure 5d–f show the model transfer process of a ramp. Figure 5d shows the real-world  $M^*$  scene.

The green vehicle is the current vehicle position,  $\mathbf{x} = [x, y, \varphi, v, \omega]^T$ , and the pose information is  $Car_{M^*} = (x, y, \varphi)$ . The green dot in the center of left lane is the scattered point of path planning. Here, the red point is the destination of current driving task  $P_{tar} = (x_{tar}, y_{tar}, \varphi_{tar})$ . Figure 5b,e show the scene after  $Car_{M^*}, P_{tar}$  is migrated to  $\mathfrak{R}$  through the model. Mapping to the virtual environment  $M$ , the intelligent vehicle pose is  $Car = (x_c, y_c, \varphi_c)$ .  $\Sigma(Sensor)$  is acquired according to the distance beam in  $M$ ; then,  $\delta_{old}$  is merged into state  $s$ , and the control sequence  $\delta = \{\delta_1, \delta_2, \dots, \delta_t\}$  and trajectory sequence  $\zeta = \{p_1, p_2, \dots, p_t\}$  are obtained by the model  $M_\theta$ . Figure 5c,f show the corresponding scene of control sequence  $\delta = \{\delta_1, \delta_2, \dots, \delta_t\}$  and track sequence  $\zeta = \{p_1, p_2, \dots, p_t\}$  in the real environment  $M^*$ .



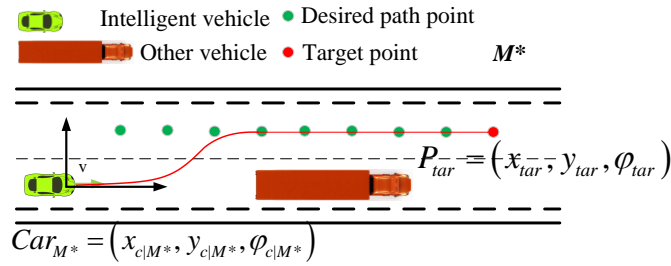
(a) The scene of real environment  $M^*$



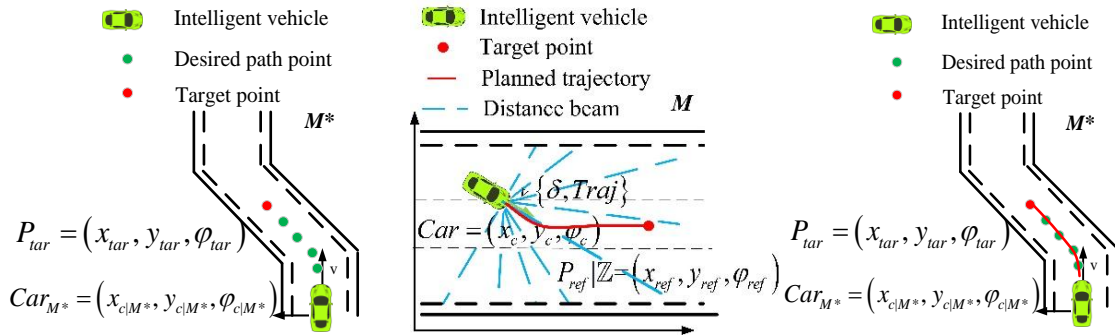
(b) The scene after model transfer  $\mathfrak{R}$

Figure 5. Cont.





(c) The corresponding scene of real environment  $M^*$



(d) The scene of real environment  $M^*$

(e) The scene after model transfer  $\mathfrak{R}$

(f) The corresponding scene of real environment  $M^*$

Figure 5. Transition diagram of the switch task model.

### 3.3. Algorithm Framework of Model Transfer Based on Deep Reinforcement Learning

DRL-MTTP aims to abstract the complex real environment and transfer it to a simple virtual environment through the model. Furthermore, the optimal intelligent driving strategy is applied to the virtual environment, which is trained by the agent after deep reinforcement learning. Thus, the optimal trajectory control sequence is obtained to realize the end-to-end trajectory planning in the real environment. Figure 6 shows a technical diagram of DRL-MTTP.

The framework of DRL-MTTP is shown in Algorithm 2. Firstly, sub-task  $\gamma$  and path-planning datasets are initialized according to upper data streams. Afterward, the trajectory-planning stage begins. Then, an appropriate target point  $P_{target}$  is selected as the local goal based on the sub-task  $\gamma$ . Thus, the selection process is shown in Equation (11).

$$\begin{cases} s_{t-1,t} = \sqrt{(x_t - x_{t-1})^2 + (y_t - y_{t-1})^2} \\ l^2 = \sum_{t=1}^{target} s_{t-1,t} \end{cases}, \quad (11)$$

where  $(x_i, y_i)$  are the corresponding coordinates of  $P_i$ ,  $s_{t-1,t}$  is the straight-line distance between  $P_{t-1}$  and  $P_t$ ,  $l$  is the arc length threshold of path scatter points which is determined by sub-task  $\gamma$ , and  $P_{target}$  is the target point that satisfies the threshold requirement. Subsequently, the target set is obtained by adding noise to the target point  $P_{target}$ , and  $(\varepsilon_x, \varepsilon_y, \varepsilon_\varphi)$  satisfies a Gaussian distribution. Furthermore, the corresponding position  $Car$  of the intelligent vehicle in the virtual environment  $M$  is calculated by a model transfer for each target point  $P_{target}$ . The status of the environment  $s$  and the status of the intelligent vehicle  $x$  are gained by observing its ranging light beam in  $M$ .

During the planning time  $T$ , the deep reinforcement learning model  $M_\theta$  is utilized in each unit of time  $t$  to analyze state  $s$  and predict the action  $\delta_t$ . At the same time, the dynamic model is adopted to simulate the prediction action. Meanwhile, the environment state  $s$  and intelligent vehicle state  $x$  are

updated, and the track  $\zeta_t$  is recorded. Finally, the control-trajectory sequence pair  $C = \{\delta, \zeta\}$  under real environment  $M^*$  is acquired by model transfer.

The second stage is about optimal trajectory selection. In this stage, the evaluation function of each control-trajectory pair is calculated. At the same time, the collision probability of the trajectory  $\zeta$  is also judged. The control-trajectory pair with minimum  $J$  and no collision trajectory are taken as the optimal trajectory.

$$\min J = \kappa_1 \int_0^T (\Delta\delta^2) dt + \kappa_2 [h(\zeta_T) - h(\zeta_{target})], \quad (12)$$

where  $T$  is the termination time,  $\Delta\delta$  is the difference of continuous action,  $\kappa_1, \kappa_2$  is the weight coefficient, and  $h$  represents the rectangular area of the vehicle outline at the end of its trajectory.

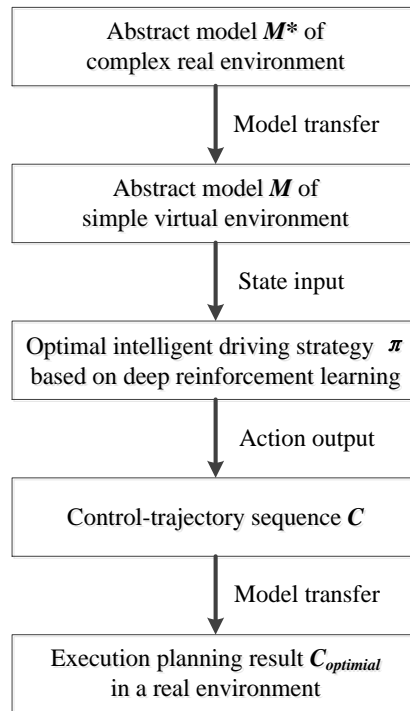
Finally, the result of planning is executed. The intelligent vehicle implements the first  $\tau$  steps of control sequence  $\delta$ , and the model reference control is utilized to enhance the robustness and to reduce the influence of system error by the model error. If the task is not terminated, planning action of the first two phases is repeated to realize the intelligent vehicle dynamic trajectory planning.

---

**Algorithm 2. Model transfer trajectory planning based on deep reinforcement learning (DRL-MTTP) frame.**

---

1. initialize *terminal*,  $S_1 : \{P_1, P_2, \dots, P_m\}$ ; //receive tasks and data from the top
  2. **while** *terminal* = *false*
  3. //trajectory planning stage
  4.  $P_{target} \leftarrow S_1 : \{P_1, P_2, \dots, P_m\}$ ; //select a target from a path planning scatter set  $S_1$  based on a task
  5. **for**  $i = 0, N$  **do**
  6.  $P_{target,i} = P_{target} + \xi_{noise}$ ; //add noise to generate target sets
  7.  $Car_i \xleftarrow{\mathfrak{R}} (P_{target}, P_{ref}, Car_{M^*})$ ; //calculate the corresponding position and pose of the intelligent vehicle by model transfer
  8.  $s, \mathbf{x} \xleftarrow{M} (Car_i, Sensor)$ ; //calculate state based on ranging light beam
  9. **for**  $t = 0, T$  **do**
  10.  $\delta_t \xleftarrow{M_\theta} s$ ; //calculate action by deep reinforcement learning model
  11.  $\mathbf{x} = \int_0^{\Delta t} f(\mathbf{x}, \delta_i)$ ; //dynamic model simulation
  12.  $\zeta_i \leftarrow \mathbf{x}$ ; //record track
  13. **end for**
  14.  $C_i : \{\delta, \zeta^*\} \xleftarrow{\mathfrak{R}} (\delta, Car_i, Car_{M^*}, \zeta)$ ; //calculate control-trajectory sequence pair by model transfer
  15. **end for** //optimal trajectory selection stage
  16.  $J_{min} = \infty$ ;
  17. **for**  $i = 0, N$  **do**
  18.  $J = \kappa_1 \int_0^T (\Delta\delta^2) dt + \kappa_2 [h(\zeta_T) - h(\zeta_{target})]$ ; //calculate and obtain evaluation function
  19. **if**  $J < J_{min}$  **and** no collision //select the control-trajectory pair with minimum  $J$  value and no collision as the optimal trajectory
  20.  $J_{min} = J$ ;
  21.  $C_{optimal} = C_i$ ;
  22. **end if**
  23. **end for**
  24. //stage of execute the planning result
  25.  $\delta_{optimal} : \{\delta_1, \delta_2, \dots, \delta_\tau\}, \tau \leq T$ ; //get the results of the first  $\tau$  steps
  26. update *terminal*; //whether the task ends or not
  27. **end while**
-



**Figure 6.** Block diagram of model transfer trajectory planning based on deep reinforcement learning (DRL-MTTP).

#### 4. Simulation Test on Trajectory Planning of Intelligent Vehicle

##### 4.1. Deep Reinforcement Learning Model Training

In the virtual simulation environment *Env*, the circle map was 100 m long and 50 m wide. The driveway was 3.4 m wide. Here, the speed was  $v = 36$  km/h, which was kept constant. The dimension of the ranging light beam was  $n = 9$ , and the farthest range was  $d_{Max} = 20$  m, while the largest output of angle was  $\delta_{max} = 0.3$  rad. The desired  $path_d$  was the centerline of middle lane. The errors were  $\varepsilon_1 = \varepsilon_2 = 0.1m$ ,  $\varepsilon_3 = 0.5236$ , the continuous action penalty coefficient was  $\lambda_1 = 0.01$ , and the random initial position was  $x_0 \in [10, 980]$ ,  $y_0 \in [10, 50]$ ,  $\varphi_0 \in [-\pi/2, \pi/2]$ . The maximum step number was  $Num_{max} = 600$ , and the step gap was 0.1 s. The software environment was a Linux operating system with 16 Gb of memory, and the graphics card was a GTX1080 Ti (NVIDIA, Santa Clara, CA, USA). The system took advantage of the deep learning framework TensorFlow.

The greater the learning rate is, the lower the effect of previous training being retained. Similarly, the greater the discount factor is, the more emphasis is placed on experience. The smaller the discount factor is, the more attention is paid to the current return. If the numbers of hidden layers and hidden layer neurons are too little, then the data cannot be fitted well. Conversely, if the numbers of hidden layers and hidden layer neurons are too large, this can easily lead to over-fitting. Therefore, a better network structure and network parameters were designed after several trials. The hyper parameters in the deep reinforcement learning model  $M_\theta$  were set as follows: the discount factor was  $\gamma = 0.9$ , the learning rates of the Actor and Critic networks were both  $10^{-4}$ , the optimization method of Adam [32] was adopted, the soft update rate was  $\tau = 0.001$ , the number of hidden layer neurons was 512, the size of the experience replay pool was  $10^4$ , the size of the batch was 64, the error was generated by a Gaussian process, the initial variance was  $var_{max} = 2$ , the minimum variance was  $var_{min} = 0.01$ , and the attenuation rate was  $10^{-4}$ .

At low speed, the vehicle dynamic model can be approximated as a bicycle model with two degrees of freedom [4]. The vehicle dynamics model is described in Equation (13).

$$\begin{cases} \dot{x} = U \cos \varphi - v \sin \varphi \\ \dot{y} = U \sin \varphi + v \cos \varphi \\ \dot{\varphi} = \omega \\ \dot{v} = -\frac{C_f + C_r}{mU} v - \left( \frac{aC_f - bC_r}{mU} + U \right) \omega + \frac{C_f}{m} \delta \\ \dot{\omega} = \frac{bC_r - aC_f}{I_Z U} v - \frac{a^2 C_f + b^2 C_r}{I_Z U} \omega + \frac{aC_f}{I_Z} \delta \end{cases} \quad (13)$$

where  $(x, y)$  is the location of the vehicle,  $\varphi$  is the yaw angle,  $\omega$  is the yaw rate,  $\delta$  is the front-wheel steering angle,  $U$  is the longitudinal velocity, and  $v$  is the lateral velocity. The definitions and values of the vehicle parameters in Equation (13) are shown in Table 1.

The vehicle status is represented by  $\mathbf{x} = [x, y, \varphi, v, \omega]^T$ , and thus, Equation (13) can be expressed as Equation (14).

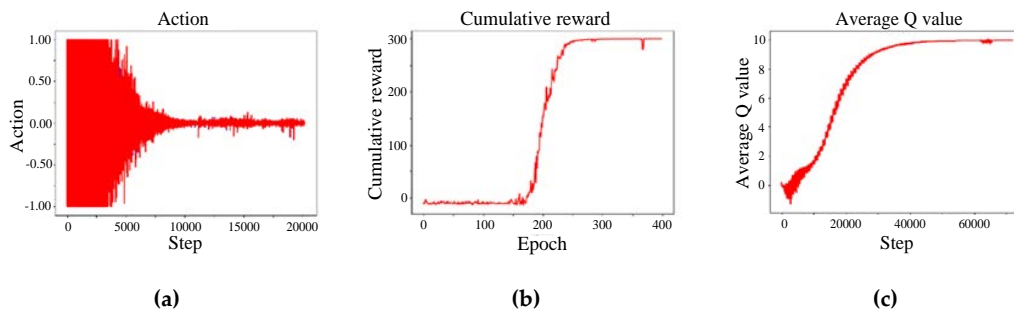
$$\dot{\mathbf{x}} = f(\mathbf{x}, \delta). \quad (14)$$

**Table 1.** Vehicle parameters.

Features	Symbols	Parameters
Complete vehicle kerb mass /kg	$m$	17,800
Length	$l$	11,950
Width	$h$	2540
Vehicle yaw moment of inertia /kg·m <sup>2</sup>	$I_Z$	20,000
Distance from center of mass to front axle /m	$a$	2.795
Distance from center of mass to rear axle /m	$b$	3.105
Wheel base /m	$d$	5.9
Cornering stiffness of front wheel /N·rad <sup>-1</sup>	$C_f$	6500
Cornering stiffness of rear wheel /N·rad <sup>-1</sup>	$C_r$	5200

Figure 7 shows the changes in parameters during training. Figure 7a has an abscissa of “step” and an ordinate of “action”. Figure 7b has an abscissa of “epoch” and an ordinate of “cumulative reward”. Figure 7c has an abscissa of “step” and an ordinate of “average Q-value”. Figure 7d has an abscissa of “step” and an ordinate of “gradient”. Figure 7e has an abscissa of “epoch” and an ordinate of “noise”. Figure 7f has an abscissa of “step” and an ordinate of “loss”.

As shown in Figure 7, the rewards and the average Q-value of the agent gradually increased with the number of iterations during the training process. Finally, it tended to be stable. On the other hand, the loss gradually decreased to 0 with the increase in the number of iterations, indicating the evaluation of the network becoming more and more effective. The noise decreased as the number of iterations increased, providing sufficient exploration capability in the early stages and sufficient exploitation capability in the late stages. Figure 7 shows that the model learned from experience, and continuously approached the optimal strategy.



**Figure 7.** Cont.

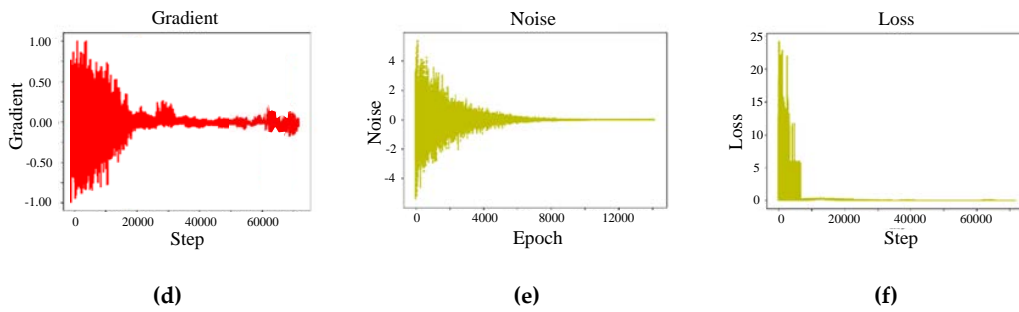


Figure 7. Training results of the deep reinforcement learning model.

4.2. Verification of Intelligent Vehicle Trajectory Planning Based on DRL-MTTP

4.2.1. Trajectory Planning of Lane Keeping

During the simulation test, the intelligent vehicle speed was set as 10 m/s (36 km/h). The intelligent vehicle started from the center position of the middle lane at a 30° yaw angle and a -30° yaw angle, which are shown in Figure 8a,e, respectively. Figure 8b,f have abscissas of “X/m” and ordinates of “Y/m”. Figure 8c,g have abscissas of “time/0.1 s” and ordinates of “heading angle/rad”. Figure 8d,h have abscissas of “time/0.1 s” and ordinates of “steering-wheel angle/°”.

Through the algorithm model, the steering-wheel control sequence is adjusted to keep in its lane. In the beginning, the intelligent vehicle had a deviation from the initial position, and it then corrected itself to keep to the lane. The average value of front-wheel angle across the three experiments was -0.0001635 rad (-0.009367°). Therefore, the steering angle was stable around 0°, with a mean value of -0.1562°. The average lateral deviation after stabilization was 0.04 cm.

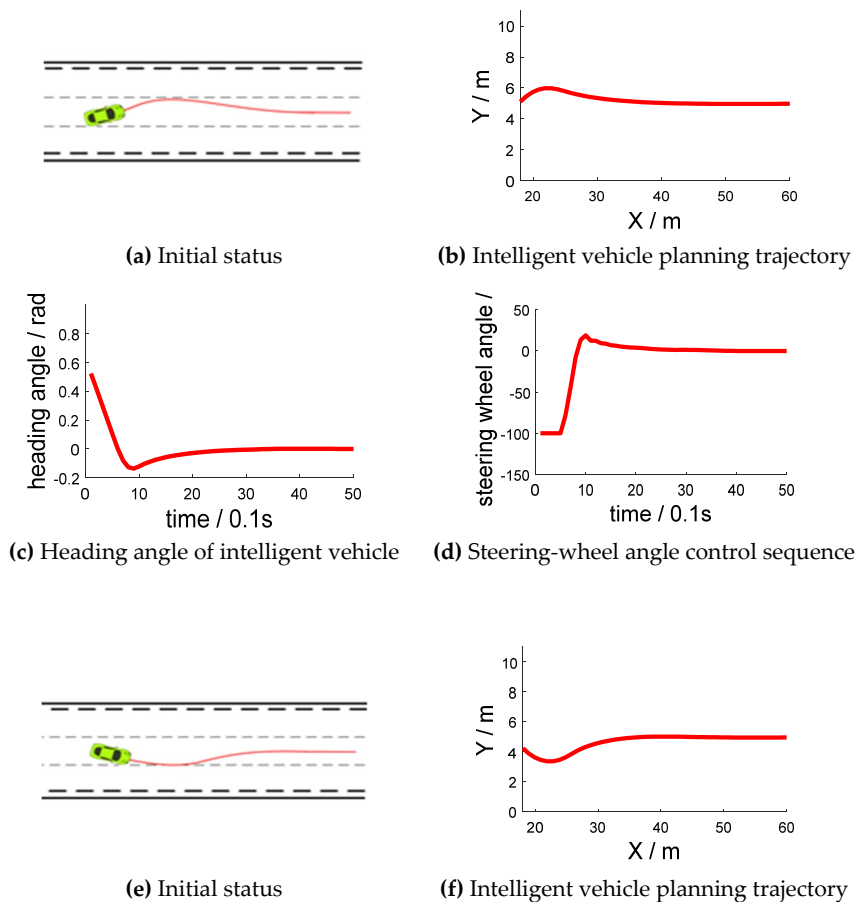


Figure 8. Cont.

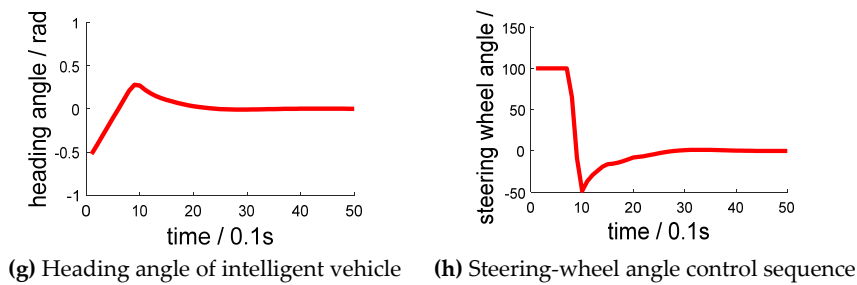


Figure 8. Lane retention experiment.

4.2.2. Curve Track Planning

During the simulation test, the intelligent vehicle set out with different yaw angles, which are shown in Figure 9. Figure 9b,f have abscissas of “X/m” and ordinates of “Y/m”. Figure 9c,g have abscissas of “time/0.1 s” and ordinates of “heading angle/rad”. Figure 9d,h have abscissas of “time/0.1 s” and ordinates of “steering-wheel angle/°”. Figure 9i,j have abscissas of “time point” and ordinates of “lateral error/m”. Figure 9i,j show the lateral error from the center of the lane at each time point. When the vehicle was on the left side of the lane, the lateral error was negative. As shown in Figure 9i, the lateral error was nearly 0 when going straight. When the vehicle turned, the lateral error increased to about 0.2 m. After turning, the lateral error decreased to nearly 0. As shown in Figure 9j, as the initial heading angle was 0.5 rad, and the absolute value of the vehicle lateral error increased first before decreasing to nearly 0. The following trend was the same as that in Figure 9i.

Although there was a shock in the curve, the intelligent vehicle still ultimately navigated the curve successfully. Meanwhile, the curve of the control sequence and the change rate of the yaw angle were relatively smooth.

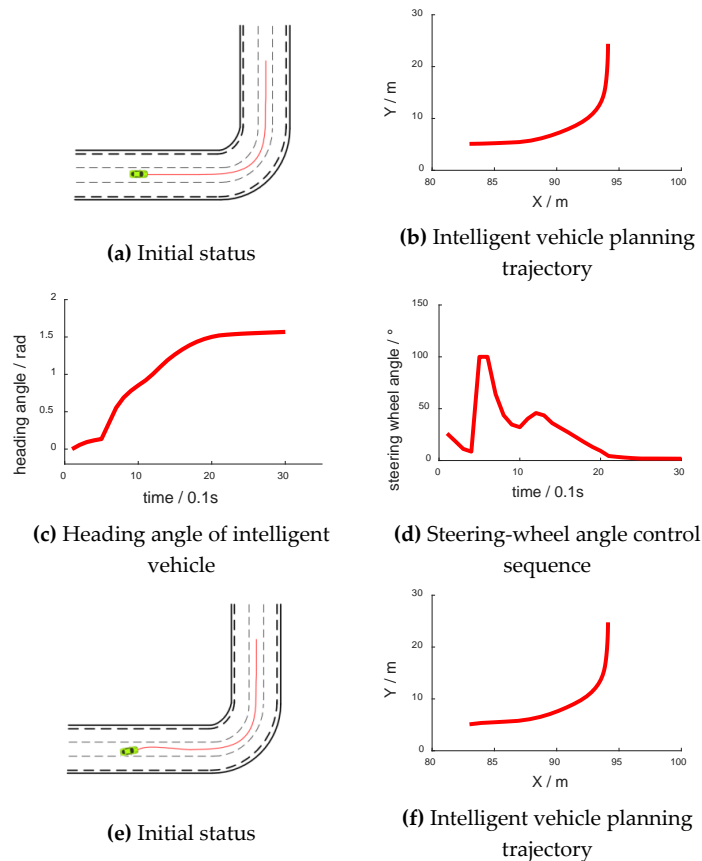
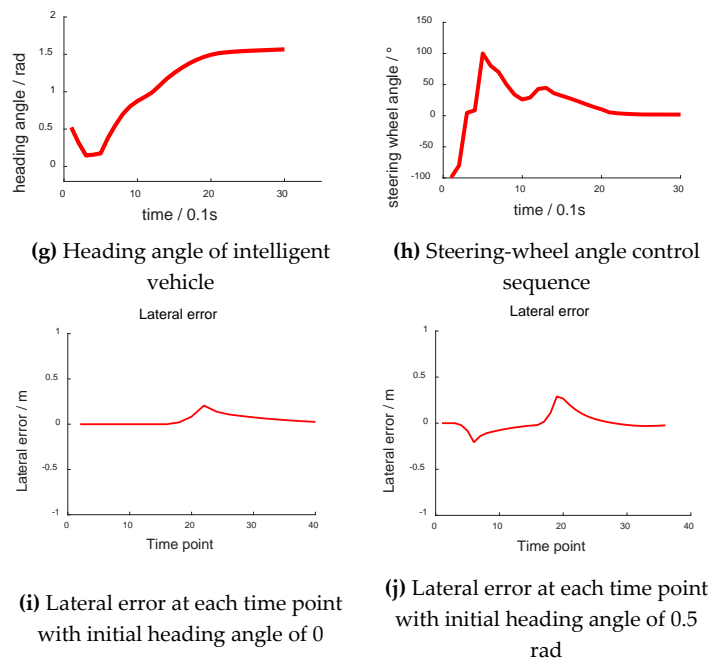


Figure 9. Cont.



**Figure 9.** Curve driving experiment.

#### 4.3. Experimental Comparison and Analysis of the Three Trajectory Planning Methods

Currently, there are three trajectory-planning methods comparisons and experimental analyses, as described in this section. These trajectory-planning methods are called the optimal trajectory-planning method based on a cubic polynomial [33], the end-to-end trajectory-planning method [9,10], and the model transfer trajectory-planning method based on deep reinforcement learning. The intelligent vehicle drove following the planning results in each simulation step with the same tracking control error. Here, the experiments only compare and analyze the performance of the planning. The traditional trajectory-planning method adopts the angle control sequence based on a preview window. However, the end-to-end trajectory planning method outputs the angle control sequence directly. The period of the trajectory planning was 100 ms. In other words, the intelligent vehicle was reprogrammed after keeping the same deflection angle for 100 ms.

##### 4.3.1. Arc Straight Track Scene

The arc shown in Figure 10 had a radius of 300 m; thus, it was large enough to be considered as a straight line in a small range. However, the curvature was not equal to zero. The contrast experiment is shown in Figure 11. Figure 11a–c are the experimental results based on the cubic-polynomial dynamic optimal trajectory-planning method. Figure 11d–f are the experimental results of the end-to-end trajectory-planning method. Figure 11g–i are the experimental results of the model transfer trajectory-planning method based on deep reinforcement learning. Figure 11j shows the lateral error from the center of the lane at each time point. Figure 11a,d,g have abscissas of “X/m” and ordinates of “Y/m”. The blue dotted lines in the figures represent the “expected path” and the red solid lines represent the “actual trajectory”. Figure 11b,e,h have abscissas of “time/0.1 s” and ordinates of “heading angle/rad”. Figure 11c,f,i have abscissas of “time/0.1 s” and ordinates of “steering-wheel angle/°”. Figure 11j has an abscissa of “time point” and an ordinate of “lateral error/m”.

The actual trajectories, represented by the solid line, in the three methods were basically the same as the expected trajectory, represented by the dotted line. However, the rotation-angle control sequence of the first two methods oscillated continuously in the small range. The control sequence of our proposed method showed a periodic and continuous variation. The experimental results verified that the model transfer trajectory-planning method based on deep reinforcement learning performs well, and the control action is continuous when driving nearly straight. As shown in Figure 11j, when



adopting the optimal trajectory-planning method based on the cubic polynomial, the trajectory of the vehicle was prone to oscillation. There was a medium mean lateral error when adopting the end-to-end trajectory-planning method, while there was a minimum mean lateral error when adopting the method based on MTTP/DDPG.

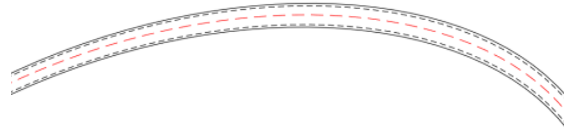


Figure 10. Arc straight track scene diagram.

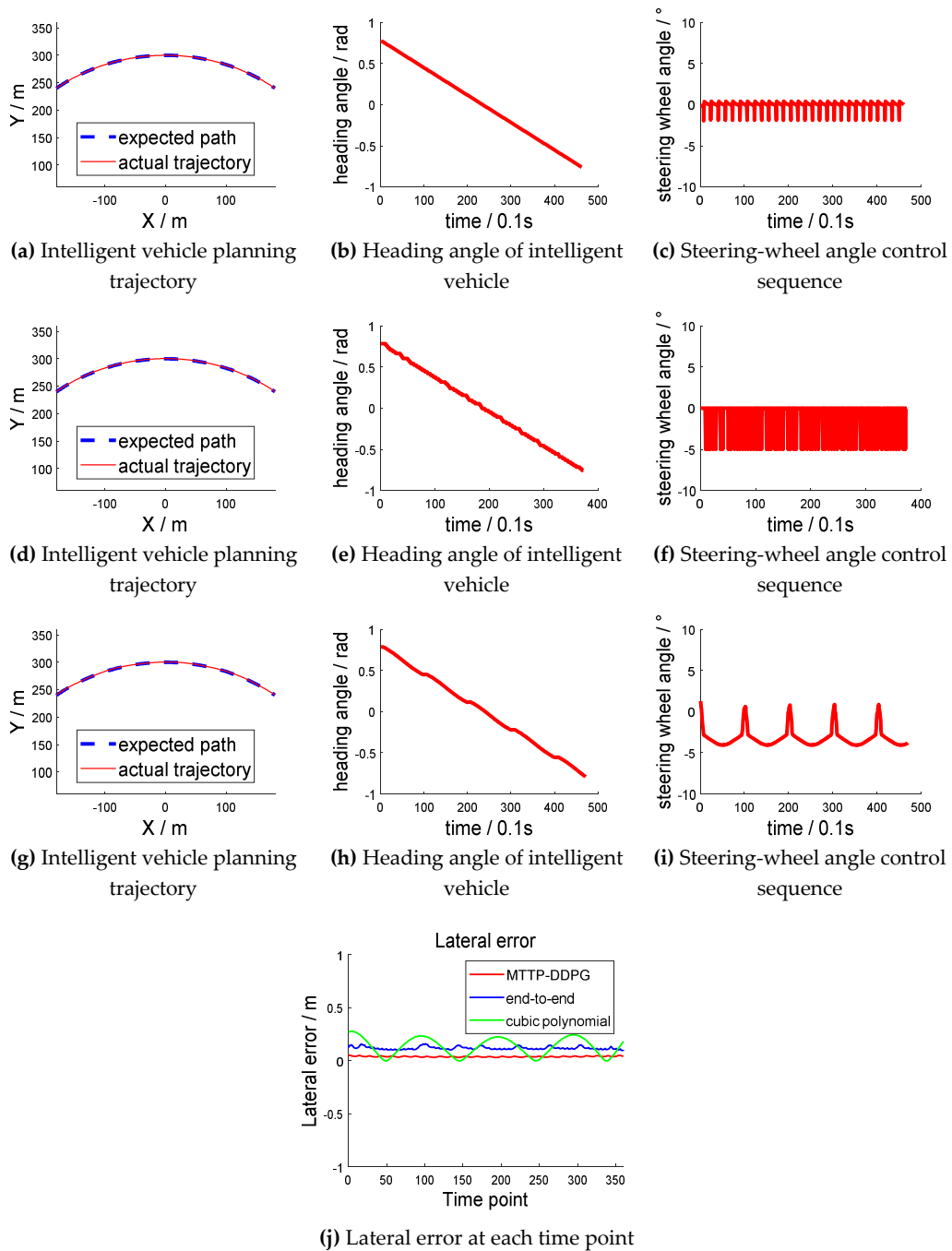


Figure 11. Contrast experiment of arc straight track scene.

### 4.3.2. S-type Ramp Scene

Figure 12 is the schematic diagram of the S ramp. The intelligent vehicle was initially situated in the left lane of the lower right corner. After driving at a yaw angle from the north, the vehicle turned left and entered a 45° ramp. After driving a distance, it turned right in the middle lane and completed the intelligent driving maneuver. The experiment results are shown in Figure 13. Figure 13a–c are the experimental results based on the cubic polynomial dynamic optimal trajectory planning. Figure 13d–f are the experimental results of the end-to-end trajectory planning. Figure 13g–i are the experimental results of the model transfer trajectory-planning method based on deep reinforcement learning. Figure 13j shows the lateral error from the center of the lane at each time point. The abscissas of Figure 13a,d,g are “X/m”, and the ordinates are “Y/m”. The blue dotted line in Figure 13a represents the “expected path” and the red solid line represents the “actual trajectory”. Figure 13b,e,h have abscissas of “time/0.1 s” and ordinates of “heading angle/rad”. The abscissas of Figure 13c,f,i are “time/0.1 s”, and their ordinates are “steering-wheel angle/°”. Figure 13j has an abscissa of “time point” and an ordinate of “lateral error/m”.

According to the comparison results between actual trajectory and desired path based on the three planning methods, the results show that the proposed method allowed the intelligent vehicle to maintain a better turning performance, especially in the combination of straight and curved roads. Finally, the lateral deviation of our proposed method was the least, and the it outperformed the other two methods. When entering the ramp or leaving the ramp, there was a greater lateral error. In general, the average lateral error adopting the MTTP/DDPG method was the minimum.

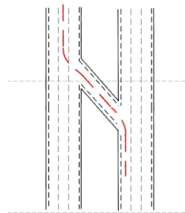


Figure 12. Schematic diagram of the S ramp.

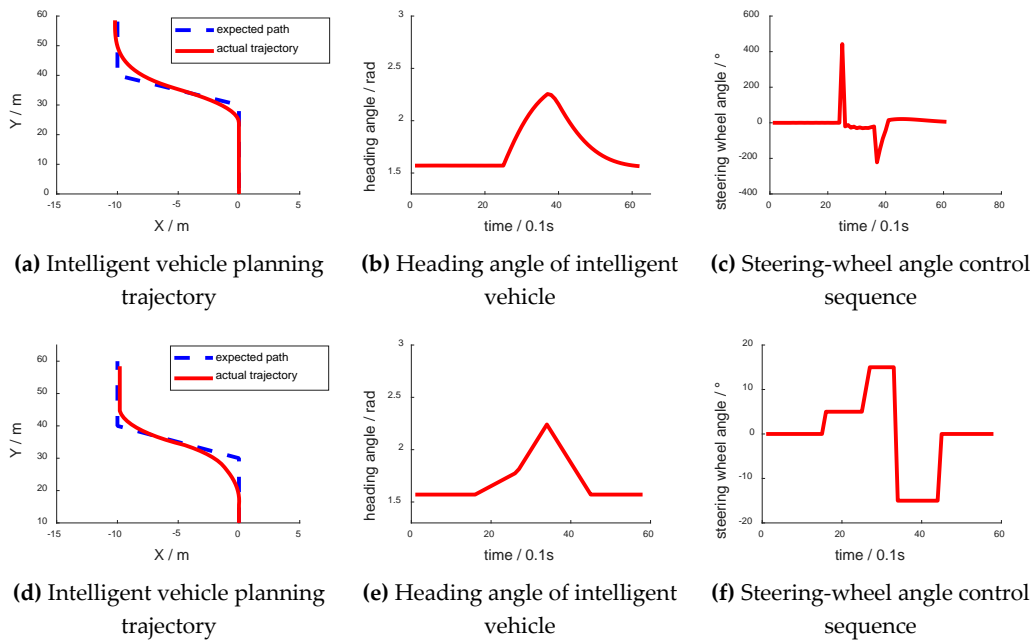


Figure 13. Cont.

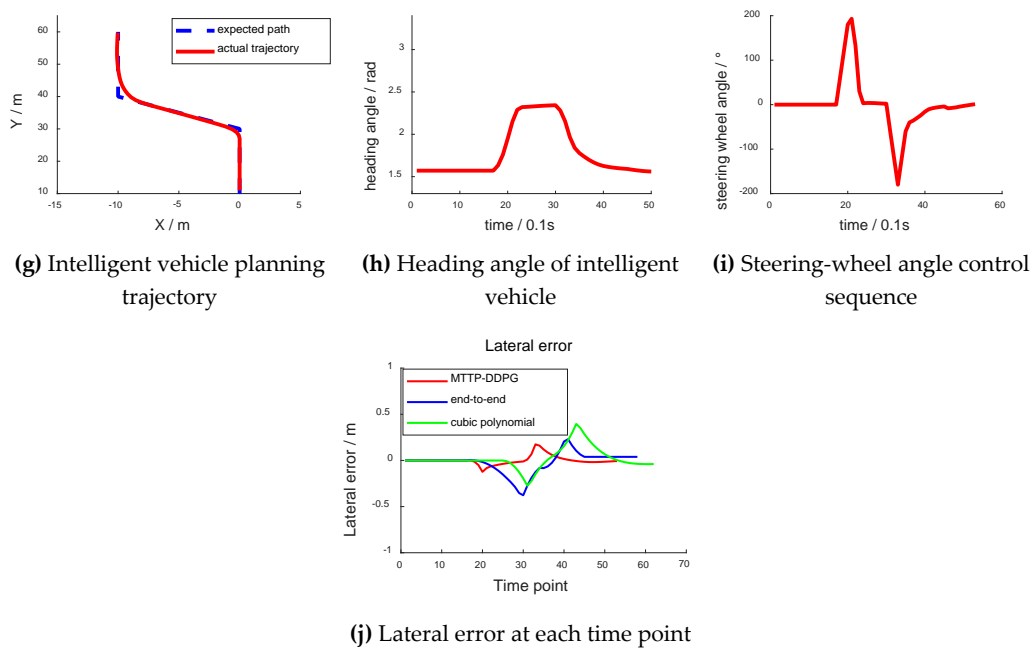


Figure 13. Contrast experiment of S-type ramp scene.

## 5. Real Vehicle Verification of Dynamic Trajectory Planning

The vehicle's real-time position was collected using the GPS (Global Positioning System) and IMU (Inertial Measurement Unit). The sensory data of the surroundings were acquired using a camera, lidar, and millimeter-wave radar, which were mounted on the driverless vehicle.

The camera was adopted to detect lane lines. The identification of obstacles depended mainly on the radar and lidar. The fusion process of obstacle information detected by multiple sensors was as follows: firstly, each sensor extracted obstacles accordingly. Then, the Hungarian algorithm (HA) [34] was adopted to match each object extracted by each sensor. Finally, the matched data were fused with the Kalman filter (KF) [35] to get the fusion data.

The lane's centerline was taken as the desired path based on the lane line detected by the camera. Different points on the desired path were taken as  $P_{target}$ . After the model transfer strategy, a set of trajectories was obtained. Considering the constraints of the fused obstacle information, the optimal trajectory with no collisions was selected according to the cost function.

Figure 14 shows the actual vehicle's dynamic trajectory planning process. Figure 14a–i were captured when the vehicle was turning in different scenes. Here, Figure 14a–f are the outside view, while Figure 14g–i are the inside view of the driverless vehicle. Figure 14 shows that the driverless performance of the intelligent vehicle was stable and efficient when turning around  $90^\circ$ . Figure 14j–o were captured when the driverless vehicle was overtaking and lane changing. Figure 14j–l show the outside view and Figure 14m–o show the internal view. They indicate that the intelligent vehicle achieved self-overtaking and lane-changing driving behaviors safely and stably. Figure 14p–r are real-time screenshots and the interface of the intelligent driving trajectory planning. Figure 14r is the tracking result when the actual vehicle verified the curve driving.



(a) Curve (stage 1)



(b) Curve (stage 2)



(c) Curve (stage 3)



(d) U-turn (stage 1)



(e) U-turn (stage 2)



(f) U-turn (stage 3)



(g) Curve (stage 1)



(h) Curve (stage 2)



(i) Curve (stage 3)



(j) Change lane (stage 1)



(k) Change lane (stage 2)



(l) Change lane (stage 3)



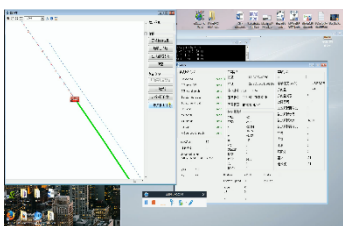
(m) Change lane (stage 1)



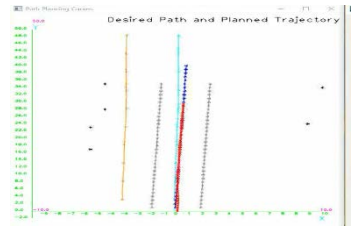
(n) Change lane (stage 2)



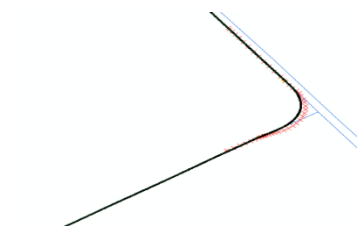
(o) Change lane (stage 3)



(p) GUI - 1



(q) GUI - 2



(r) GUI - 3

Figure 14. Experimental results of the real vehicle.

## 6. Conclusions

It is difficult for traditional trajectory-planning method to eliminate errors due to vehicle models and road conditions. Furthermore, there are no vehicle dynamics constraints. A model transfer trajectory-planning method based on deep reinforcement learning was proposed in this paper. At first, the complex real environment was abstracted using MTTP, then the abstracted model was transferred into a simple virtual environment through the transfer model. Secondly, the optimal intelligent driving maneuver after deep reinforcement learning training was applied to obtain the optimal control-trajectory sequence in the virtual environment. Thereby, the end-to-end trajectory planning of the intelligent vehicle in a real environment was realized. Moreover, an evaluation function was designed to estimate the planning validity of the control-trajectory sequences, and to judge the risk of collision in a real environment. Furthermore, an optimal control-trajectory sequence was decided and executed by the intelligent land vehicle. Finally, the comparison analysis of multiple driving scenes and multiple trajectory-planning methods verified the better optimization performance of MTTP, showing that it achieved a more continuous rotation-angle control sequence and a smaller lateral error for the intelligent land vehicle. However, the speed of the vehicles was assumed as a constant, and that they were driving in a typical structured environment. Because the virtual simulation environment in this paper did not consider turning around, the proposed method is not applicable when the actual road curvature is very large. The next stage will be to further consider variable-speed driving and more complex environments.

**Author Contributions:** Conceptualization, X.S. and Y.W. Methodology, X.S. Software, X.S. and Y.W. Writing—original draft preparation, Y.W. Writing—review and editing, L.Y. and K.Z. Supervision, L.Y. and K.Z.

**Funding:** This research was funded by the Major Projects of Science and Technology in Hunan (Grant No.2017GK1010), the National Key Research and Development Plan (Grant No. 2018YFB1201602), the State Key Laboratory of Mechanical Transmissions of Chongqing University (SKLMT-KFKT-201602), the State Key Laboratory of Robotics and System (HIT) (SKLRS-2017-KF-13), the National Natural Science Foundation of China (Grant No.61403426), National Natural Science Foundation of Hunan (Grant No.2018JJ2531, 2018JJ2197), and the Fundamental Research Funds for the Central Universities of Central South University (Grant No.2018zzts557).

**Conflicts of Interest:** The authors declare no conflict of interest.

## Nomenclature

VAE	Variational auto-encoder
GAN	Generative adversarial network
end-to-end	Algorithm for inputting original data and outputting result
DQN	Deep Q network
OU	Ornstein–Uhlenbeck
DRL	Deep reinforcement learning
MC	Monte Carlo
TD	Temporal difference
AC	Actor-Critic
DDPG	Deep deterministic policy gradient
MTTP	Model transfer trajectory planning
BN	Batch normalization
HA	Hungarian algorithm
KF	Kalman filter

## References

1. Wei, K.; Ren, B. A Method on Dynamic Path Planning for Robotic Manipulator Autonomous Obstacle Avoidance Based on an Improved RRT Algorithm. *Sensors* **2018**, *18*, 571. [[CrossRef](#)] [[PubMed](#)]
2. Coombes, M.; Fletcher, T.; Chen, W.H.; Liu, C. Optimal Polygon Decomposition for UAV Survey Coverage Path Planning in Wind. *Sensors* **2018**, *18*, 2132. [[CrossRef](#)] [[PubMed](#)]



3. Rastelli, J.P.; Lattarulo, R.; Nashashibi, F. Dynamic trajectory generation using continuous-curvature algorithms for door to door assistance vehicles. In Proceedings of the IEEE Intelligent Vehicles Symposium, Dearborn, MI, USA, 8–11 June 2014; pp. 510–515.
4. Cong, Y.; Sawodny, O.; Chen, H.; Zimmermann, J.; Lutz, A. Motion planning for an autonomous vehicle driving on motorways by using flatness properties. In Proceedings of the IEEE International Conference on Control Applications, Yokohama, Japan, 8–10 September 2010; pp. 908–913.
5. Yu, L.; Long, Z.; Zhou, K. Non-time trajectory tracking method based on Bezier curve for robot. *Chin. J. Sci. Instrum.* **2016**. [[CrossRef](#)]
6. Sahingoz, O.K. Generation of Bezier Curve-Based Flyable Trajectories for Multi-UAV Systems with Parallel Genetic Algorithm. *J. Intell. Rob. Syst.* **2014**, *74*, 499–511. [[CrossRef](#)]
7. Silver, D.; Huang, A.; Maddison, C.J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. Mastering the game of Go with deep neural networks and tree search. *Nature* **2016**, *529*, 484–489. [[CrossRef](#)] [[PubMed](#)]
8. Santana, E.; Hotz, G. Learning a Driving Simulator. *arXiv* **2016**, arXiv:1608.01230.
9. Paxton, C.; Raman, V.; Hager, G.D.; Kobilarov, M. Combining Neural Networks and Tree Search for Task and Motion Planning in Challenging Environments. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 6059–6066.
10. Pfeiffer, M.; Schaeuble, M.; Nieto, J.; Siegwart, R.; Cadena, C. From Perception to Decision: A Data-driven Approach to End-to-end Motion Planning for Autonomous Ground Robots. In Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 29 May–3 June 2017; pp. 1527–1533.
11. Liu, W.; Li, Z.; Li, L.; Wang, F.Y. Parking Like a Human: A Direct Trajectory Planning Solution. *IEEE Trans. Intell. Transp. Syst.* **2017**, *18*, 3388–3397. [[CrossRef](#)]
12. Lin, Y.L.; Li, L.; Dai, X.Y.; Zheng, N.N.; Wang, F.Y. Master general parking skill via deep learning. In Proceedings of the 2017 IEEE Intelligent Vehicles Symposium, Los Angeles, CA, USA, 11–14 June 2017; pp. 941–946.
13. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529. [[CrossRef](#)] [[PubMed](#)]
14. 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.
15. Gu, S.; Lillicrap, T.; Sutskever, I.; Levine, S. Continuous deep Q-learning with model-based acceleration. In Proceedings of the International Conference on International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 2829–2838.
16. Schaul, T.; Horgan, D.; Gregor, K. Universal value function approximators. In Proceedings of the International Conference on Machine Learning, Lille, France, 7–9 July 2015; pp. 1312–1320.
17. Metz, L.; Ibarz, J.; Jaitly, N.; Davidson, J. Discrete sequential prediction of continuous actions for deep RL. *arXiv* **2017**, arXiv:1705.05035.
18. Schaul, T.; Quan, J.; Antonoglou, I.; Silver, D. Prioritized experience replay. *arXiv* **2015**, arXiv:1511.05952.
19. Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, O.P.; Zaremba, W. Hindsight experience replay. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 5048–5058.
20. Uhlenbeck, G.E.; Ornstein, L.S. On the Theory of the Brownian Motion. *Phys. Rev.* **1930**, *17*, 323–342. [[CrossRef](#)]
21. Plappert, M.; Houthoofd, R.; Dhariwal, P.; Sidor, S.; Chen, R.Y.; Chen, X.; Asfour, T.; Abbeel, P.; Andrychowicz, M. Parameter space noise for exploration. *arXiv* **2017**, arXiv:1706.01905.
22. Gu, S.; Holly, E.; Lillicrap, T.; Levine, S. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 29 May–3 June 2017; pp. 3389–3396.
23. Genders, W.; Razavi, S. Using a deep reinforcement learning agent for traffic signal control. *arXiv* **2016**, arXiv:1611.01142.
24. Isele, D.; Rahimi, R.; Cosgun, A.; Subramanian, K.; Fujimura, K. Navigating Occluded Intersections with Autonomous Vehicles using Deep Reinforcement Learning. *arXiv* **2017**, arXiv:1705.01196.
25. Tai, L.; Paolo, G.; Liu, M. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 31–36.

26. Mirowski, P.; Pascanu, R.; Viola, F.; Soyer, H.; Ballard, A.J.; Banino, A.; Denil, M.; Goroshin, R.; Sifre, L.; Kavukcuoglu, K.; Kumaran, D. Learning to navigate in complex environments. *arXiv* **2016**, arXiv:1611.03673.
27. Abseher, M.; Dusberger, F.; Musliu, N.; Woltran, S. Improving the efficiency of dynamic programming on tree decompositions via machine learning. In Proceedings of the 24th International Conference on Artificial Intelligence, Buenos Aires, Argentina, 25–31 July 2015; AAAI Press: Palo Alto, CA, USA, 2015; pp. 275–282.
28. Van, R.D.; Cassey, P.; Brown, S.D. A simple introduction to Markov Chain Monte-Carlo sampling. *Psychon. Bull. Rev.* **2018**, *25*, 1–12.
29. Foster, D.J.; Morris, R.G.M.; Dayan, P. A model of hippocampally dependent navigation, using the temporal difference learning rule. *Hippocampus* **2015**, *10*, 1–16. [[CrossRef](#)]
30. Bahdanau, D.; Brakel, P.; Xu, K.; Goyal, A.; Lowe, G.; Pineau, J.; Courville, A.; Bengio, Y. An Actor-Critic Algorithm for Sequence Prediction. *arXiv* **2016**, arXiv:1607.07086.
31. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *Comput. Sci.* **2015**, *8*, A187.
32. Kinga, D.; Adam, J.B. A method for stochastic optimization. *arXiv* **2017**, arXiv:1412.6980.
33. Pérez, J.; Godoy, J.; Villagra, J.; Onieva, E. Trajectory generator for autonomous vehicles in urban environments. In Proceedings of the 2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, 6–10 May 2013; pp. 409–414.
34. Mills-Tettey, A.; Stent, A.; Dias, M.B. *The Dynamic Hungarian Algorithm for the Assignment Problem with Changing Costs*; Technical Report; Carnegie Mellon University: Pittsburgh, PA, USA, 2007.
35. Felipe, E.; Carlos, S.; Marta, M.R.; Pizarro, D.; Valdés, F.; Dongil, J. Odometry and Laser Scanner Fusion Based on a Discrete Extended Kalman Filter for Robotic Platooning Guidance. *Sensors* **2011**, *11*, 8339–8357. [[CrossRef](#)]



© 2018 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 (<http://creativecommons.org/licenses/by/4.0/>).