Article | Received 20 June 2024; Accepted 10 December 2024; Published 24 December 2024
https://doi.org/10.55092/aias20240009

Deep reinforcement learning Lane-Change Decision-Making for autonomous vehicles based on motion primitives library in hierarchical action space

Guizhe Jin[1], Zhuoren Li[1], Bo Leng[1,*] and Minhua Shao[2]

[1] School of Automotive Studies, Tongji University, Shanghai, China
[2] Key Laboratory of Road and Traffic Engineering of the Ministry of Education, Tongji University, Shanghai, China

* Correspondence author; E-mail: lengbo@tongji.edu.cn.

Abstract: Deep Reinforcement Learning (DRL) is capable of learning a policy with great scene adaptation ability through interactions with the environment, and has application potential in the field of autonomous driving. However, using DRL to directly control the vehicle motion command is easy to lead to fluctuation and non-smoothness. Discrete DRL decision-making method can generate stable behavior but lose some maneuverability. Thus, the trade-off between flexibility and stability to enhance the performance of DRL policy is an important issue. In this paper, a Deep Reinforcement Learning Decision-Making algorithm based on Motion Primitives Library (MPL) in hierarchical action space is proposed to provide flexible and reliable maneuvers for autonomous driving. The upper action space contains discrete lane-changing targets, and the lower action space is mapped as a motion primitives library. In addition, model predictive control method (MPC) based on the vehicle kinematics model is used to optimize the motion primitives instantly. The performance of the proposed method is evaluated through highway simulation. The results show that the method can make the autonomous driving lane changing process safer, more efficient and more comfortable.

Keywords: artificial intelligence; autonomous vehicles; machine learning; reinforcement learning; motion planning

1. Introduction

Autonomous driving technology is rapidly evolving and can bring a safer, more efficient, environmentally friendly and comfortable mode of transportation to society. Traditional autonomous driving approaches rely on mechanism models to realize maneuver functions through pipeline framework including perception, prediction, decision making, trajectory planning and control [1–3]. However, these approaches suffer from problems such as lack of flexibility in complex environments and high-cost algorithm maintenance [4]. In recent years, learning-based methods have been applied to autonomous driving systems [5]. Consequently, it can facilitate vehicles' capacity to adapt more effectively to complex and evolving traffic environments with large amount data training.

Reinforcement learning (RL) is a significant branch of machine learning, which is distinguished by many advantages including high control flexibility, straightforward maintenance of algorithms. It is attracting increasing attention and emphasis from researchers engaged in autonomous driving [6,7]. RL applications typically involve the use of deep neural networks and are therefore often referred to as Deep Reinforcement Learning (DRL). Based on the designed

Copyright©2024 by the authors. Published by ELS Publishing. This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium provided the original work is properly cited

rewards, the DRL agent can interact with the traffic environment through trial-and-error to achieve self-evolution of its policy, thus enabling strong scene adaptation capability [8,9]. Therefore, it has been applied in many autonomous driving scenarios such as lane-changing, intersection, merging, etc. [10–12].

However, some current DRL-based methods may still affect the smoothness or maneuverability of the vehicle. (1) Some studies have focused on developing the high maneuver capability of the DRL agent: researchers attempted to enable the agent to directly control the vehicle's steering angle and accelerator pedal to emulate human driving behaviors [13,14]. However, these methods outputting continuous actions are randomly selected from a mathematical distribution, resulting in continuous shaking of vehicle's steering angle and acceleration, which has a negative effect on driving stability and safety. (2) Some other studies are more conservative, which only use the DRL agent to assist decision-making: here, DRL is used to solve high-level semantic decision-making problems in autonomous driving scenarios, such as whether to keep the lane or how to change the lane. Based on the high-level decision-making information, traditional mechanism model-based planners and controllers are then designed to generate motion commands, such as PID [15,16]. Nevertheless, in complex traffic scenarios, the lack of flexibility in trajectory planning and control may reduce the driving efficiency and even make the vehicle in a dangerous situation.

Therefore, it is necessary to both improve the flexibility and stablity performance by involving the DRL agent in vehicle motion planning with a more reasonable degree. Some researchers have made attempts. In reference [17], the DRL agent is used to output the expected time of the trajectory and the lateral offset distance, and the trajectory data obtained on the basis of these parameters can be used to train a neural network model that outputs the optimal trajectory of the vehicle driving process. However, this method requires a neural network model to be trained offline first, which has poor transferability when dealing with different traffic scenarios. Reference [18] allow agents to participate in trajectory planning based on the DDQN algorithm in a hierarchical architecture, where the upper layer outputs discrete task actions and the lower layer selects the type of trajectory. However, this method not only does not consider the speed decision, but also has limitations on the types of trajectories that the agent can select. Based on the DDPG algorithm, reference [19] let the agent output the target point and target speed of the trajectory, and then track it with a PID controller. However, the target point of this method is selected in a planar continuous region, and the computation is time-consuming. In reference [20], a hybrid system is designed, consisting of a controlled MDP and uncontrolled continuous dynamics, but the trajectory guidance in this system lacks flexibility. In conclusion, the existing research results still cannot enable autonomous vehicles to use DRL for proper motion planning.

To address the shortcomings of existing studies, this paper proposes a reinforcement learning motion planning method based on a Motion Primitive Library (MPL) in hierarchical action space. The upper action space of this method contains discrete high-level semantic decision options. For each high-level option, there are corresponding number of motion primitives in the lower action space for further selection by the DRL agent. In Section 3.2 we explain the hierarchical action space, where we discuss the relationship between high-level actions, low-level motion primitives, and the vehicle's driving behavior. Then, an objective function will be constructed based on the vehicle kinematic model to optimize the motion primitive that will further improve its smoothness. The proposed method is called RL-MPL, which can effectively improve the driving efficiency and safety of the vehicle in lane changing scenarios. The specific contributions are summarized as follows:

- Propose a deep reinforcement learning framework based on motion primitives library, which realizes the selection of motion primitives through hierarchical action space. It can fully utilize the flexibility of the DRL agent while enhancing driving stability.
- Model Predictive Control (MPC) based on vehicle kinematics model is used to achieve

the optimization of motion primitives, enhancing the smoothness of vehicle driving.
- Combine the SAC-Discrete algorithm with the motion primitives library and conduct experiments in a straight structured simulation road. The results show that the method proposed in this paper can realize safer and more efficient driving.

The rest of the paper is organized as follows: Section 2 introduces the RL fundamentals and shows the overall system framework of RL-MPL. Section 3 presents the key details of the DRL agent. Section 4 provides the development of motion primitives and the way they combine with the RL method. In Section 5, simulation validation and analysis of experimental results are provided. Section 6 concludes this paper.

2. System framework

This section will introduce the fundamentals about RL and then further introduce the SAC-Discrete algorithm. Then, the overall framework of the proposed RL-MPL method will be shown and explained.

2.1. Preliminary

2.1.1. Markov decision processes

The Markov Decision Processes (MDPs) is meant to be a straightforward framing of the problem of learning from interaction to achieve a goal [21]. The learner and decision maker is called the agent, which choose actions based on their observed state. The environment reacts to the agent's actions and feeds new observations and rewards back to the agent. The agent is able to gain experience from the constant interaction and modify its strategy based on that experience to gain greater rewards.

An MDP can be represented by a tuple $< S, A, P, R, \gamma >$, where S is the state space, A is the action space, P is the state transition probability, R is the reward function, and γ is the discount factor. Specifically, the agent and the environment interact at each of a series of discrete time steps. At each time step t, the agent receives a representation of the environment's state s_t, $s_t \in S$. Then the agent is able to select an action a_t, $a_t \in A$, based on s_t. After a time step, as a partial result of its action, the agent is able to receive a numerical reward r_t, $r_t \in R$, and finds itself in a new state s_{t+1}. In this way, the cycle is able to generate experience for the agent, which is the basis for training. The goal of agent is to find the optimal a policy maximizing the expected reward. This optimal action-value function can be described by the Bellman equation:

$$Q(s_t, a_t) = \mathbb{E}\left[r_t + \gamma \max_{a' \in A} Q(s_{t+1}, a') | s_t = s, a_t = a \right] \tag{1}$$

2.1.2. Maximum entropy RL

Traditional RL methods rely on tabular methods or simple function approximations, which are ineffective when dealing with high dimensionality and complex state spaces. Therefore, neural networks are introduced into reinforcement learning. By utilizing the powerful expressive ability of neural networks, arbitrarily complex value functions and policy functions can be approximated. A widely used architecture in DRL is Actor-Critic. Critic is responsible for estimating the value function of the action and guiding the optimization of the behavior. In turn, Actor is responsible for generating a policy to select an action based on the current state.

The algorithm that applies maximum entropy in the Actor-Critic architecture is Soft Actor-Critic (SAC) [22], which has the advantage of being efficient and stable. The agent of SAC attempts to find a policy that maximizes the entropy objective:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_t r(s_t, a_t) + \alpha H(\pi(\cdot | s_t)) \right] \tag{2}$$

where π is a policy and π^* is the optimal policy. α is a regularized coefficient that determines the importance of the entropy term with respect to the reward, also called the temperature parameter. H denotes the degree of stochasticity of the policy π in state s_t, which is computed as:

$$H(\pi(\cdot\,|s_t)) = -\log \pi(\cdot\,|s_t) \tag{3}$$

Upon consideration of the maximum entropy, the following soft Bellman equation can be derived:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}}\left[V(s_{t+1})\right] \tag{4}$$

where the state value function can be written as:

$$V(s_t) = \mathbb{E}_{a_t \sim \pi}\left[Q(s_t, a_t) - \alpha \log \pi(a_t\,|s_t)\right] \tag{5}$$

In SAC implementation, two Q-networks (with parameters ω_1, ω_2) are used to output the action value function. Each time a Q-network is employed, the one with the smaller output value is selected in order to alleviate the problem of overestimation of the Q value [1]. Additionally, a policy network π is utilized to output the agent's action with network parameters of θ. Consequently, the loss function of any Q-network can be written as:

$$l_Q(\omega) = \mathbb{E}\left[\frac{1}{2}\left(Q_\omega(s_t, a_t) - \left(r_t + \gamma\left(\min_{j=1,2} Q_{\omega_j^-}(s_{t+1}, \pi(s_{t+1})) - \alpha \log \pi(s_{t+1})\right)\right)^2\right] \tag{6}$$

where D is the experience replay pool. In order to enhance the stability of the training process, target networks are introduced for the two Q networks with parameter ω^-.

To obtain the loss function of the policy network, SAC uses the reparameterization trick [1]. During policy updating, a_t can be obtained by $f_\theta(\varepsilon_t; s_t)$, where ε is a random noise variable. So, the loss function of the policy network can be written as:

$$l_\pi(\theta) = \mathbb{E}_{s_t \sim D, \varepsilon_t \sim N}\left[\alpha \log(\pi_\theta(f_\theta(\varepsilon_t; s_t)\,|s_t)) - \min_{j=1,2} Q_{\omega_j}(s_t, f_\theta(\varepsilon_t; s_t))\right] \tag{7}$$

It is worth noting that how the temperature coefficient α of the entropy regularity term is chosen is important. So during the training process of SAC, α will change as the state s changes. Reference [2] provides a method to learn the temperature coefficient α and get the loss function of α:

$$l(\alpha) = \mathbb{E}_{s_t \sim D}\left[-\alpha \log \pi_\theta(s_t) - \alpha H_0\right] \tag{8}$$

where H_0 is a constant vector equal to the hyperparameter representing the target entropy.

2.2. SAC discrete action space

SAC is a kind of algorithm for continuous action space, with a policy network that can directly output a parameterized family of distributions (e.g. Gaussian distribution). In contrast, the MPL proposed in this paper is designed under discrete action space. In order to make SAC capable of outputting discrete actions, some changes must be made in the process of optimizing the objective function [23].

Since the action space is discrete, we can fully recover the action distribution without forming a Monte Carlo estimate. Therefore, the expectation of the state value function can be calculated directly, and Equation (5) needs to be modified as:

$$V(s_t) = \pi_\theta(s_t)^T\left[Q_\omega(s_t, a_t) - \alpha \log \pi_\theta(s_t)\right] \tag{9}$$

Similarly, modifications can be made to the loss function calculation for the temperature coefficient α in a manner analogous to that described above in order to reduce the variance of this estimate. Equation (8) can be modified as:

$$l(\alpha) = \pi_\theta(s_t)^T\left[-\alpha \log \pi_\theta(s_t) - \alpha H_0\right] \tag{10}$$

In order to minimise the loss function of the policy network, SAC employs a reparameterisation trick, which allows the gradient to pass through the expectation operator. However, in the case of a discrete action space, the policy network outputs the exact action distribution, allowing the expectation to be computed directly. Consequently, Equation (7) can be modified as:

$$l_\pi(\theta) = \mathbb{E}_{(s_t,a_t)\sim D}\left[\pi_\theta(s_t)^T(\alpha\log(\pi_\theta(s_t)) - \min_{j=1,2} Q_{\omega_j}(s_t,a_t))\right] \tag{11}$$

2.3. *Overall framework of RL-MPL*

The overall framework of the proposed RL-MPL method is shown in Figure 1. The observation information containing the state of the ego vehicle (EV) and its surrounding traffic environment is transformed into one-dimensional vectors, which are used as inputs to the neural network. Based on the output of the neural network, the flexible lane changing option can be performed in the hierarchical action space and the optimal motion primitives in MPL will be finally selected. Specifically, each semantic lane-change option corresponds to i abstract motion primitives from which the agent can select. To simplify the action space, we set $i = 3$, with the motion primitives varying in their endpoint positions. In practice, the policy network implicitly encodes the selection of semantic lane-change actions, so that when the agent chooses a motion primitive, the corresponding lane-change action is inherently determined. After the generation of the motion primitive, trajectory optimization is performed under the framework of MPC to better fit the vehicle kinematics model. The motion primitive will finally be fed to the underlying controller of the EV to control the vehicle motion through steering angles and accelerations. The experiences during vehicle driving will be used to guide the training of the DRL agent.

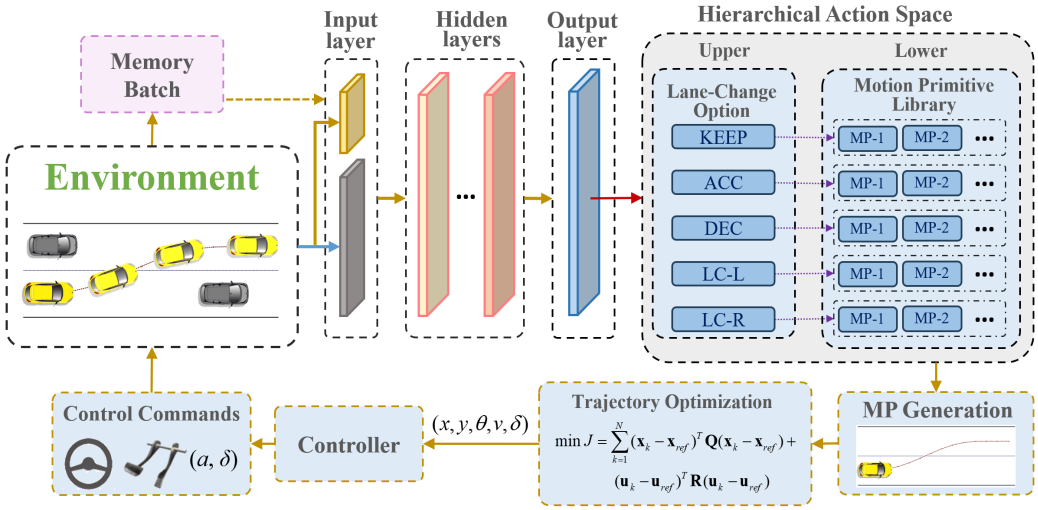

Figure 1. The whole framework. The proposed RL-MPL method incorporates a hierarchical action space, where the upper layer consists of a discrete set of lane-change options, and the lower layer contains a library of motion primitives.

3. RL agent formulation

This section will introduce the design of important concepts of DRL agent, including state space, action space, reward function and neural network structure.

3.1. State space definition

The agent needs to consider the information of the ego vehicle and the surrounding vehicles (SV) to perform the driving task on straight structured roads. To facilitate the use of information features as network inputs, we use a vectorized representation of the state space. The vector that can represent the EV information is defined as:

$$s^e = [x_e, y_e, v_x, v_y, a_x, a_y, ID_c, ID_t] \tag{12}$$

where x_e and y_e denote the lateral and longitudinal positions of the EV, respectively. v_x and v_y denote the lateral and longitudinal velocities. a_x and a_y denote the lateral and longitudinal accelerations. ID_c is the id of the lane in which the EV is currently located. ID_t is the id of the target lane of the EV.

To make the agent fully understand the traffic environment, we extract the information of six surrounding vehicles. The information of the i-th SV, after vectorization, can be expressed as:

$$s_i^s = [\Delta x_i, \Delta y_i, \Delta v_{x_i}, \Delta v_{y_i}, \Delta a_{x_i}, \Delta a_{y_i}, ID_{c_i}, ID_{t_i}] \tag{13}$$

where Δx_i and Δy_i denote the relative distances between SV and EV in the lateral and longitudinal directions. Δv_{x_i} and Δv_{y_i} denote the relative speeds between SV and EV in the lateral and longitudinal directions. Δa_{x_i} and Δa_{x_i} denote the relative accelerations between SV and EV in the lateral and longitudinal directions. ID_{c_i} and ID_{t_i} are the id of SV's current lane and the target lane.

In order for the state information to be used as an input to the neural network, the 7×8 two-dimensional information matrix needs to be expanded into a 1×56 one-dimensional vector:

$$s = [s^e, s_1^s, s_2^s, \cdots, s_6^s] \tag{14}$$

3.2. Hierarchical action space definition

The method proposed in this paper is based on a discrete hierarchical action space. The upper layer action space contains discrete options for lane changing. For each upper layer option, the lower layer action space correspondingly contains multiple alternative trajectory primitives. Therefore, the hierarchical action space can be defined as:

$$A_U = \{KEEP, LCL, LCR, ACC, DEC\} \tag{15}$$

$$A_L = \{M_1, \cdots, M_i\}_{option \in A_U} \tag{16}$$

where, A_U is the upper action space that contains five discrete options that determine what action the EV will take. Selecting *KEEP* means that the EV's target lane remains unchanged, while *LCL* or *LCR* means that the target lane will be changed to the left or right lane. And *ACC* and *DEC* mean to increase and decrease the target speed of the vehicle, respectively. A_L is the lower action space, where the number of discrete actions is i. M_i denotes the i-th motion primitive when option is fixed, and all the motion primitives together form the one MPL. Considering the computation time consuming, we take $i = 3$ in this paper, so the MPL contains 15 motion primitives. So, the DRL agent will select one motion primitive from the 15 motion primitives as an output. It should be noted that the motion primitive at this point is not a trajectory, but only a label. Section 4 describes how to build a motion primitive based on this label that can be used by the underlying EV controller.

3.3. Reward function definition

The objective of the DRL agent is to identify the globally optimal EV driving strategy, which is as efficient and comfortable as possible while ensuring safety. Therefore, a reasonable

reward function must be designed to guide the agent during training. The reward function R, designed in this paper, includes three parts:

$$R = R_s + R_e + R_c \tag{17}$$

where R_s is the safety reward, R_e is the efficiency reward and R_c is the comfort reward.

The safety aspect is the most important and it should be ensured that the EV does not crash while driving. Therefore, a large negative reward should be given when a collision occurs between the EV and the SV. R_s can be expressed by the following equation:

$$R_s = -k_s f_{coll} \tag{18}$$

where f_{coll} is the collision flag bit, which is set to 1 when collision occurs, otherwise to 0. k_s is the weighting factor for the safety reward, which is set to 10 in this paper.

In terms of driving efficiency, EV is encouraged to make lane changes to gain higher driving speeds and set the speed of the traffic stream to be slightly lower than the EV's target speed v_t. The closer the EV's speed is to v_t, the greater the positive reward it receives. R_e can be expressed by the following equation:

$$R_e = k_e \frac{|v - v_t|}{v_t} \tag{19}$$

where v is the current speed of the EV and v_t is the target speed of the EV. k_e is the weight coefficient of the efficiency reward, which is set to 5 in this paper.

In terms of comfort, the EV should be encouraged to maintain a uniform linear motion and minimise the number of lane changes and acceleration/deceleration operations. Meanwhile, it should be ensured to be as smooth as possible during EV lane changes. The equation for R_c is as follows:

$$R_c = -k_{c1} f_c - k_{c2} \frac{|\delta_{EV}|}{|\delta_{\max}|} \tag{20}$$

where f_c is the collision flag bit, f_c is 0 when the motion primitive corresponding to keep is selected and 1 in all other cases. δ_{EV} is the EV's real-time steering angle of the front wheels, and δ_{max} is the maximum steering angle. k_{c1} and k_{c2} are the weight factors of the comfort reward, which are both set to 1 in this paper.

3.4. Neural network design

The neural network architecture of Soft Actor Critic algorithm based on discrete action space is shown in Figure 2. In this paper, the networks of Actor and Critic are similar.

For actor, a policy network $\pi_\theta(s_t)$ is needed to be constructed to output the probability distribution of all discrete actions. The policy network contains three parts: input layer, hidden layers, and output layer, totaling four layers. The input layer has 56 units to obtain state information. Layer-2 has 256 units, while layer-3 has 128 units, which together form the hidden layers part. Considering the size of the MPL, the output layer has 15 units, each corresponding to the probability of selecting a motion primitive. The learning rate of the actor is set to 10^{-4}.

For critic, two value networks $Q_{\omega_1}(s_t)$ and $Q_{\omega_2}(s_t)$ with exactly the same architecture must be constructed to make the actor training more stable. The input layer has 56 units to receive state information, and both layers-2 and layer-3 have 128 units. The output layer has 15 units to output the values of all motion primitives. The learning rate of both value nets is set to 10^{-3}. In addition, two target value networks, $Q_{\omega_1^-}(s_t)$ and $Q_{\omega_2^-}(s_t)$, are needed to improve the training stability, which are set up in the same way as the value networks. $Q_{\omega_1^-}(s_t)$ and $Q_{\omega_2^-}(s_t)$ take a soft update to keep approaching $Q_{\omega_1}(s_t)$ and $Q_{\omega_1}(s_t)$, and the parameter update method is as follows:

$$\omega^- \leftarrow \tau\omega + (1-\tau)\omega^- \tag{21}$$

through multiple experiments by adjusting τ, we found that when $\tau = 0.005$, it achieves a good balance between the stability of the learning process and convergence efficiency.

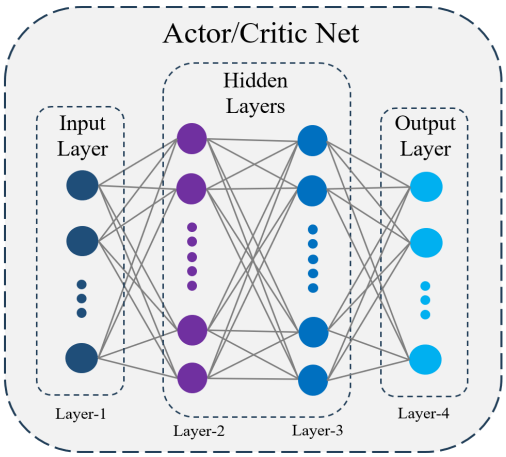

Figure 2. Neural network architecture. Both the actor and critic networks of the SAC-Discrete we used have four layers: the input layer, two hidden layers, and the output layer.

4. Motion primitive development and combination with RL

Each motion primitive is a reference trajectory of the EV for a future time period, and detailed information about the motion primitives is presented in this section. This section includes a polynomial curve based path point generation and a trajectory optimization method based on the vehicle kinematic model.

4.1. Path generation

When planning path points for motion primitives, the continuity of the path curve and the computational time consuming need to be considered [24]. The most common path generation method is based on polynomial functions. In the Cartesian coordinate system, the fifth degree polynomial function curve is given by:

$$y = \gamma_0 + \gamma_1 x + \gamma_2 x^2 + \gamma_3 x^3 + \gamma_4 x^4 + \gamma_5 x^5 \tag{22}$$

the points on the curve of this function can be used as reference path points when the vehicle performs a lane change action. The coefficients of the fifth degree polynomial γ_i can be estimated by solving linear equations when the position information of the start and end points is known. The EV's position information that needs to be obtained includes the x-position, the y-position, the yaw angle θ, and the curvature κ. We can denote the EV's position information at the initial moment T_0, with $(x_0, y_0, \theta_{e_0}, \kappa_{e_0})$, and then denote the EV's position information at the termination moment T_f, with $(x_f, y_f, \theta_{e_f}, \kappa_{e_f})$. As shown in Figure 3 is a schematic diagram of the motion primitive during lane change.

Consequently, when solving the linear equation, it is necessary to extract information from on-board sensors at the initial point of the path and to compute information at the termination point based on the decision-making result of the DRL agent. It is assumed that the EV will be traveling along the target lane in a straight line at the termination moment. In this case, θ_{e_f} and κ_{e_f} can be regarded as 0. The value of y_f is related to the selection of the DRL agent in the upper action space, which satisfies the following correspondence:

$$y_f = \{-L, 0, L\} \Leftarrow option = \{LCR, KEEP, LCL\} \tag{23}$$

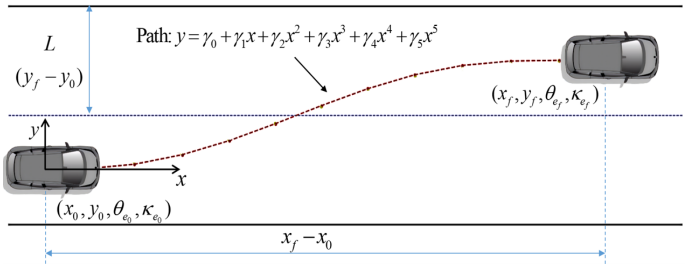

Figure 3. Motion Primitive during lane-change. The coefficients of the fifth-degree polynomial can be calculated based on the starting and ending states of the EV, and then the reference path points on the motion primitives are determined.

where L is the lane width. The value of x_f is not only related to the agent's decision, but also to the EV's velocity and termination moment T_f. The motion primitives in this paper are the possible reference trajectories of the EV in the next 5 s, so $T_{f_{max}} = 5$s. When the option based on the upper workspace is fixed, the terminal lateral position x_{f_i} is selected for the i-th motion primitive M_i in the lower action space:

$$x_{f_i} = \min\left(\sqrt{4R_0L - L^2}, \frac{v_x^2}{2a_{\max}^-}\right) + \frac{i}{i_{\max}} v_x T_{f_{\max}} \tag{24}$$

where R_0 is the minimum turning radius of the EV, $a_{\max}^-$ is the maximum braking acceleration of the EV, and $i_{\max} = 3$. Different values of x_f result in different shapes of the constructed motion primitives.

4.2. *Trajectory optimization*

Under the ground coordinate system, the MPC problem can be constructed based on the two-degree-of-freedom vehicle kinematics model, which in turn enables the optimization of the trajectory. Three states of the EV are selected to describe the system, $\mathbf{x} = [x_e, y_e, \theta_e]^T$. Then the velocity and front wheel steering angle of the EV are used as the control inputs, $\mathbf{u} = [v_e, \delta_e]^T$. Therefore, the description for the motion of the EV using the ordinary differential equation can be written as:

$$\begin{cases} \dot{x}_e = v_e \cos\theta_e \\ \dot{y}_e = v_e \sin\theta_e \\ \dot{\theta}_e = v_e \tan\delta_e / l_e \end{cases} \tag{25}$$

where l_e is the vehicle wheelbase of EV. The above model is a constant nonlinear system, which can also be denoted as:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \tag{26}$$

using Eulerian forward discretization, a discrete time representation of the above equation can be obtained:

$$\mathbf{x}(k+1) = \mathbf{x}(k) + f(\mathbf{x}(k), \mathbf{u}(k))\Delta t = F(\mathbf{x}(k), \mathbf{u}(k)) \tag{27}$$

where Δt is the discrete time interval. For convenience, $\mathbf{x}(k)$ and $\mathbf{u}(k)$ can be abbreviated as $\mathbf{x}_k$ and $\mathbf{u}_k$.

The objective of optimizing the motion primitives during lane-changing is to ensure a more consistent trajectory with the vehicle's kinematics model and greater smoothness. If both the number of control steps and the predicted steps are designated as N, an objective function can be formulated in the following form:

$$\min J = \sum_{k=1}^{N} (\mathbf{x}_k - \mathbf{x}_{ref})^T \mathbf{Q} (\mathbf{x}_k - \mathbf{x}_{ref}) + (\mathbf{u}_k - \mathbf{u}_{ref})^T \mathbf{R} (\mathbf{u}_k - \mathbf{u}_{ref}) \tag{28}$$

where $\mathbf{Q}$ and $\mathbf{R}$ are weight matrices. $\mathbf{x}_{ref}$ is the reference path point information calculated in section 4.1, $\mathbf{x}_{ref} = [x_{ref}, y_{ref}, \theta_{ref}]$. $\mathbf{u}_{ref}$ is the reference input information, $\mathbf{u}_{ref} = [v_{ref}, \delta_{ref}]$. v_{ref} is the target speed of the EV, and is related to the agent's option in the upper action space. To enhance the stability of EV during lane changing, the value of δ_{ref} is set to 0. For system 27, in conjunction with the driving scenario, the constraints of the control inputs need to be considered:

$$\begin{cases} 0 \leq v(k) \leq v_{\max} \\ -\delta_{\max} \leq \delta(k) \leq \delta_{\max} \end{cases} \tag{29}$$

Solving the above optimization problem results in a sequence of control inputs $U = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N]^T$ in the predicted time horizon, and the optimized state sequence $X = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^T$ can be obtained by substituting U into Equation (25). Using the results of the above solution, a suitable motion primitive M_i can be generated, which contains five kinds of information for each trajectory point: x-coordinate, y-coordinate, target yaw angle, target velocity, and target steering angle of the front wheels. Consequently, M_i can be expressed as follows:

$$M_i = [(x_1, y_1, \theta_1, v_1, \delta_1), \ldots, (x_N, y_N, \theta_N, v_N, \delta_N)]_i \tag{30}$$

4.3. Combination of MPL and RL

Based on the motion primitive, the DRL agent is able to participate in motion planning for the EV. At moment t, the policy network $\pi_\theta(s_t)$ can select the motion primitive $M_{i,t}$ in the lower action space based on the current observation information s_t. After optimizing the trajectory of the motion primitive, the underlying controller can be used to output driving commands. The EV can obtain a reward R_t and the environmental state feedback s_{t+1} at the next moment after driving based on the motion primitive. Therefore, the loss function of the policy network can be written according to Equation (11) as:

$$l_\pi(\theta) = \mathbb{E}_{(s_t, M_{i,t}) \sim D} \left[\pi_\theta(s_t)^T (\alpha \log(\pi_\theta(s_t)) - \min_{j=1,2} Q_{\omega_j}(s_t, M_{i,t})) \right] \tag{31}$$

For the two Q networks used in the training process, their loss function when considering $M_{i,t}$ according to Equation (6) can be written as:

$$l_Q(\omega) = \mathbb{E}\left[\frac{1}{2} (Q_\omega(s_t, M_{i,t}) - (R_t + \gamma(\min_{j=1,2} Q_{\omega_j^-}(s_{t+1}, \pi_\theta(s_{t+1})) - \alpha \log \pi_\theta(s_{t+1}))))^2 \right] \tag{32}$$

However, the form of the loss function for the temperature coefficient alpha remains unchanged. The detailed training process about the RL-MPL method is shown in Algorithm 1.

5. Experiment and results

In this section, the setup of the traffic environment model under straight structured roads is given. Then the training and testing results are analyzed to verify the superiority of the proposed method in this paper.

5.1. Experiment setup

We constructed a straight structured road simulation scenario with three lanes based on the highway-env platform, as shown in Figure 4. The SVs in the traffic flow are also randomly generated in each lane when the scenario is initialized, and the target speeds of all SVs are randomly generated in the range of [8 m/s,16 m/s]. In addition, to improve the ability of the traffic environment to interact with EV, the longitudinal and lateral driving models of the SVs are respectively IDM and MOBIL, and there are random lane-changing behaviors in the SVs.

Algorithm 1 Training process of proposed RL-MPL

Require: Total training steps N_T, learning rate of Actor l_r^A, learning rate of Critic l_r^A, temperature coefficient α, soft update coefficient τ.

Initialize: experience replay pool D, networks $\{Q, \pi\}$ with random parameters $\{\omega, \theta\}$, target networks with random parameters ω^-.
for $t = 0$ to N_T **do**
    Get state s_t from environment.
    Select $M_{i,t} = \pi_\theta(s_t)$ from MPL.
    Create motion primitive to map to $M_{i,t}$.
    Optimize trajectory for $M_{i,t}$.
    Use $M_{i,t}$ as reference to control EV.
    Get s_{t+1} and R_t from environment.
    Store transition $\{s_t, M_{i,t}, R_t, s_{t+1})\}$ into D.
    Sample randomly from D to compute $l_\pi(\theta), l_Q(\omega), l(\alpha)$.
    Update θ, ω, α with $\nabla_\theta l_\pi(\theta)$, $\nabla_\omega l_Q(\omega)$ and $\nabla_\alpha l(\alpha)$.
    $\omega^- \leftarrow \tau\omega + (1-\tau)\omega^-$
    $s_t \leftarrow s_{t+1}$
    if s_t is terminal **then**
        Reset environment.
    end if
end for
return

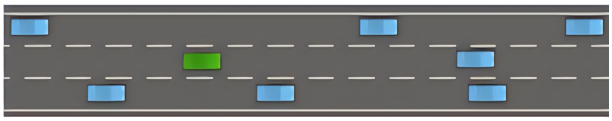

Figure 4. Straight structured road simulation environment in highway-env. The green vehicle represents the EV, and the blue vehicles represent the SVs.

The EV will be randomly generated in one of the lanes when the scenario is initialized, and its initial speed is randomly generated in the range of [8 m/s,16 m/s,]. The maximum speed limit of the EV is 20 m/s, and the target speed is 18 m/s. When a motion primitive is selected, the controller inside the EV will use the motion primitive as a reference to give the steering angle and acceleration commands of the vehicle. In this paper, we choose the stanley algorithm as the control algorithm inside the EV.

The update time step of the entire simulation scenario is 0.1s, and the maximum simulation time of a single episode is 200 s. When the EV collides with the SV, the simulation scenario ends and is reinitialized.

5.2. Comparison methods

In order to verify the superiority of the SAC-Discrete algorithm after combining MPL, four methods are selected as baselines in this paper:

(1) Model-based Method (IDM [25]+MOBIL [26]): In this approach architecture, the longitudinal and transverse control of EV is based on IDM and MOBIL respectively.
(2) Direct-RL Method (DQN [27], SAC-Discrete [23]): In this architecture, the output of DQN or SAC-Discrete is a discrete high-level semantic action option, not a motion primitive. Based on the agent's options, the EV's underlying PID controller will directly control the vehicle's motion.
(3) DQN+MPL: In this architecture, the DQN algorithm will be combined with MPL to enable the agent to output motion primitives to the underlying EV controller.

5.3. *Performance comparison*

As shown in Figure 5 is the average reward change in each episode for all methods during the training process. Of these, using IDM and MOBIL for EV control the average reward for each episode fluctuates within a stable range and is significantly worse than the learning-based approach. Based on the reward change curves, it can be found that all learning-based algorithms are able to converge to a stable driving strategy after 1000 episodes. Both for SAC-Discrete and DQN, the combination with MPL enables faster convergence and higher rewards. In addition, SAC-Discrete can be found to be more suitable than DQN for vehicle driving decisions in discrete action spaces.

To further verify the superiority of RL-MPL, we tested each of the trained models on 300 episodes, and the results are shown in Table 1. It is easy to find that the SAC-Discrete algorithm combining MPL still has the highest reward during the test. On straight structured roads, our approach enables the EV to travel at higher average speeds and more lane change times, while being able to maintain the lowest number of collisions. Analyzing the steering angle variance shows that the EV is able to have a smoother steering angle during lane changes due to the application of motion primitives.

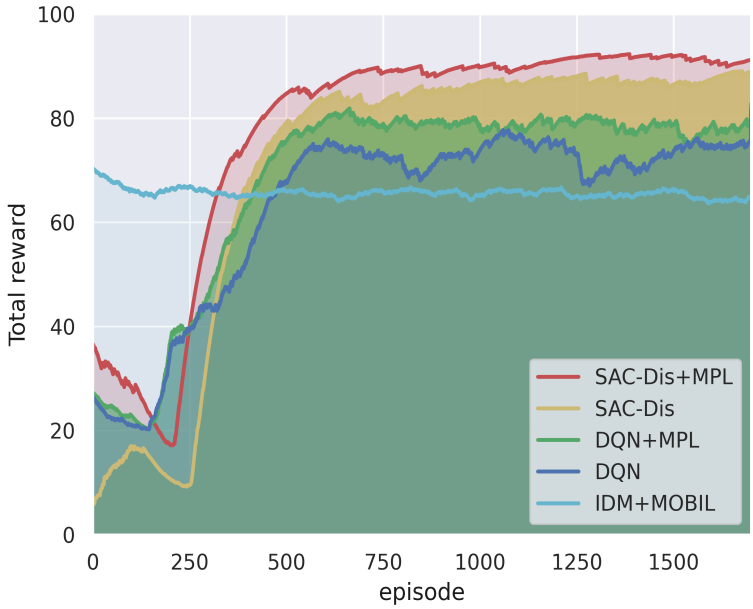

Figure 5. Training results. The figure shows the change in total reward for all methods during the training process over 1700 episodes.

Table 1. Comparison of test result.

Method	Average reward	Average speed	Episode LC times	Collision rate	Steering variance
IDM+MOBIL	0.651	8.3	5.22	0.17%	0.023
DQN	0.797	11.5	7.09	0.35%	0.027
DQN+MPL	0.834	12.3	7.65	0.21%	0.009
SAC-Dis	0.909	14.1	8.31	0.26%	0.025
SAC-Dis+MPL	0.942	14.8	8.99	0.11%	0.008

In summary, both training and testing results can prove that our proposed RL-MPL method is more outstanding. RL-MPL can make the EV obtain higher efficiency and flexibility, and can ensure the safety of the vehicle as well. Moreover, the combination of motion primitives and RL also improves the stability of the vehicle and the smoothness when changing lanes.

6. Conclusion

In this paper, a reinforcement learning method based on motion primitives library is proposed to select the optimal motion primitives from the hierarchical motion space. And MPC are used to optimize the trajectory and improve the fit of the motion primitives to the vehicle kinematics model. The SAC-Discrete algorithm, which has good performance in discrete action decision making, is used to combine motion primitives library. The experimental results show that the proposed RL-MPL method enables vehicle to achieve more flexible and stable motion planning under straight structured roads, improving driving efficiency, safety and smoothness. The future work will focus on:(1) enhancing the flexibility of DRL agents in motion planning by extending motion primitives to continuous action spaces; (2) improving the algorithm's generalization ability to handle more complex scenarios in autonomous driving; and (3) incorporating additional RL comparison algorithms in the experiments to provide a more comprehensive analysis.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China under Grant 52372394, in part by the Science and Technology Commission of Shanghai under Grant 21DZ1203802 and in part by the Fundamental Research Funds for the Central Universities.

Conflicts of interests

The authors declare no competing financial interests.

Authors' contribution

Conceptualization, Guizhe Jin and Zhuoren Li; methodology, Guizhe Jin; software, Guizhe Jin; validation, Guizhe Jin, Zhuoren Li, and Bo Leng; formal analysis, Guizhe Jin; investigation, Guizhe Jin; resources, Guizhe Jin; data curation, Guizhe Jin; writing—original draft preparation, Guizhe Jin; writing—review and editing, Zhuoren Li, Bo Leng, and Minhua Shao; visualization, Guizhe Jin; supervision, Minhua Shao; project administration, Minhua Shao; funding acquisition, Zhuoren Li. All authors have read and agreed to the published version of the manuscript.

References

[1] Li Z, Hu J, Leng B, Xiong L, Fu Z. An integrated of decision making and motion planning framework for enhanced Oscillation-Free capability. *IEEE Trans. Intell. Transp. Syst.* 2024, 25(6):5718–5732.

[2] Kopelias P, Demiridi E, Vogiatzis K, Skabardonis A, Zafiropoulou V. Connected & autonomous vehicles–Environmental impacts—A review. *Sci. Total Environ.* 2020, 712:135237.

[3] Guanetti J, Kim Y, Borrelli F. Control of connected and automated vehicles: State of the art and future challenges. *Annu. Rev. Control* 2018, 45:18–40.

[4] Gu Z, Gao L, Ma H, Li SE, Zheng S, *et al.* Safe-State enhancement method for autonomous driving via direct hierarchical reinforcement learning. *IEEE Trans. Intell. Transp. Syst.* 2023, 24(9):9966–9983.

[5] Liu Q, Li X, Yuan S, Li Z. Decision-Making technology for autonomous vehicles: Learning-Based methods, applications and future outlook. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, Indianapolis, USA, 19–22 September, 2021, pp. 30–37.

[6] Kiran BR, Sobh I, Talpaert V, Mannion P, Sallab AAA, *et al.* Deep reinforcement

learning for autonomous driving: a survey. *IEEE Trans. Intell. Transp. Syst.* 2022, 23(6):4909–4926.

[7] Chen L, Li Y, Huang C, Li B, Xing Y, *et al.* Milestones in autonomous driving and intelligent vehicles: survey of surveys. *IEEE Trans. Intell. Veh.* 2023, 8(2):1046–1056.

[8] Xing J, Wei D, Zhou S, Wang T, Huang Y, *et al.* A comprehensive study on Self-Learning methods and implications to autonomous driving. *IEEE Trans. Neural Netw. Learn. Syst.* 2024, pp. 1–20.

[9] Zhu Z, Zhao H. A survey of deep RL and IL for autonomous driving policy learning. *IEEE Trans. Intell. Transp. Syst.* 2021, 23(9):14043–14065.

[10] Li Z, Xiong L, Leng B, Xu P, Fu Z. Safe reinforcement learning of lane change decision making with Risk-Fused constraint. In *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*, Bilbao, Spain, 24–28 September, 2023, pp. 1313–1319.

[11] Wang P, Chan CY, de La Fortelle A. A reinforcement learning based approach for automated lane change maneuvers. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, Changshu, China, 26–30 June, 2018, pp. 1379–1384.

[12] Teng S, Hu X, Deng P, Li B, Li Y, *et al.* Motion planning for autonomous driving: the state of the art and future perspectives. *IEEE Trans. Intell. Veh.* 2023, 8(6):3692–3711.

[13] Jaladi SR, Chen Z, Malayanur NR, Macherla RM, Li B. End-To-End training and testing gamification framework to learn human highway driving. In *IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, Macau, China, 8–12 October, 2022, pp. 4296–4301.

[14] Kendall A, Hawke J, Janz D, Mazur P, Reda D, *et al.* Learning to drive in a day. In *2019 International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, 20-24 May, 2019, pp. 8248–8254.

[15] Hoel CJ, Wolff K, Laine L. Automated speed and lane change decision making using deep reinforcement learning. In *International Conference on Intelligent Transportation Systems (ITSC)*, Rhodes, Greece, 4–7 November, 2018, pp. 2148–2155.

[16] Li S, Wei C, Wang Y. Combining decision making and trajectory planning for lane changing using deep reinforcement learning. *IEEE Trans. Intell. Transp. Syst.* 2022, 23(9):16110–16136.

[17] Zhang Y, Gao B, Guo L, Guo H, Chen H. Adaptive Decision-Making for automated vehicles under roundabout scenarios using optimization embedded reinforcement learning. *IEEE Trans. Neural Networks Learn. Syst.* 2021, 32(12):5526–5538.

[18] Naveed KB, Qiao Z, Dolan JM. Trajectory planning for autonomous vehicles using hierarchical reinforcement learning. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, Indianapolis, United States, 19–22 September, 2021 pp. 601–606.

[19] Lu X, Fan FX, Wang T. Action and trajectory planning for urban autonomous driving with hierarchical reinforcement learning. *arXiv* 2023, arXiv:2306.15968.

[20] Wang XF, Jiang J, Chen WH. A hierarchical control framework for autonomous decision-making systems: integrating HMDP and MPC. *arXiv* 2024, arXiv:2401.06833.

[21] Bellman R. A markovian decision process. *Indiana Univ. Math. J.* 1957, 6:679–684.

[22] Lillicrap TP, Hunt JJ, Pritzel A, Heess NMO, Erez T, *et al.* Continuous control with deep reinforcement learning. *CoRR* 2015, abs/1509.02971.

[23] Christodoulou P. Soft actor-critic for discrete action settings. *arXiv* 2019 arXiv:1910.07207.

[24] Shui Y. Strategic trajectory planning of highway lane change maneuver with longitudinal speed control. Bachelor's Thesis, The Ohio State University, 2015 .

[25] Treiber M, Hennecke A, Helbing D. Congested traffic states in empirical observations and microscopic simulations. *Phys. Rev. E* 2000, 62(2):1805.

[26] Kesting A, Treiber M, Helbing D. General lane-changing model MOBIL for car-following models. *Transp. Res. Rec.* 2007, 1999(1):86–94.

[27] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, *et al.* Human-level control through deep reinforcement learning. *Nature* 2015, 518(7540):529–533.