sensors

Article

Safe Decision Controller for Autonomous Driving Based on Deep Reinforcement Learning in Nondeterministic Environment

Hongyi Chen [1], Yu Zhang [1,*], Uzair Aslam Bhatti [2] and Mengxing Huang [2]

[1] School of Computer Science and Technology, Hainan University, Haikou 570228, China
[2] School of Information and Communication Engineering, Hainan University, Haikou 570228, China
* Correspondence: yuzhang2015@hainanu.edu.cn

Abstract: Autonomous driving systems are crucial complicated cyber–physical systems that combine physical environment awareness with cognitive computing. Deep reinforcement learning is currently commonly used in the decision-making of such systems. However, black-box-based deep reinforcement learning systems do not guarantee system safety and the interpretability of the reward-function settings in the face of complex environments and the influence of uncontrolled uncertainties. Therefore, a formal security reinforcement learning method is proposed. First, we propose an environmental modeling approach based on the influence of nondeterministic environmental factors, which enables the precise quantification of environmental issues. Second, we use the environment model to formalize the reward machine's structure, which is used to guide the reward-function setting in reinforcement learning. Third, we generate a control barrier function to ensure a safer state behavior policy for reinforcement learning. Finally, we verify the method's effectiveness in intelligent driving using overtaking and lane-changing scenarios.

Keywords: autonomous driving; formal specification; deep reinforcement learning; safe decision controller generation; nondeterministic environment

Citation: Chen , H.; Zhang, Y.; Bhatti, U.A.; Huang, M. Safe Decision Controller for Autonomous Driving Based on Deep Reinforcement Learning in Nondeterministic Environment. *Sensors* **2023**, *23*, 1198. https://doi.org/10.3390/s23031198

Academic Editors: Sanislav Teodora, Ovidiu P. Stan and Dan Gota

Received: 29 November 2022
Revised: 15 January 2023
Accepted: 17 January 2023
Published: 20 January 2023

1. Introduction

Autonomous driving systems are complicated cyber–physical systems (CPS) that integrate physical environment awareness with intelligent computing, and their functioning includes both computational and physical processes that interact with and influence one another. A typical autonomous driving system includes the physical environment, plant, sensor, decision controller, and actuator. The decision controller is based on sensing physical environment information and integrating the physical environment and self-vehicle information to generate safe and reasonable driving behavior and control the vehicle's movement.

Currently, a data-driven approach is common when obtaining autonomous driving decisions. Among data-driven approaches, training through deep reinforcement learning (DRL) is popular. Deep reinforcement learning seeks the best strategies through extensive learning training by interacting with the environment; however, for complex driverless traffic scenarios, the decision-making system cannot understand the complex traffic environment around it. Safe decision-making behavior cannot be provided for unrecognized states; therefore, the safety of deep reinforcement learning has been widely discussed.

However, most DRL tasks not only require a considerable amount of training time and iterations, but they also do not guarantee the safety of the actions taken during the learning process, which can be a serious problem in some safety-critical areas (e.g., autonomous driving, medical, etc.).

There are two main reasons why reinforcement learning is currently primarily applied in simulation environments rather than real-world scenarios. On the one hand, there is a lack of interpretability for deploying reward functions for deep reinforcement learning because the state and action space inputs for reinforcement learning come from a network of black boxes. The reward function allows the combination of prior knowledge [1]; however, reward functions based on prior human knowledge are typically subjective and sparse and thus cannot accurately represent the task goal [2]. Subjectivity, in other words, leads to reward functions that may focus on specific explicit criteria while ignoring the relationship between specific states and task goals. This lack of interpretability has become one of the main obstacles in the field. On the other hand, the real-world environment is characterized by many uncertain environmental factors. Because of the complexity of the physical environment in which autonomous driving exists, it is possible to ignore some nondeterministic environmental factors, which means that performing the same action in the same state may result in a different next state or reward value, implying that the intelligent driving system is unpredictable. This means that the autonomous driving system cannot complete the task based on the original planning decision, and it must adjust the original decision based on the cognitive outcomes of the nondeterministic environment. In the face of these uncontrolled changes in conditions, reinforcement learning cannot provide strict safety constraints for safety-critical unmanned systems.

To address the above issues, we combine formal methods and DRL to propose a DRL-based approach for generating decision controllers for autonomous driving CPS in a nondeterministic environment, where the learning process is guided by expected behavioral constraints.

First, we modeled the environment for describing the physical environment of the autonomous driving system and quantitatively evaluated its impact on the system. Second, we designed the structure of a reward machine through a nondeterministic environment model. The reward machine, based on the policy model, guides the design of the reward function, which allows the system state and goal to be combined and provides an interpretable method for the reward function. Simultaneously, we used safety properties to generate control barrier functions to monitor the safety of the decision's output to the actuators and to improve the satisfaction of safety specifications to compensate for the DRL. Finally, we evaluated the decision controller's performance in the overtaking lane-change scenario in terms of safety, efficiency, and satisfaction with the constraints. The following are our main research contributions:

- We proposed a method for modeling the environment of autonomous driving systems in nondeterministic environments. We quantitatively described the surrounding environmental risk by means of a well-defined nondeterministic physical environment model.
- Our proposed method combined safety verification and DRL. We defined a form of reward-machine ground description based on the environmental risk model to guide the setting of the reward function in DRL, making the reward function for DRL interpretable. We also designed a monitor using safety properties to ensure that decisions satisfying the safety properties were output to the controller.

The rest of our paper is structured as follows: Section 2 provides a synopsis of related work. Section 3 introduces the fundamental theory of metric temporal logic and reinforcement learning. Section 4 defines the general structural framework. Section 5 describes methods for modeling systems in nondeterministic environments. Section 6 proposes a decision-making generation method for safety-constrained guidance. Section 7 includes experiments and evaluations of the proposed method's effectiveness. Finally, we draw our conclusions in Section 8.

2. Related Work

The safety control of a CPS system is a new research field, in which the use of safety reinforcement learning (RL) is a considerable topic. There are many methods of safety RL, which were summarized by Garcia and Fernández [3]. They divided these methods into two categories: modifying the optimality criteria and changing the learning process of agents.

Many approaches to safety RL use a restricted optimality criterion in RL. The agent searches the entire policy space for a priori safe control behaviors [4]. Xu and Mannor modified the optimality criterion to reflect the safety problem [5], providing the agent with some preliminary knowledge to guide policy exploration away from unsafe states.

It is critical to distinguish between optimization within known safety policies and an exploration of existing control options. Katz et al. [6] examined learning-process policies, which are appropriate when the learning phase is not safety critical but not when the system must operate safely while learning. Value consistency and safety policy validation are two considerable evaluation metrics in safety RL. The goal of value consistency is to ensure that the learning system's goals are consistent with human intentions [7]. This is because RL agents can profit from the wrong goals.

The goal of policy validation is to understand how and why learned policies make certain decisions. This is critical for the secure deployment of RL agents. The task's goal can be expressed in a reward-based or -free manner through value consistency. Reward shaping is a popular method for developing intense reward functions.

On the one hand, forming a dense reward function that is well-aligned with the true task goal can be difficult for complex tasks. Learning from demonstrations, on the other hand, allows agents to learn without an explicit reward function; to this end, J. Lee et al. [8] provided a recent overview of recent advances in value alignment. Recent value alignment efforts include iterative scaling [9,10], in which agents and personnel collaborate to learn the desired strategy (possibly during iteration). However, the approaches described above do not allow strategies to avoid risk simply by using maximizing returns as an optimization criterion.

For RL in an uncertain environment, G. Mason et al. [11] used a probabilistic model to check the verification strategy according to the constraints specified in probabilistic temporal logic; then, they modeled the environment as an abstract Markov decision process to validate the policy. However, few studies consider the worst-case scenario under uncertainty and minimax criteria, outputting a control strategy after optimizing the target to maximize the reward.

At the same time as maximizing the income expectation, Richa et al. [12] proposed that other types of utility, including risk, should be within certain given boundaries, taking such constraint-system states as fence functions and strategy space constraints or boundaries. Further research [13] examined the risk-related prior information for agents, which allows to avoid risk states during training and ensures a more efficient learning.

Another study [14] combined interpretable rule-based policies and black-box reinforcement learning to achieve system safety and robustness. Krasowski [15] used an ensemble-based approach to enhance the safety of reinforcement learning by extending the reinforcement-learning safety layer to limit the action space. Wachi et al. [16] proposed a reinforcement learning method with a constrained Markov decision process, which continuously learned security constraints by extending the security region. Bastani et al. [17] proposed a method for learning and verifying decision tree strategies. In the DRL field, strategy verification to a large extent belongs to the category of neural network verification. Strategy verification is usually used to explain the learned strategies after training. Some studies [18,19] combine temporal logic and automata to solve the non-Markov reward decision process, while other studies [20,21] use the robustness of signal sequential logic to stimulate the learning process. Wen et al. [22] combined the maximum likelihood IRL with a task constraint in the form of common safety linear sequential logic. Alshiekh et al. [23] used finite-state automata as a safety measure to supervise the agent-release operation. In our study, we used temporal logic and the corresponding automata to represent task goals and guide learning, respectively. In addition to verifying the decision, there are also interventions to ensure the safety of the training process. Hahn et al. [24] proposed a technique to compile w-regular attributes into limit-deterministic Büchi automata, which led to w-regular rewards that RL agents could learn. Toro et al. [25] introduced the reward machine, which closely reorganizes an automaton, and demonstrated its use on the RL benchmark. Araki et al. [26] developed a logic-based value iteration network, which integrated FSA into value iteration through demonstration.

Li [27] proposed a deep reinforcement learning algorithm that incorporated a risk-assessment function to find the optimal driving strategy and minimize the desired risk. However, the authors only assessed driving risk in terms of positional uncertainty and distance-based safety measures, which lacked any consideration of uncertain environmental factors, such as driving style. As such, we used a formal approach to define driving risk for nondeterministic environmental factors. Muzahid et al. [28] made several contributions to the scheme for adapting the reward function. They used a multiobjective reward system to balance achievement rewards from cooperative and competitive approaches, accident severity, and passenger comfort. However, they considered this from the perspective of multivehicle crashes and overlooked the effects of differences in speed and driving style.

In general, the lack of interpretability and ensuring security in complex environments have become considerable research topics for deep reinforcement learning research.

3. Basic Theory

3.1. Metric Temporal Logic

Linear temporal logic (LTL) is a kind of temporal logic. It models time as a series of states and describes the constraints that the system needs to meet in the past, present, and future through time operators. Linear temporal logic is used to represent the properties or constraints that the system is expected to meet. If the AP is an atomic proposition set, the LTL formula on the set AP can be recursively defined as follows [29].

$$\varphi ::= p|\neg\varphi|\varphi_1 \wedge \varphi_2|\varphi_1 \rightarrow \varphi_2|\varphi_1 \leftrightarrow \varphi_2|G\varphi|F\varphi|X\varphi|\varphi_1 U \varphi_2 \tag{1}$$

In the above definition, they are all LTL formulas, and the standard operators logical and ($\wedge$), logical or ($\vee$), not ($\neg$) and implication ($\rightarrow$) are propositional logic symbols. Operators F (future), G (globally), X (next) and U (until) are sequential operators. The explanation is as follows.

- $\varphi, \varphi_1, \varphi_2$ are logical propositions, and the return value is true or false.
- $G(\varphi)$:φ is always true on the time axis.
- $F(\varphi)$:φ is always true on the time axis and will be true at some time in the future.
- $X(\varphi)$:φ is true at the next point in time.
- $\varphi_1 U \varphi_2$: Before φ_2 becomes true at some point in the future, φ_1 is continuously true.

Metric temporal logic (MTL) [30] is an extension of LTL that uses discrete-time, interval-time arithmetic to specify the time limits that must be maintained for certain temporal properties and is thus well-suited to represent real-time monitoring requirements. It is a timing logic that operates in real time. Each temporal logic operator, such as $G[i,j], F[i,j], \varphi_1 U[i,j]\varphi_2$ and so on, has a subscript representing the duration of the operator. $G[1,2]\varphi$ indicates that φ must be true between the times 1 and 2. LTL formulas are analogous to MTL formulas without time intervals.

3.2. Reinforcement Learning (RL)

RL is a method for maximizing long-term rewards in Markov decision processes (MDPs) that has found widespread application in artificial intelligence [31]. In RL training, the agent without prior knowledge must obtain the information about the environment through continuous exploration (i.e., interaction with the environment) and obtain the optimal decision strategy through repeated experiments. The agent can learn the decision-making strategies for specific tasks independently, without the need for designers to describe the implementation of tasks in detail. In the learning process of the agent, after each agent makes a tentative decision, the environment will change its state correspondingly, and give the agent a reward value (or reward) according to the changed state. According to the continuous observation of the environment and the feedback obtained, the agent will optimize its strategy in the process of continuous exploration, continue to evolve, and finally get the optimal decision strategy.

Definition 1. *A MDP (Markov decision process) is defined as a tuple $M = (S, A, T, R)$, where*

- *S is the state space, $S \subseteq R^n$.*
- *A is the action space, $S \subseteq R^m$.*
- *T is a transfer function $S \times S \times A \to [0, 1]$. $T(s_{t+1}s_t, a_t)$ is the conditional probability that $s_t \in S, a_t \in A$ is in state s_{t+1}.*
- *R is a reward function $R : S \times A \times S \to R$, which is a reward obtained by performing operation a_t in state s_t and transitioning to state s_{t+1}.*

Time is defined as discrete $t = 0, 1, 2, \ldots$. s_t and a_t represent the state and action at time t, respectively. The best strategy $\pi = S \to A$ is a strategy for maximizing the expected return:

$$\pi = \arg\max_{\pi}\left(E\left[\sum_{t=0}^{T-1} r(s_t, a_t, s_{t+1})\right]\right). \tag{2}$$

T is the maximum number of time steps allowed per execution P and thus the maximum length of the trajectory. E is the expected value after policy p in the equation. The action-value function of the state is defined as

$$Q^p(s, a) = E\left[\sum_{t=0}^{T-1} r(s_t, a_t, s_{t+1})s_0 = s, a_0 = a\right] \tag{3}$$

This is the expected payoff for selecting as an action a strategy P in state s. Q^p is typically used to assess the quality of strategy P. Q^p and P are usually in the form of parametric function approximators for problems with continuous states and action spaces (e.g., automatic driving).

4. Framework

The overall proposed framework for the safe decision controller generation proposed in this paper is shown in Figure 1. Figure 1 shows a diagram of the framework generated by the controller which describes the forward flow of data. More specifically, Figure 1 is divided into two components. The left and right regions are processes in the training and physical domains, respectively, and the right region is a process in the physical domain. The controller obtained through the training domain is able to act on the entity.

In the training domain, in the first step, a simulation environment is obtained by accurately modeling the real environment of the autonomous driving system, from which nondeterministic environmental factors are obtained. In the second step, a nondeterministic environment model is constructed by analyzing the classifiers of risk factors and safety properties, and the environment model is defined based on a formal approach to the constraints. In the third step, the environment model is constructed as both a reward machine and a monitor, respectively. The reward machine is used to guide the reward function in RL. The monitor is used to set the monitoring barrier, which monitors the output actions. If the safety properties are not satisfied, the action that satisfies the safety properties are selected from the action set as a way to ensure the decision. The fourth step is the routine execution of reinforcement learning. Data are fed to the agent from the simulation environment, after which an action is obtained for feedback.

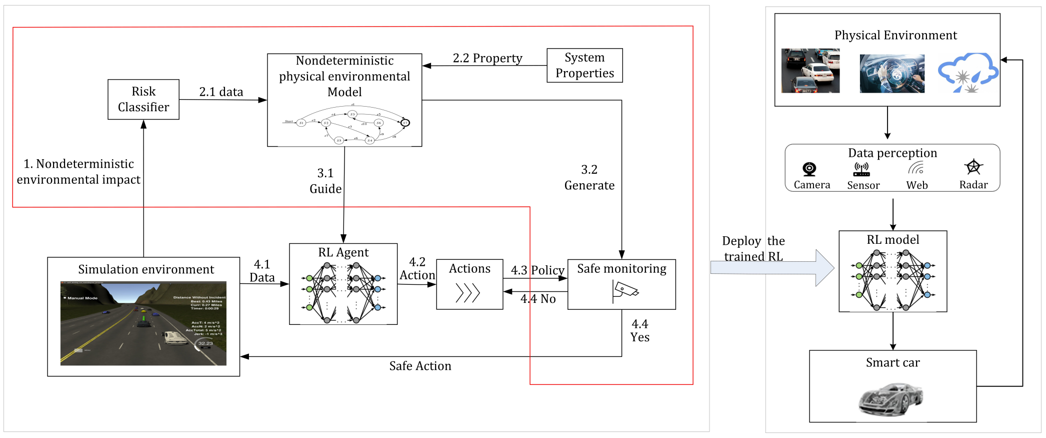


Figure 1. Framework for safe decision controller generation.

5. Modeling of the Nondeterministic Physical Environment

There are three types of nondeterministic effects on autonomous driving systems: internal uncertainty, external uncertainty, and sensor uncertainty [32]. Internal uncertainty refers to the differences among the system's actions, external uncertainty refers to the uncertainty of the system's surrounding factors, and the sensor uncertainty refers to the deviation of the information received by the sensor. Because the internal and sensor uncertainties are mainly affected by their own design, we mainly focused on the external nondeterminism; that is, the nondeterminism of environmental factors around the system.

Weather and location type are examples of nondeterministic surrounding environmental factors. Temperature, humidity, rainfall, and other factors all have an impact on the weather category. The location category refers to the unpredictability of the activity trajectory of the surrounding objects. The vehicle lane-changing decision-making process is often affected by the nondeterministic environment, in which the interaction with other vehicles is a factor. In the overtaking lane-change scenario, we may be concerned about the vehicle behind us colliding with us because we do not know whether it will accelerate or decelerate. The nondeterministic environment around the vehicle is defined as follows.

Definition 2. *The nondeterministic physical environment model ENV is represented as the tuple $ENV = (ER, PR)$, where*

- *ER is a formal description of the static structure of the nondeterministic physical environment model;*
- *PR is a dynamic description of the nondeterministic physical environment model.*

Definition 3. *The static structure of the nondeterministic physical environment model ER is a four-tuple $ER = (Objects, Parameters, Function, Type)$, where*

- *Objects represents the set of surrounding nondeterministic objects;*
- *Parameters represents the parameter set of nondeterministic factors;*
- *Function represents the probability formula for calculating the category under the nondeterministic environment;*
- *Type represents the set of types of nondeterministic environments.*

Definition 4. *The dynamic description of the nondeterministic physical environment model PR is represented as a six-tuple $PR = (Q, Q_0, acts, prob, E, L)$, where*

- *Q represents a finite state set;*
- *Q_0 represents the initial state;*
- *$acts$ is a finite set of actions;*
- *$prob$: $Q \times prob \rightarrow E$; $prob$ is the probability of possibility transfer, expressed in a nondeterministic environment;*
- *E: $E \subset Q \times acts \times L \times Q$ is the transfer function;*

- *L: the migration condition.*

The driving style of surrounding vehicles, as a typical factor of the nondeterministic environment, has great influence on automatic driving. Thus, we took the driving style of surrounding vehicles as an example to demonstrate the nondeterministic physical environment model. We assigned a trust level to the surrounding vehicles and attached a value. For example, in the interval [0, 1], if we really do not trust it, its value is 0; if we completely trust it, its value is 1. Typically, we will have an intermediate value depending on how much we know about the vehicle behind us. If we have elements that help the vehicle to perform safe driving (e.g., the amount of speed change, distance between cars, etc.), we can set a value closer to 1, while in the opposite case it is closer to 0. With this level of trust, we can consider the assertion's veracity.

Figure 2a shows a forced lane change for the rear car's aggressive style may result in a collision. It is easy to avoid a collision with a conservative style, as shown in Figure 2b. Driving styles were classified as aggressive, normal, or conservative. The aggressive type of rear car should avoid changing lanes, the ordinary type could be taken to change lanes or not, and the conservative type could safely change lanes.

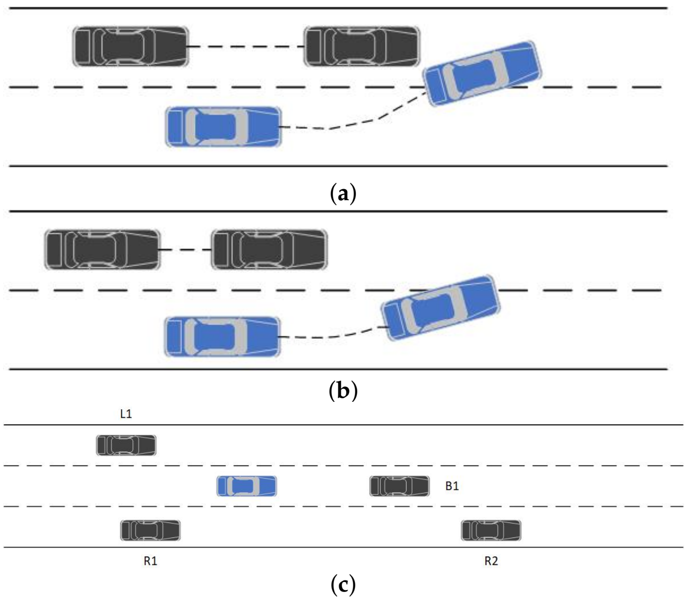


Figure 2. Driving environment. (**a**) Radical collision. (**b**) Conservative successful lane change. (**c**) Surrounding objects.

We used the environmental model ER to identify environmental factors and risk probabilities. $Objects = \{L_1, B_1, R_1, R_2\}$ represented the other vehicles in each of the three lanes. $Parameter = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$ represented the parameter for calculating the probability. $Function = \{Funp(x_i|1 < i < 7) \rightarrow (type, p)\}$ represented the formula for calculating the probability. $type = \{type_1, type_2, type_3\}$ represented three different driving styles. Each style of transition had a migration relation with probability p. We used the parameters in Table 1 to calculate the likelihood of each driving style being judged.

We calculated driving style probabilities using the following formula based on the mean values of the plain Bayesian classifier's driving style classification parameters [33].

$$Funp = \sum_{i=1}^{7} w_i \times B_{xi} \tag{4}$$

w_i denotes the weights obtained by training the classifier, where $\sum_{i=1}^{7} w_i = 1$. B_{x_i} denotes whether x_i satisfies the computed metric, with a value of 0 or 1. The driving style was predicted by inferring high-level information from low-level data, namely the experimental calculated value and the parameter index. Figure 3 is an example of a dynamic structural representation of a risk model, containing aggressive, conservative, and ordinary types, which were transformed between them by an identification function, and entering a state triggered a reset condition that reset the probabilities.

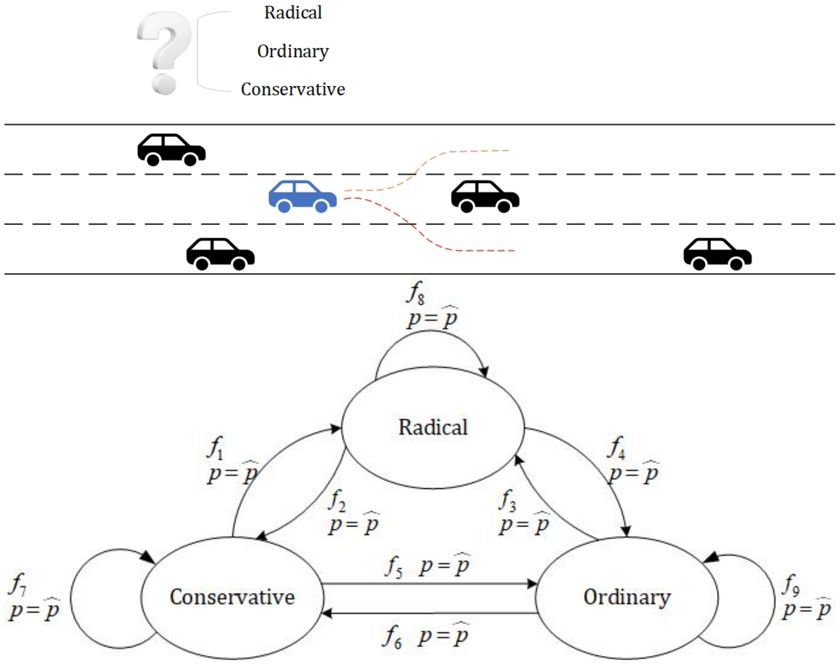


Figure 3. Obstacle avoidance and lane change decision.

Table 1. Various characteristics of the parameters.

	Parameters' Characteristics
x_1	Average velocity
x_2	Standard deviation of vehicle speed
x_3	Maximum lateral acceleration
x_4	Standard deviation of lateral acceleration
x_5	Following distance
x_6	Lane change time
x_7	Lane change clearance

Self-driving cars on the road need to avoid obstacles as a case of extreme safety. When the blue car on the road overtakes, it has two options: change lane from the left or right. However, the car's CPS system can only observe the distance between the vehicles and cannot determine the trajectory of the side vehicle. If the rear vehicle is aggressive, it is likely to change lane when the rear vehicle speeds up, resulting in a collision.

6. Decision Controller Generation Algorithm for Safety Protocol Guidance

Although RL has been shown to be applicable to complex environmental-interaction-like problems, it faces the problem of lack of interpretability in safety-sensitive domains, and it is difficult for the output policies to guarantee their safety reliability, which poses safety hazards. The reward function of RL is designed to be interpretable in this section. The structure of the reward machine is designed through migration changes between environmental risk model states to guide the setting of the reward function in RL. Additionally, the monitor is designed to guarantee that the output decisions are safe according to the safety properties.

6.1. System Specification Modeling

In the automatic driving environment, the safety property that needs to be monitored is often related to time. For example, before changing lanes, it is necessary to turn on the turn signal for 3 s before completing the lane-changing task. This kind of system-safety property with a constraint relationship is usually called a real-time property. The difference between real-time and conventional nature is often reflected in timeliness, which affects the effective time of the decision-making and determines whether the decision can be completed within the specified time.

Because real-time monitoring is time-constrained, it is different from a traditional monitoring process. To monitor the real-time property at run-time, a real-time sequential logic is first used to describe it. In this paper, MTL was used to describe the safety property.

R1: the autonomous vehicle maintains the maximum safe speed of 50 km/h.

$$\mathrm{G}(V_{\max} \leq 50)$$

R2: when the vehicleis in the leftmost lane, it is not allowed to change lanes to the left.

$$\mathrm{G}((loc = left) \rightarrow (\neg turnleft\_cmd))$$

R3: when deciding to change lanes to the left, 3 s after the left side still has a safe distance, the lane change can be completed in 1 s.

$$G(((CMD = turnleft) \wedge F[0,3]not\_left) \rightarrow F[3,4]turnleft)$$

R4: the time spent not in the center of the lane while driving shall not exceed 10 s.

$$\mathrm{G}\neg(G[0,9](loc \neq center))$$

6.2. Setting of Reward Function for Safety Constraint Guidance

The reward function is usually treated as a black box in RL, with no way to understand the settings inside. Because of this approach, the agent-trained decision-maker is unaware of the environmental transfer changes or the state–change relationships and can only obtain the final result. We must accurately perceive the returns from environmental data and allow the agent to learn from environmental interactions. We propose using safety properties and a nondeterministic environment model to guide the setting of the reward function. To express the reward function, we used a reward machine.

Definition 5. *The reward machine RM is a six-tuple $RM = (U, u_0, acts, \sigma, E, \tau_r)$, where*

- *U is a finite set of states;*
- *$u_0 \in U$ is the initial state;*
- *acts is a collection of actions;*
- *σ are migration conditions;*
- *E is the transfer set, $E \subseteq U \times acts \times \sigma \times U$;*
- *τ_r is a reward function; the current reward $\tau_{ri} = \tau_r(u_{i-1}, u_i)$.*

The reward machine decomposes the problem of safety constraints into a group of high-level states U and defines transitions of δ_u using if conditions. These conditions apply to the safety constraint properties φ, which can be tested to see if they satisfy truth values.

Definition 6. *The reward form RF is a five-tuple: $\tau_r =< r, \varphi, \varphi_r, risk, risk_r >$, where*

- *r represents the base reward set;*
- *φ represents the safety constraint property;*
- *φ_r represents positive feedback provided by the satisfaction of φ when it is true;*
- *risk represents the risk's probability of occurrence;*
- *$risk_r$ is a function of the negative feedback.*

First, we set basic rewards. The primary concerns are safety and efficiency, which can be obtained by understanding the nature of safety. The system designer wishes to drive the vehicle efficiently while remaining safe on the road in order to avoid collisions. The reward function provides a way to combine prior knowledge and change the decision according to the previous feedback response. One criterion for determining the reward function is to improve the alignment of values between intentions and system goals, as well as to reduce decision errors caused by malicious perturbations of the reward function. Therefore, setting the reward value according to the task result cannot accurately express the advantages and disadvantages of the actual situation, and it needs to be able to measure the different good and bad situations of the execution in different environments.

To improve efficiency, self-driving cars should try to drive as fast as possible while not exceeding the maximum speed limit. Therefore, the following rewards were defined according to the vehicle speed.

$$r_v = \lambda_1(v - v_{res} + \omega_1) \tag{5}$$

where v represents an average speed since the last decision period, v_{res} is the threshold speed limit, λ_1, ω_1 are the normalization coefficient, and $r_v \in [-1, 1]$ is the reward function of the vehicle speed.

According to the formal description of the safety problem of self-driving cars, in the process of learning the optimal policy, the agent needs to respond to the dynamic evaluation of policy changes in a dynamic environment. Different decisions are made in the current environment due to the different driving styles of other driving vehicles around, resulting in different changes in the distance between vehicles. Due to the different driving styles of other surrounding vehicles and the different decisions made in the current environment, the change of vehicle distance is also different. A big reason for collision accidents on the road is that the vehicle distance from the surrounding vehicles is far less than the safe distance. Therefore, the reward function of dynamic change concerning vehicle distance was formulated to measure whether it was good or bad. According to the critical safety time distance [34], it was defined as

$$d = \frac{(v_r + |a_r|t_\tau - v_f)^2}{2(|a_r| - |a_f|)} - \frac{|a_r|t_\tau^2}{2} \tag{6}$$

where v_r is the vehicle speed of the rear vehicle, v_f is the vehicle speed of the front vehicle, a_r is the acceleration of the rear vehicle, a_f is the acceleration of the front vehicle, and t_τ is the response lag time. The acceleration a and vehicle speed v were calculated by the instantaneous displacement, and the formula was as follows:

$$\begin{cases} v = \frac{d(t+\Delta t) - d(t)}{\Delta t} \\ a = \frac{v(t+\Delta t) - v(t)}{\Delta t} \end{cases} \tag{7}$$

The violation of the safety distance constraint causes RL to make the vehicle drive at a safe distance during the learning process. According to the relationship of the safety distance, we gave a reward function about the distance:

$$r_d = \lambda_2(d - d_{act} + \omega_2) \tag{8}$$

where d represents the critical safety time distance, d_{act} is the observed distance, λ_2 and ω_2 are the normalization coefficients, and the reward function of the vehicle speed is $r_d \in [-1, 1]$.

According to the automaton representation of safety property transformation, when facing the leftmost or rightmost side of the road, the vehicle is not allowed to change lanes left or right, so a negative feedback reward $r_{illegal}$ is set. When the decision is made successfully, a positive feedback reward $r_{succeed}$ is given. If a collision occurs, it is a great safety problem, and a strong negative feedback reward $r_{collision}$ is set.

At the same time, there is a possibility that the RL will cause the vehicle to make a lane change even if there is no vehicle in front of it. There is no safety accident following a legal lane change, but this violates one of the constraints (frequent lane changes increase the probability of collision and reduce the number of lane changes to reach the destination as soon as possible). Thus, in order to alter this phenomenon, we established a negative feedback reward r_{valid}.

$$r = \begin{cases} r_y + r_d & \text{Current Lane} \\ r_y + r_d + r_{success} & \text{Succeed Change} \\ r_y + r_d + r_{invalid} & \text{Invalid Change} \\ r_{illegal} & \text{Illegal Change} \\ r_{collision} & \text{Collision} \end{cases} \tag{9}$$

We designed the rewards directly from high-level specifications to make the reward synthesis problem more formal and easier to interpret and handle. Figure 4 shows a reward machine with reward specification $\tau_r = < r, \varphi, \varphi_r, risk, risk_r >$.

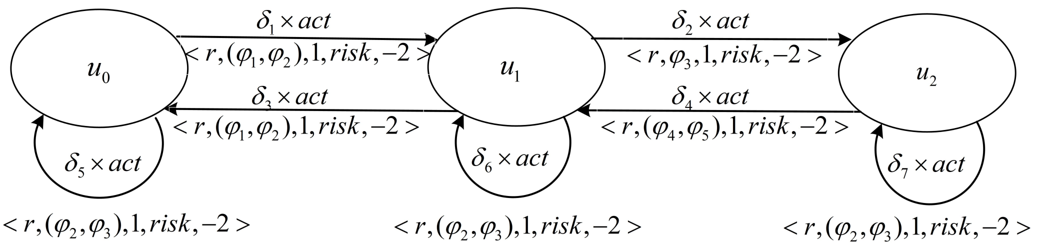

u_0
u_1
u_2
$\delta_1 \times act$
$< r,(\varphi_1,\varphi_2),1,risk,-2 >$
$\delta_2 \times act$
$< r,\varphi_3,1,risk,-2 >$
$\delta_3 \times act$
$< r,(\varphi_1,\varphi_2),1,risk,-2 >$
$\delta_4 \times act$
$< r,(\varphi_4,\varphi_5),1,risk,-2 >$
$\delta_5 \times act$
$\delta_6 \times act$
$\delta_7 \times act$
$< r,(\varphi_2,\varphi_3),1,risk,-2 >$
$< r,(\varphi_2,\varphi_3),1,risk,-2 >$
$< r,(\varphi_2,\varphi_3),1,risk,-2 >$

Figure 4. Reward machine.

Figure 4 shows the state change of an autonomous vehicle in a three-lane highway. $U = (u_0, u_1, u_2)$ represents a high-level state in the right lane, the middle lane, and the left lane. $\delta = (\delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_6, \delta_7)$ represents a state transition, which is determined by the previous state and the current state. Where $< r, (\varphi_1, \varphi_2), \varphi_r, risk, risk_r >$ is the reward function for changing from the right lane state u_0 to the middle lane u_1, and r is the set basic reward. (φ_1, φ_2) denotes the property constraint of changing lanes at a safe distance and whether there is speeding, respectively, φ_r denotes the positive feedback reward of 1 when the property φ is true. *risk* is the risk probability of identifying the nondeterministic environment, which is obtained by quantifying the nondeterministic environment risk mentioned above, $risk_r$ is -2, and the coefficient of negative feedback given to the probability risk function is -2.

6.3. Feedback Based on Safety Monitoring Barrier

The ability of a CPS real-time system to react to surrounding data in a timely and accurate manner and then hand it over to the controller for response is critical to safe system operation. It is also possible to quickly determine whether the system will be impacted in a way that is not expected. To monitor the system's operation, a runtime monitoring barrier was established. Its safety was verified for various safety properties, i.e., whether the safety constraints were violated and that the controller performed safe operations. Figure 5 depicts the structure of the monitoring barrier.

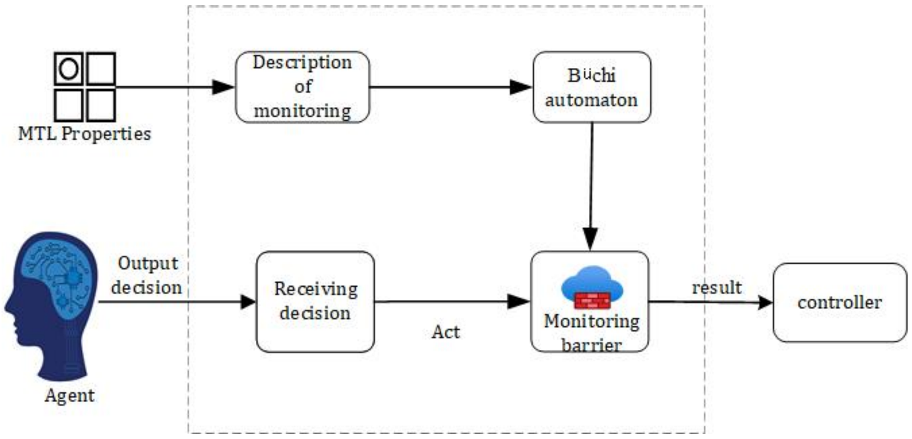


Figure 5. Monitoring barrier.

The monitoring barrier intercepts the intelligence's decision, accepts the decision action as input, and determines whether the action decision satisfies the safety property constraint. The monitoring barrier has two primary functions: satisfaction verification and safety assurance output.

Definition 7. *Satisfaction verification. When the monitor verifies the satisfaction of a property, it can be considered as a Boolean function output to determine whether the current system state satisfies the safety property constraint through the state migration triggered by the selected action. It is expressed as*

$$(s, act, \varphi) \rightarrow bool$$

where $s \in S$ is the current state, $act \in acts$ is the selected action, φ is the safety property specification, and the output is $\{true, false\}$.

Definition 8. *Safety assurance output. When the input action does not satisfy the property statute, a safe new action should be selected from the set of candidate actions.*

6.4. Generation Algorithm of Safe Decision Controller

In this section, monitoring barriers transformed by safety properties are used in conjunction with the DRL algorithm to create a controller capable of making safe decisions through training. We propose a general safety learning algorithm, which sets the reward function guided by the safety property, and then uses the monitoring barrier to verify the output of the algorithm, so as to ensure the safety of the output decision and to be able to apply the safety RL algorithm to the safety-critical CPS system. By adding a monitor to the DRL algorithm as the intelligence's supervisory layer, the information exchange between the environment and the intelligence is constantly monitored, and the decision's safety is judged through verification, so that when a risk is perceived, a safe decision can be provided, and the risk can be avoided as much as possible.

Algorithm 1 gives the algorithm for solving the optimal action for the conventional DQN. In the traditional DQN algorithm, the agent selects an action with probability p, then obtains immediate returns and subsequent states from the environment feedback, and finally performs the update function.

Algorithm 1 DQN Algorithm

```
Input: $MDP = (S, A, T, R), done, update$
Output: Optimal policy $\pi$
 1: Init ($\pi$)
 2: $s_t \leftarrow s_0$
 3: $a \leftarrow NOP$
 4: while $s_t != done$ and $episode < episodes$ do
 5:     // Iterative selection of optimal value
 6:     $A' = A$
 7:     $a \leftarrow Random_Select(A')$
 8:     $update(s_t, a, \pi)$ // Update parameters
 9:     Go to the next state $s_{t+1}$
10: end while
```

However, when faced with inexperienced scenarios and complex decision-making tasks, black-box-based DRL systems do not guarantee the safety of the system and the complex interpretability of the reward function settings for complex tasks, so using the traditional DQN algorithm to generate decisions does not work very well. In the following, the nondeterministic environment model is combined into the algorithm implementation based on the methodology from the previous sections.

Algorithm 2 shows the algorithm for safe decision-making generation. When the environment is accurately modeled, the system selects from a set of validated safety actions; otherwise, the system selects conservative policy actions from the action space. The inputs are a reinforcement model $MDP = (S, A, T, R)$, a reward machine $\tau_r = < r, \varphi, \varphi_r, risk, risk_r >$, an end state, and an update of a function that records state transitions. The monitoring barrier is used to monitor the system for previously observed states, previous control actions, and the current state to satisfy safety requirements. The controller receives the decision whenever the monitoring barrier is true.

Algorithm 2 Generation algorithm for the safe decision controller

```
Input: $MDP = (S, A, T, R), \tau_r = < r, \varphi, \varphi_r, risk, risk_r >, done, update$
Output: Optimal safety policy $\pi$
 1: Init ($\pi$)
 2: $s_t \leftarrow s_0$
 3: $a \leftarrow NOP$
 4: while $s_t != done$ and $episode < episodes$ do
 5:     // Iterative selection of optimal value
 6:     $A' = A$
 7:     $a \leftarrow choose(A')$ // Select action
 8:     $env \leftarrow$ Environmental sampling
 9:     while $monitor(s_t, a, env)$ do // monitor
10:         $A' \leftarrow A' - a$
11:         if $A' != \varnothing$ then
12:             $a \leftarrow choose(A')$
13:         else
14:             $a \leftarrow choose(a_{\text{conservative}} \in A)$
15:             break
16:         end if
17:     end while
18:     $update(s_t, a, \pi)$ // Update parameters
19:     Go to the next state $s_{t+1}$
20: end while
```

Lines 1–3 show the process starting in its initial state. Lines 4–20 represent the choice of the next action and its execution until the termination state is reached or the maximum number of rounds is reached. Lines 6–7 describe the chosen action. Line 8 describes the process of acquiring environmental parameters. Lines 9–17 express the monitoring barrier, which monitors the state, environment, and the selected action; if it returns true, an action is output; otherwise, the greedy idea is used to choose another action from the action subset; and if none of them return true, a conservative policy action is output. Line 18 updates the state based on the environment and the learning model. The entire algorithm consists of two layers of loops, the first of which is about the abort state and the number of training rounds (episodes). The second loop is a monitor that selects the optimal values from the action space for the output. The algorithm complexity is $O(n^2)$.

7. Evaluation

7.1. Training Details

In our study, we used the path-planning project simulator from the Udacity self-driving car project [35]. The simulator can provide the vehicle's positioning and sensor fusion data, obtain the map information around the vehicle, give a visual graphical interface, and produce an effective definition and simulation of time and position. It also provides a visual interface and a controller module, which can communicate information with each other through sockets. The observation-state information data in the simulation environment was transmitted to the algorithm, and then the high-level action decisions calculated by the algorithm were transmitted back to the target vehicle agent. The simulator is shown in Figure 6.

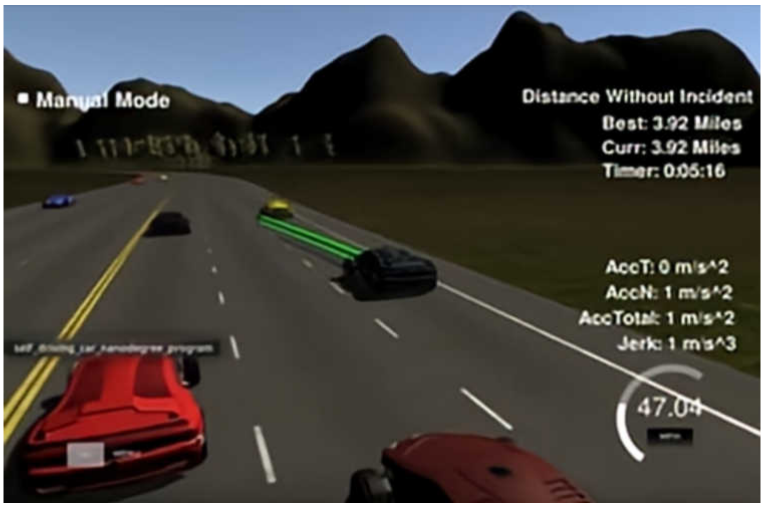


Figure 6. Simulation environment.

The experimental scenario was designed as a self-driving car traveling at 50 MPH on a highway with surrounding vehicles traveling at 40–60 MPH. The car should drive as close to the speed limit as possible. When facing a blocked road section, it should try to change lanes to overtake. At the same time, the vehicle should ensure safety when making lane-change decisions. The experimental environment we used is shown in Table 2. As for the safety properties, we chose a safe distance for side vehicles when changing lanes. The property is Section 6.1—R3.

The simulation environment was a three-lane highway road map. A 45 $\times$ 3 matrix was used as the road's state-information representation, with three columns representing the corresponding three lanes and each vehicle in each lane occupying four cells. The number filled in the matrix was the speed of the vehicle. The action space was chosen as $a = a_0, a_1, a_2$, a_0 for keeping the current lane, with a_1 for changing lanes to the left and a_2 for changing lanes to the right. The reward function was defined by the reward machine in Section 6.

The algorithm used in this paper was based on Tensorflow's DQN algorithm [36]. We improved the safe DQN algorithm by incorporating safety monitoring to implement the car intelligent lane-change decision. In our study, we trained the two algorithms for 100 rounds of deep reinforcement, with each round consisting of a loop on a 6946-meter-long highway, which took about 6 min to complete. We set all the parameter weights of nondeterministic probabilities to 1/7. During the training process, after each round of the task was completed, the vehicle agent returned to the initial point for the next round of training.

Table 2. Experimental settings for the simulation.

Experimental Settings
Processor: Intel Xeon W-2225
Video card: NVIDIA Quadro rtx5000, 16 GB
Memory: 128 GB
Operating system: Ubuntu 18.04
Development language: Python 3.6
Development framework: tensorflow 2.4.0

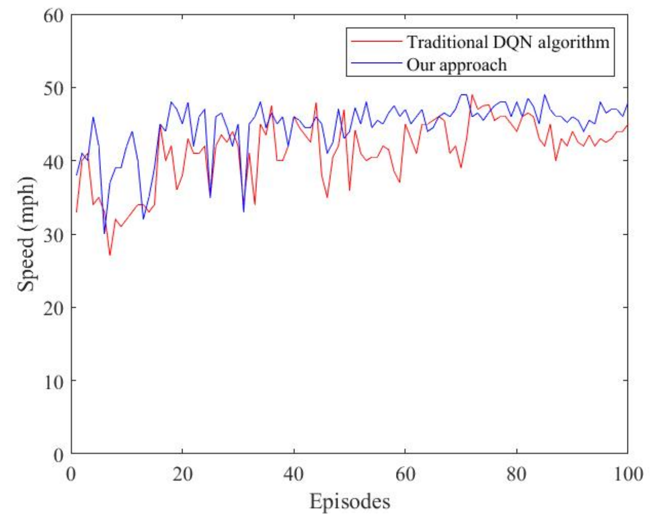
7.2. Comparing with Tensorflow's DQN Algorithm

We compared the average speed, lane-change time, and collision time for each phase of the two methods.

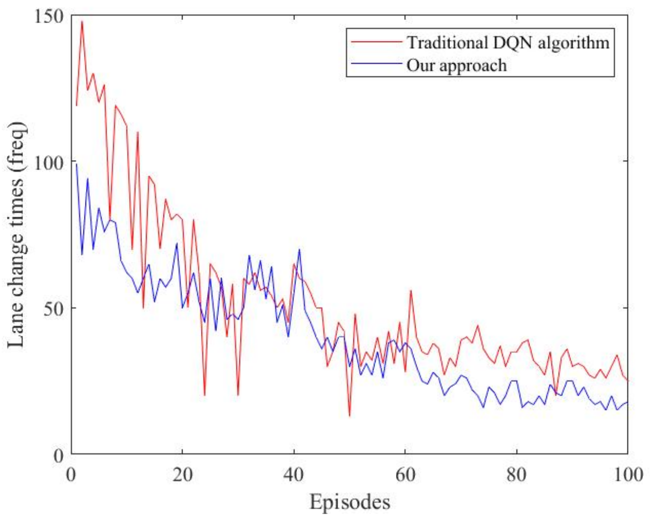
Figure 7 shows that the car's behavior was ideal. Figure 7a shows the average speed obtained by the algorithm in each round of training. As the number of rounds increased, the average speed was closer to the speed threshold, and the vehicle agent could complete the route task more efficiently. The vehicle's speed was greatly improved in the 20th round of training. At the same time, when the training round reached 72 rounds, the safety RL algorithm could achieve speed stability faster; Figure 7b shows the number of lane-changing events in each round. The frequency of lane changing was significantly reduced, and the average vehicle speed was increased. A faster speed was obtained by fewer lane-changing behaviors, and a faster speed in the task was met to reach the destination. Figure 7c shows the number of collisions per round. The number of intelligent vehicle collisions decreased with the increase in the number of training rounds, which indicated that the intelligent vehicle learned correct driving actions through exploration and RL, reducing the number of collisions.

The graph shows a counterexample at rounds 30–40, when the algorithm had not yet stabilized, which produced a high collision rate. The figure compares the traditional way of designing the DQN algorithm with our method over the course of 100 training rounds. The overall average increase in driving speed was 3.371 MPH, the overall average reduction in the number of lane changes was 11.64, and the overall average reduction in the number of collisions was 25.52. Collisions could be avoided as much as possible by using the safety RL algorithm; however, the risk of collision still existed in the experiment, which we observed through the visual interface when the distance between the vehicle and the rear vehicle was too small when changing lanes, causing a collision when decelerating. From our experimental results, we observed that the use of safety monitoring in RL could make the agent's decision-making safer while ensuring the algorithm's decision-making output converged faster.

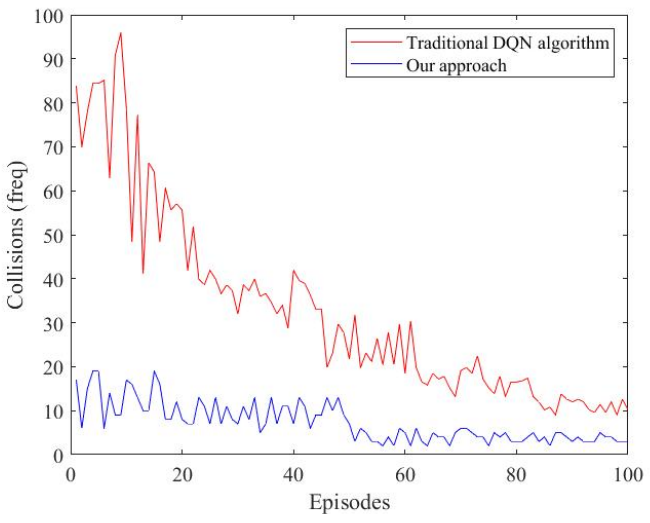
We compared the trained model with several different approaches, including a random lane-change strategy with rule constraints, a rule-based lane-change strategy, and a pure DQN lane-change strategy. Compared with several other methods, the car's average execution speed in our method was slightly better. Additionally, our method had a higher security rate than several of the remaining policies, which ensured the safety of the decision. This showed that our approach achieved a more efficient and safer performance compared with other methods (see Table 3).



(**a**)



(**b**)



(**c**)

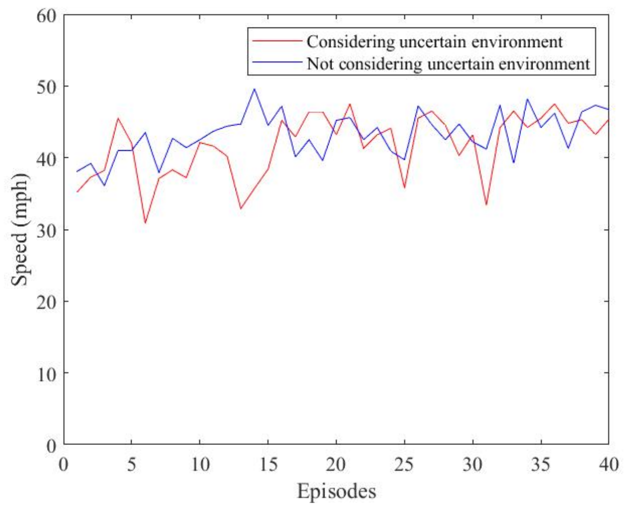
Figure 7. Comparison of traditional and improved methods. (**a**) Comparing speed. (**b**) Comparing lane-change times. (**c**) Comparing collisions.

Table 3. Advantages and disadvantages of different methods.

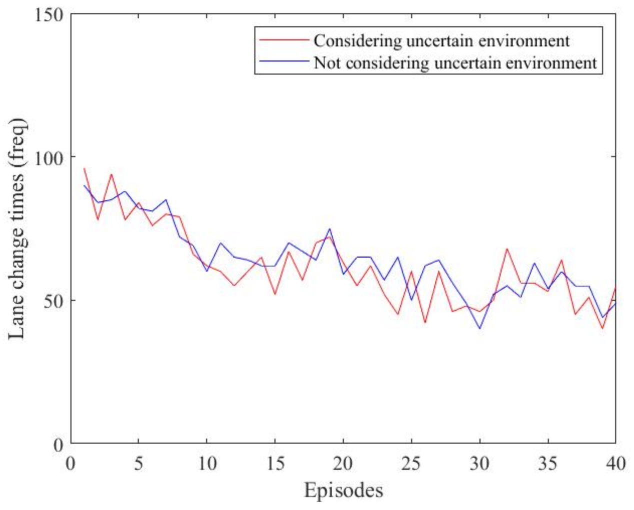
Method	Avg Speed	Avg Lane Changes	Safety Rate
Random action [37]	44.59	152.60	0.6
Rule-based [38]	45.22	8.40	0.6
DQN-based [39]	46.16	37.40	0.2
Our method	46.73	11.80	0.9

7.3. Assessing the Effect of the Nondeterministic Environment

Figure 8 shows a comparison experiment between the monitor considering the effect of the nondeterministic environment and the case without it. We intercepted the data from rounds 0–40. In the figure, we compared the effect of considering environmental factors with not considering environmental factors and analyzed the first 40 rounds with significant differences. Our method showed a significant reduction in the number of collisions; however, at the same time, the driving speed was slightly lower compared with the conventional method. This is because our method considered the uncertain driving style variation, and if there was a high probability of risk, it chose to follow the car, resulting in a lower speed.

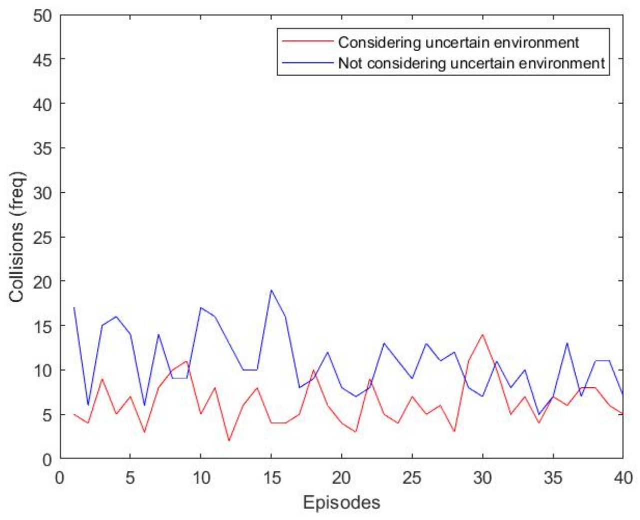


(**a**)



(**b**)

Figure 8. *Cont.*



(**c**)

Figure 8. Comparison of nondeterministic environmental impact. (**a**) Comparison of speed under environmental factors. (**b**) Comparison of lane change times under environmental factors. (**c**) Comparison of collisions under environmental factors.

Through the above analysis, our method overcame the disadvantage of not being able to guarantee safety in the nondeterministic environment compared with the traditional method and achieved the safety requirements of autonomous driving.

8. Conclusions

In our study, we proposed a DRL-based generation method for autonomous driving CPS decision controllers to address the problem of obtaining a safe decision controller for CPS systems in nondeterministic environments. First, we obtained a simulation environment by accurately modeling the real environment of the autonomous driving system, from which we obtained nondeterministic environmental factors. In the second step, we constructed a nondeterministic environment model by analyzing the classifiers of risk factors and safety properties, and then we defined the environment model based on a formal approach of the constraints. In the third step, we constructed the environment model as both a reward machine and a monitor. We used the reward machine to guide the reward function in RL and utilized the monitor to set the monitoring barrier, which monitored the output actions. If the safety properties were not satisfied, we selected the action that satisfied the safety properties from the action set to ensure the decision. The fourth step was the routine execution of reinforcement learning. Data were fed to the agent from the simulation environment, after which an action was obtained for feedback. Through experiments, we verified the feasibility of our method, and the addition of our method ensured the safety of the decisions. We verified the practicability of formal modeling and verification methods in nondeterministic environments and strengthened the learning process from safety property specification and formal modeling constraints. The definition of a reward function in a practical application can be explained by the formal method. Our method provides an idea for the combination of formal methods and reinforcement learning.

However, our work was carried out in a three-lane motorway environment, which still lacks validation for more complex scenarios. In future work, we also need to consider optimizing the design and training of reinforcement learning safety constraints, considering the influence of traffic rules. Secondly, we can also improve the reward–reward function of the reinforcement learning by making more precise definitions to improve the efficiency of learning and the interpretability of learning strategies. Finally, simulations should be validated with more nondeterministic environmental factors to gradually promote the diffusion and practical application of safety-based reinforcement learning in unmanned systems.

Author Contributions: Conceptualization, H.C. and Y.Z.; methodology, H.C. and Y.Z.; software, H.C. and Y.Z.; validation, H.C. and Y.Z.; formal analysis, H.C. and Y.Z.; investigation, H.C. and Y.Z.; resources, H.C., Y.Z. and M.H.; writing—original draft preparation, H.C., U.A.B. and Y.Z.; writing—review and editing, H.C., U.A.B., Y.Z. and M.H.; visualization, H.C. and Y.Z.; supervision, H.C., Y.Z. and M.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the National Natural Science Foundation of China (grant no. 62062030), in part by the Key R&D Project of Hainan province (grant no. ZDYF2021SHFZ243), in part by the Major Science and Technology Project of Haikou (grant no. 2020-009).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Luo, J.H.; Xiao, Z.H.; Zhong, C.P. Analysis of the development trend of information physics systems. *Telecommun. Sci.* **2012**, *28*, 127–132. (In Chinese)
2. Xie, Y.; Zeng, G.; Kurachi, R.; Xiao, F.; Takada, H.; Hu, S. Timing Analysis of CAN FD for Security-Aware Automotive Cyber-Physical Systems. *IEEE Trans. Dependable Secur. Comput.* **2022**. *99*, 1–14.
3. García, J.; Fernández, F. A comprehensive surveyon safe reinforcement learning. *J. Mach. Learn. Res.* **2015**, *16*, 1437–1480.
4. Moldovan, T.M.; Abbeel, P. Safe exploration in Markov decision processes. In Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, UK, 26 June–1 July 2012.
5. Tamar, A.; Xu, H.; Mannor, S. Scaling up robust MDPs by reinforcement learning. *arXiv* **2013**, arXiv:1306.6189.
6. Katz, G.; Barrett, C.W.; Dill, D.L.; Julian, K.; Kochen-Derfer, M.J. Reluplex: An efficient SMT solver for verifying deep neural networks. In Proceedings of the Computer Aided V Erification—29th International Conference, CAV 2017, Heidelberg, Germany, 24–28 July 2017; Part I, pp. 97–117.
7. Arnold, T.; Kasenberg, D.; Scheutz, M. Value Alignment or Misalignment—What Will Keep Systems Accountable? *AAAI Workshops*; AAAI Press: Palo Alto, CA, USA, 2017.
8. Leike, J.; Krueger, D.; Everitt, T.; Martic, M.; Maini, V.; Legg, S. Scalable agent alignment via reward modeling: A research direction. *arXiv* **2018**, arXiv:1811.07871.
9. Christiano, P.F.; Abate, M.; Amodei, D.; Supervising strong learners by amplifying weak experts. *arXiv* **2018**, arXiv:1810.08575.
10. Hadfield-Menell, D.; Russell, S.J.; Abbeel, P.; Dragan, A. Cooperative inverse reinforcement learning. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS 2016), Barcelona, Spain, 5–10 December 2016; pp. 3909–3917.
11. Mason, G.; Calinescu, R.; Kudenko, D.; Banks, A. Assured reinforcement learning with formally verified abstract policies. In Proceedings of the 9th International Conference on Agents and Artificial Intelligence (ICAART 2017), Porto, Portugal, 24–26 February 2017; pp. 105–117.
12. Cheng, R.; Orosz, G.; Murray, R.M.; Burdick, J.W. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019. [CrossRef]
13. Lütjens, B.; Everett, M.; How, J.P. Safe Reinforcement Learning with Model Uncertainty Estimates. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 8662–8668. [CrossRef]
14. Talamini, J.; Bartoli, A.; De Lorenzo, A.; Medvet, E. On the Impact of the Rules on Autonomous Drive Learning. *Appl. Sci.* **2020**, *10*, 2394. [CrossRef]
15. Krasowski, H.; Wang, X.; Althoff, M. Safe Reinforcement Learning for Autonomous Lane Changing Using Set-Based Prediction. In Proceedings of the 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), Rhodes, Greece, 20–23 September 2020. [CrossRef]
16. Wachi, A.; Sui, Y. Safe Reinforcement Learning in Constrained Markov Decision Processes. In Proceedings of the 37th International Conference on Machine Learning, Virtual, 13–18 July 2020; pp. 9797–9806.
17. Bastani, O.; Pu, Y.; Solar-Lezama, A. Verifiable reinforcement learning via policy extraction. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS 2018), Montreal, QC, Canada, 3–8 December 2018; pp. 2494–2504.
18. De Giacomo, G.; Iocchi, L.; Favorito, M.; Patrizi, F. Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications. In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 2019), Berkeley, CA, USA, 11–15 July 2019; pp. 128–136.
19. Camacho, A.; Chen, O.; Sanner, S.; Mcllraith, S.A. Non-Markovian rewards expressed in LTL: Guiding search via reward shaping. In Proceedings of the 10th Annual Symposium on Combinatorial Search (SoCS 2017), Pittsburgh, PA, USA, 16-17 June 2017; pp. 159–160.

20. Aksaray, D.; Jones, A.; Kong, Z.; Schwager, M.; Belta, C. Q-learning for robust satisfaction of signal temporal logic specifications. In Proceedings of the IEEE 55th Conference on Decision and Control (CDC 2016), Las Vegas, NV, USA, 12–14 December 2016; pp. 6565–6570.
21. Balakrishnan, A.; Deshmukh, J. Structured reward functions using STL. In Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2019), Montreal, QC, Canada, 16–18 April 2019; pp. 270–271.
22. Wen, M.; Papusha, I.; Topcu, U. Learning from demonstrations with high-level side information. In Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017), Melbourne, Australia, 19–25 August 2017; pp. 3055–3061.
23. Alshiekh, M.; Bloem, R.; Ehlers, R.; Könighofer, B.; Niekum, S.; Topcu, U. Safe reinforcement learning via shielding. In Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018), New Orleans, LA, USA, 2–7 February 2018; pp. 2669–2678.
24. Hahn, E.M.; Perez, M.; Schewe, S.; Somenzi, F.; Trivedi, A.; Wojtczak, D. Omega-regular objectives in model-free reinforcement learning. In Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2019), Prague, Czech Republic, 8–11 April 2019; Springer: Berlin/Heidelberg, Germany, pp. 395–412.
25. Icarte, R.T.; Klassen, T.Q.; Valenzano, R.A.; Mcllraith, S.A. Using reward machines for high-level task specification and decomposition in reinforcement learning. In Proceedings of the International Conference on Machine Learning (ICML 2018), Stockholm, Sweden, 10–15 July 2018; pp. 2112–2121.
26. Araki, B.; Vodrahalli, K.; Leech, T.; Vasile, C.I.; Donahue, M.; Rus, D. Learning to plan with logical automata. In Proceedings of the Robotic: Science and Systems (RSS 2019), Breisgau, Germany, 22–26 June 2019; pp. 1–9.
27. Li, G.; Yang, Y.; Li, S.; Qu, X.; Lyu, N.; Li, S.E. Decision making of autonomous vehicles in lane change scenarios: Deep reinforcement learning approaches with risk awareness. *Transp. Res. Part C Emerg. Technol.* **2021**, *134*, 103452. 10.1016/j.trc.2021.103452. [CrossRef]
28. Muzahid, A.J.M.; Rahim, M.A.; Murad, S.A.; Kamarulzaman, S.F.; Rahman, M.A. Optimal Safety Planning and Driving Decision-Making for Multiple Autonomous Vehicles: A Learning Based Approach. In Proceedings of the 2021 Emerging Technology in Computing, Communication and Electronics (ETCCE), Dhaka, Bangladesh, 21–23 December 2021; pp. 1–6. 10.1109/ETCCE54784.2021.9689820. [CrossRef]
29. Pnueli, A. The temporal logic of programs. In Proceedings of the 18th Annual Symposium on Foundations of Computer Science, Washington, DC, USA, 30 September–31 October 1977; pp. 46–57.
30. Thati, P.; Roşu, G. Monitoring Algorithms for Metric Temporal Logic Specifications. *Electron. Notes Theor. Comput. Sci.* **2005**, *113*, 145–162. [CrossRef]
31. Kober, J.; Peters, J. *Reinforcement Learning in Robotics: A Survey*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 579–610.
32. Tan, C.; Zhang, J.X.; Wang, T.X.; Yue, T. Uncertainty-wise software engineering of complex systems: A systematic mapping study. *J. Softw.* **2021**, *32*, 1926–1956.
33. Tan, D.D.; Liu, J.; Chen, X.H.; Sun, H.Y. Formal modeling and dynamic verification for human cyber physical systems under uncertain environment. *J. Softw.* **2021**, *32*, 1999–2015.
34. Chen, C.; Lü; N.; Liu, L.; Pei, Q.Q.; Li, X.J. Critical safe distance design to improve driving safety based on vehicle-to-vehicle communications. *J. Cent. South Univ.* **2013**, *20*, 3334–3344. [CrossRef]
35. Virgo, M.; Brown, A. Self-Driving Car Engineer Nanodegree Program. Available online: https://github.com/udacity/CarND-Path-Planning-Project (accessed on 20 August 2022).
36. Wang, J.; Zhang, Q.; Zhao, D. Highway lane change decision-making via attention-based deep reinforcement learning. *IEEE/CAA J. Autom. Sin.* **2022**, *9*, 567–569. [CrossRef]
37. Bojarski, M.; Testa, D.D.; Dworakowski, D.; Firner, B.; Flepp, B.; Goyal, P.; Jackel, L.D.; Monfort, M.; Muller, U.; Zhang, J.; et al. End to End Learning for Self-Driving Cars. *ArXiv* **2016**, arXiv:1604.07316.
38. Li, S.; Shu, K.; Chen, C.; Cao, D. Planning and Decision-Making for Connected Autonomous Vehicles at Road Intersections: A Review. *Chin. J. Mech. Eng.* **2021**, *34*, 133. [CrossRef]
39. 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*, 529–533. 10.1038/nature14236. [CrossRef] [PubMed]