

Real-World Autonomous Driving Control: An Empirical Study Using the Proximal Policy Optimization (PPO) Algorithm

Zhao, Peng

Interdisciplinary Graduate School of Engineering Sciences, Kyushu University

Yuan, Zhongxian

Faculty of Environment and Life, Beijing University of Technology

Thu, Kyaw

Interdisciplinary Graduate School of Engineering Sciences, Kyushu University

Miyazaki, Takahiko

Interdisciplinary Graduate School of Engineering Sciences, Kyushu University

<https://doi.org/10.5109/7183372>

出版情報 : Evergreen. 11 (2), pp.887-899, 2024-06. 九州大学グリーンテクノロジー研究教育センター
バージョン :

権利関係 : Creative Commons Attribution 4.0 International



Real-World Autonomous Driving Control: An Empirical Study Using the Proximal Policy Optimization (PPO) Algorithm

Peng Zhao^{1,*}, Zhongxian Yuan², Kyaw Thu¹, Takahiko Miyazaki¹

¹Interdisciplinary Graduate School of Engineering Sciences, Kyushu University, Japan

²Faculty of Environment and Life, Beijing University of Technology, China

E-mail: zhao.peng.488@s.kyushu-u.ac.jp

(Received December 7, 2023; Revised April 16, 2024; Accepted April 24, 2024).

Abstract: This article preprocesses environmental information and use it as input for the Proximal Policy Optimization (PPO) algorithm. The algorithm is directly trained on a model vehicle in a real environment, allowing it to control the distance between the vehicle and surrounding objects. The training converges after approximately 200 episodes, demonstrating the PPO algorithm's ability to tolerate uncertainty, noise, and interference in a real training environment to some extent. Furthermore, tests of the trained model in different scenarios reveal that even when the input information is processed and does not provide a comprehensive view of the environment, the PPO algorithm can still effectively achieve control objectives and accomplish challenging tasks.

Keywords: Reinforce Learning; Proximal Policy Optimization; Autonomous Driving

1. Introduction

In the field of autonomous driving control, there are three control categories based on the different ways of processing environmental information: indirect perception, direct perception, and behavior reflex(end-to-end) ^{1,2}. Indirect perception controls vehicles through classification and induction of the environment; behavior reflex requires no processing of environmental information; and direct perception lies between the two, controlling vehicles by processing partial environmental information. With the development and application of machine learning technologies, the latter two methods are gradually demonstrating their advantages in simplifying systems and reducing computational resource requirements, potentially replacing the more complex indirect perception approach.

With the enhancement of computational capabilities, deep learning has experienced rapid advancement, finding successful applications in diverse fields such as agriculture^{3,4}, architecture⁵, and road traffic management^{6,7}. Further, the integration of deep learning with reinforcement learning, which embodies a methodology capable of autonomous strategy learning, has facilitated the application of reinforcement learning in increasingly complex environments. This synergy of deep learning and reinforcement learning, often referred to as deep reinforcement learning, has demonstrated successful applications across multiple domains ⁸⁻¹⁰. Particularly in

areas such as path planning¹¹⁻¹³ and motion control¹⁴⁻¹⁶ for unmanned device control, reinforcement learning exhibits significant potential due to its advantage of not requiring prior model construction. However, in autonomous driving applications, reinforcement learning still faces certain limitations. Currently, many studies on reinforcement learning in autonomous driving control are limited to training in simulated environments^{14,17,18}. The simulation-reality gap, arising from differences between real-world and simulated scenarios, restricts the application of well-trained reinforcement learning algorithms in actual settings^{19,20}. Additionally, the design of reward functions is a key factor constraining the development of reinforcement learning in autonomous driving control²¹.

With the development of policy-based reinforcement learning methods, reinforcement learning has become more suitable for addressing continuous action space problems. However, many researchers still tend to use value-based reinforcement learning algorithms, such as DDQN, to solve continuous action space control problems. In 2017, OpenAI introduced the Proximal Policy Optimization (PPO) algorithm²², which, due to its lower reward function requirements and low dependency on algorithm parameters, shows tremendous application potential in the field of autonomous driving control.

In this paper, we employ the PPO algorithm to control vehicles by obtaining limited environmental information, aiming to verify the robustness and generalization ability

of the PPO algorithm under input information constraints. Furthermore, both the training and testing of this study are conducted on actual physical models, attempting to train algorithms in real-world environments, and exploring the feasibility of an alternative solution for applying reinforcement learning algorithms from simulated to real-world environments.

2. Relative Works

Following the birth of the DQN algorithm, which combines Q-learning with deep learning, and the surpassing of human performance in the game of Go by AlphaGo, which was trained using the DQN algorithm, reinforcement learning has entered a rapid development stage. From value-based Q-learning algorithms, such as DQN, DDQN, Dueling-DQN, and D3QN, to policy optimization-based algorithms, such as PG, Actor-Critic, DDPG, TRPO, and PPO, these methods have found mature applications in various fields, including gaming, robot control, finance, and decision support. In particular, the policy optimization-based algorithms have shown outstanding capabilities in handling continuous action spaces, making them more promising in various continuous action application scenarios.

In recent years, scholars have been studying the application of reinforcement learning to the field of autonomous driving. Wolf⁽²³⁾ and Lillicrap⁽²⁴⁾ successfully implemented vehicle control in simulations using reinforcement learning algorithms, while Kendall⁽²⁵⁾ successfully achieved real vehicle motion control using reinforcement learning. However, the models established by reinforcement learning have a "black box" characteristic, which makes it difficult to fine-tune the model in response to emerging problems. On the other hand, real-world driving conditions are diverse, and current technological means cannot collect data for all scenarios, leading to simulation-based reinforcement learning models that may not be well-suited to all driving conditions. Therefore, improving the generalization ability of the model and addressing the simulation-reality gap are urgent issues to be solved. Consequently, in the field of behavior reflex (end-to-end) autonomous driving solutions, academia is more focused on discussing how to build better algorithms to mitigate or solve the aforementioned problems.

Bojarski⁽²⁶⁾ and Kim⁽²⁷⁾ proposed using the extraction of convolutional layer feature maps and highlighting salient objects, as well as using a visual attention model, to solve the problem of how deep neural networks understand environmental information. Wenshuai Zhao⁽²⁰⁾ and B Ravi Kiran⁽²¹⁾ summarized the current solutions to the simulation-reality gap in their articles. Researchers are mainly exploring autonomous driving control by employing different algorithms or improving existing algorithms. Zhu M⁽¹⁴⁾ and others proposed a new velocity control during car following model and compared it with an adaptive cruise control (ACC) algorithm based on

model predictive control (MPC). Chen et al.⁽²⁸⁾ proposed a human-in-the-loop deep reinforcement learning method, which improves the deep neural network reward model through fuzzy evaluations provided by human drivers. Liang et al.⁽²⁹⁾ aimed to enhance the exploration efficiency of the DDPG algorithm in high-dimensional action spaces by using expert data to constrain the action space, ensuring that the algorithm always explores within feasible regions.

The direct perception method proposed in paper⁽¹⁾ pre-processes environmental information to some extent, thereby avoiding the "black box" problem in reinforcement learning and significantly reducing the training cost of reinforcement learning algorithms. The feasibility of training vehicles using this method on real objects, as well as the impact of this method on the robustness and generalization of the algorithm, are the topics of investigation and discussion in this paper.

3. Methodology

3.1 Policy-based Reinforcement Learning Algorithm

Reinforcement learning can be divided into two types of algorithms: value-based algorithms and policy-based algorithms. From a fundamental perspective, value-based algorithms require reward output as a prerequisite for policy optimization. Each action is represented using its corresponding action value. However, action values cannot be mapped to a single distribution, thus hindering the agent's ability to output continuous actions. Policy-based algorithms use a policy function to output actions, which can be mapped to a distribution. Actions are represented using action probabilities, making them well-suited for continuous action spaces (high-dimensional). In the actual operation of vehicles, changes in speed and direction are not determined by only a few specific speed and angle values. For example, there are countless speed values between 0 and 1 m/s, and similarly, there are countless steering angles between 0° and 45°. Therefore, when applying reinforcement learning to vehicle control, policy-based algorithms have advantages.

The purpose of policy gradient algorithm is to find the optimal policy without using the Q-function. This algorithm optimizes the policy function $\pi(a|s)$ by adjusting the parameters θ , such that it becomes $\pi(a|s; \theta)$. Given a sequence of state-action pairs $= s_0, a_0, s_1, a_1, \dots, s_l, a_l$, the objective function of policy gradient is:

$$J(\theta) = E(\sum_{i=0}^l R(s_i, a_i); \pi_{\theta}) = \sum_r P(\tau; \theta) R(\tau) \quad (1)$$

Here, $R(\tau) = \sum_{i=0}^l R(s_i, a_i)$ represents the return of the sequence, and $P(\tau; \theta)$ represents the probability of the sequence.

When solving with gradient ascent, we have:

$$\theta = \theta + \alpha \nabla_{\theta} J(\theta) \quad (2)$$

Taking the derivative of $J(\theta)$ with respect to θ yields:

$$\nabla_{\theta} J(\theta) = \sum_{\tau} P(\tau; \theta) \nabla_{\theta} \log P(\tau; \theta) R(\tau) \quad (3)$$

After sampling m trajectories, the policy gradient can be approximated by the average of the experiences:

$$\nabla_{\theta} J(\theta) = \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} \log P(\tau; \theta) R(\tau) \quad (4)$$

The likelihood of each sequence τ is:

$$P(\tau; \theta) = \prod_{t=0}^l P(s_{t+1}^i | s_t^i, a_t^i) \pi_0(a_t^i | s_t^i) \quad (5)$$

where $P(s_{t+1}^i | s_t^i, a_t^i)$ represents the dynamics, independent of the parameter θ . Substituting it into (4) yields:

$$\nabla_{\theta} J(\theta) = \frac{1}{m} \sum_{i=1}^m \left(\sum_{t=0}^l \nabla_{\theta} \log \pi_0(a_t^i | s_t^i) R(\tau^i) \right) \quad (6)$$

The policy gradient has a large variance, making it impossible to extract every action during action sampling. A constant baseline b is typically introduced to reduce variance:

$$\nabla_{\theta} J(\theta) = \frac{1}{m} \sum_{i=1}^m \left(\sum_{t=0}^l \nabla_{\theta} \log \pi_0(a_t^i | s_t^i) R(\tau^i) - b \right) \quad (7)$$

The value of b can be obtained by taking the derivative of the variance with respect to b and setting it to zero.

According to the PG principle, the pseudocode of the PG algorithm can be represented as follows:

Algorithm 1 Policy Gradient (PG)

```

1: Initialize policy parameters  $\theta$ 
2: for each iteration do
3:   Collect  $m$  trajectories  $\tau_i$  using policy  $\pi_{\theta}$ 
4:   Compute the rewards  $R(\tau^i)$  for each trajectory  $\tau_i$ 
5:   for  $t = 0$  to  $l$  do
6:     Compute the gradient  $\nabla_{\theta} J(\theta)$  using Equation (7)
7:   Update policy parameters  $\theta$  using Equation (8):  $\theta = \theta + \alpha \nabla_{\theta} J(\theta)$ 
8: end for
9: end for
    
```

Fig. 1: Pseudocode of the PG algorithm.

According to the policy gradient method, the parameter update equation can be written as:

$$\theta_{new} = \theta_{old} + \alpha \nabla_{\theta} J \quad (8)$$

Here, α is the step size of the update, which directly determines the quality of the new policy. If α is inappropriate, the updated policy may be worse, resulting in a worse learning process. To ensure that the new policy can make the reward function monotonically non-decreasing and prevent the learning process from deteriorating, the Trust Region Policy Optimization (TRPO) algorithm was proposed. In TRPO, the Kullback-Leibler (KL) divergence is used to add a trust region constraint to the policy gradient method, ensuring that the difference between the new and old policies is small.

The reward function for a state-action sequence τ can be expressed as:

$$\eta(\pi) = E_{\tau|\pi} [\sum_{t=0}^{\infty} \gamma^t r(s_t)] \quad (9)$$

If the new policy is π' and it is better than the old policy, the expected reward of π' can be defined as:

$$\eta(\pi') = \eta(\pi) + E_{s_0, a_0, \dots, \pi'} [\sum_{t=0}^{\infty} \gamma^t A_{\pi}(s_t, a_t)] \quad (10)$$

Here, $A_{\pi}(s_t, a_t) = Q_{\pi}(s_t, a_t) - V_{\pi}(s)$ is the advantage function, where $Q_{\pi}(s_t, a_t)$ is the action-value function, and $V_{\pi}(s)$ is the value function. $Q_{\pi}(s_t, a_t) - V_{\pi}(s)$ represents the advantage of the action-value function over the current value function.

By using the all-state rewriting equation 10, we obtain:

$$\eta(\pi') = \eta(\pi) + \sum_s \rho_{\pi'}(s) \sum_a \pi'(a|s) A_{\pi}(s, a) \quad (11)$$

Here, ρ is the discount visitation frequency. $\rho_{\pi'}(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \dots$. To ensure that the policy is monotonically non-decreasing, we need to ensure that $\sum_a \pi'(a|s) A_{\pi}(s, a)$ is non-negative. We introduce a local approximation as:

$$L(\pi') = \eta(\pi) + \sum_s \rho_{\pi}(s) \sum_a \pi'(a|s) A_{\pi}(s, a) \quad (12)$$

In the use of visitation frequency ρ_{π} in $L(\pi')$, the change in visitation frequency due to a change in policy is ignored, and the approximation error of the advantage function is also neglected. When using conservative policy iteration, the new policy $\pi_{new}(a|s)$ is determined by the following equation:

$$\pi_{new}(a|s) = (1 - \alpha) \pi_{old}(a|s) + \alpha \pi'(a|s) \quad (13)$$

Here, π_{new} represents the new policy and π_{old} represents the old policy. Furthermore, $\pi' = \argmax_{\pi'} L_{\pi_{old}}(\pi')$. Thus, we have:

$$\eta(\pi') \geq L_{\pi}(\pi') - C D_{KL}^{max}(\pi, \pi') \quad (14)$$

Here, C is the penalty coefficient and D_{KL}^{max} represents the KL divergence between the new and old policies. To ensure that the expected long-term reward η monotonically increases, we only need to maximize the right-hand side of Equation 14. Introducing a parameter θ to replace the policy π , with θ_{old} representing the policy to be improved, we have:

$$\text{maximize}_{\theta} [L_{\theta_{old}}(\theta) - C D_{KL}^{max}(\theta_{old}, \theta)] \quad (15)$$

If the penalty coefficient C is too large, the step size will be too small, which affects the update speed. Therefore, a constraint can be added to the KL divergence:

$$\begin{aligned} & \text{maximize}_{\theta} \quad L_{\theta_{old}}(\theta) \\ & \text{such that} \quad D_{KL}^{max}(\theta_{old}, \theta) \leq \delta \end{aligned} \quad (16)$$

Introducing the average KL divergence to simplify D_{KL}^{max} :

$$\begin{aligned} & \text{maximize}_{\theta} L_{\theta_{old}}(\theta) \\ & \text{such that } D_{KL}^{\theta_{old}}(\theta_{old}, \theta) \leq \delta \end{aligned} \quad (17)$$

Expanding L yields the objective function expressed as:

$$\text{maximize}_{\theta} E_s \pi_{\theta_{old}} a \pi_{\theta_{old}} \left[\frac{\pi_{\theta}(a|s)}{\pi_{\theta_{old}}(a|s)} \right] A_{\pi_{\theta_{old}}}(s, a)$$

such that

$$E_s \pi_{\theta_{old}} [D_{KL}(\pi_{\theta_{old}}(\cdot|s) | \pi_{\theta_{old}}(\pi_{\theta_{old}}(\cdot|s)))] \leq \delta \quad (18)$$

The pseudocode of the TRPO algorithm can be represented as follows:

Algorithm 2 Trust Region Policy Optimization (TRPO)

```

1: Initialize policy parameters  $\theta_{old}$ 
2: for each iteration do
3:   Collect trajectories  $\tau_i$  using policy  $\pi_{\theta_{old}}$ 
4:   Compute the rewards  $R(\tau_i)$  and advantage estimates  $A_i(\theta_{old})$  for each trajectory  $\tau_i$ 
5:   Solve the constrained optimization problem in Equation (18) to obtain  $\theta$ :
       maximize  $\theta L(\theta_{old}, \theta)$ 
       subject to  $E_s \pi_{\theta_{old}} [D_{KL}(\pi_{\theta_{old}}(\cdot|s) || \pi_{\theta}(\cdot|s))] \leq \delta$ 
6:   Set  $\theta_{old} = \theta$ 
7: end for
    
```

Fig. 2: Pseudocode of the TRPG algorithm.

The TRPO algorithm is a constrained optimization problem in which the average KL divergence between the new and old policies must be smaller than δ . However, the TRPO algorithm requires significant computational effort to perform constrained optimization using conjugate gradient calculations. In response, the OpenAI team proposed an improved method called Proximal Policy Optimization (PPO) algorithm.

In the PPO algorithm, the constraint is replaced with a penalty term, avoiding the need for conjugate gradient calculations. The new objective function can be expressed as follows:

$$L^{CPI}(\theta) = \widehat{E}_t[r_t(\theta)\widehat{A}_t] \quad (19)$$

where L^{CPI} represents conservative policy iteration, \widehat{A}_t is the advantage function, and $r_t(\theta) = \frac{\pi_{\theta}(a|s)}{\pi_{\theta_{old}}(a|s)}$ is the probability ratio between the new and old policies.

Maximizing L results in significant policy updates, which can increase variance. Therefore, a penalty is introduced to penalize larger policy updates. The modified objective function is:

$$L^{CPI}(\theta) = \widehat{E}_t[\min r_t(\theta)\widehat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\widehat{A}_t] \quad (20)$$

The penalty term clips $r_t(\theta)$ within the interval $[1 - \epsilon, 1 + \epsilon]$ to reduce its impact on the objective function.

The pseudocode of the PPO algorithm can be represented as follows:

Algorithm 3 Proximal Policy Optimization (PPO)

```

1: Initialize policy parameters  $\theta_{old}$ 
2: Initialize the value function parameters  $\phi$ 
3: Define the clipping parameter  $\epsilon$ 
4: for each iteration do
5:   Collect trajectories  $\tau_i$  using policy  $\pi_{\theta_{old}}$ 
6:   Compute the rewards  $R(\tau_i)$  and advantage estimates  $A_i$  for each trajectory  $\tau_i$ 
7:   Update the value function parameters  $\phi$  by minimizing the value function loss
8:   for each epoch do
9:     for each batch of states, actions, and advantages in the collected trajectories do
10:      Compute the probability ratio  $r_t(\theta)$  using Equation (19):  $r_t(\theta) = \frac{\pi_{\theta}(a|s)}{\pi_{\theta_{old}}(a|s)}$ 
11:      Compute the clipped probability ratio:  $\text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)$ 
12:      Compute the PPO objective:  $L^{CPI}(\theta) = E_t[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)A_t)]$  (Equation (20))
13:      Update policy parameters  $\theta$  by maximizing the PPO objective:  $\theta = \theta + \alpha \nabla_{\theta} L^{CPI}(\theta)$ 
14:    end for
15:  end for
16:  Set  $\theta_{old} = \theta$ 
17: end for
    
```

Fig. 3: Pseudocode of the PPO algorithm.

3.2 Control target

The purpose of this paper is to implement vehicle control based on the PPO algorithm, which sets a goal for the vehicle to reach using control commands generated by the PPO algorithm. The paper uses a laser radar to scan the surrounding environment of the vehicle and obtain its relative position in the environment. Based on the collected position information, the PPO algorithm is used to control the distance d' between the vehicle and the surrounding objects, that is, to automatically approach the object when the distance between the vehicle and the object exceeds a certain threshold d, and to automatically move away from the object when the distance is less than the threshold d, as shown in Fig. 4.

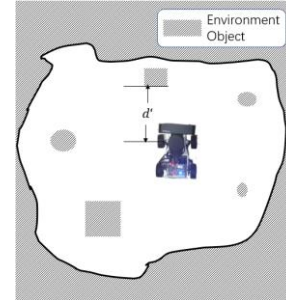


Fig. 4: Control target.

The PPO algorithm controls the movement of the vehicle (away from or closer to the object) based on the collected position information. The environment information obtained consists of information about the points on the environment objects. An environment object is made up of many points, and the distance information in different points' information of the same object does not vary greatly. However, due to the width of the environment object, there can be significant differences in the angle information of different points' information. If the relative angle between the vehicle and the environment object is required, it can cause serious interference to the PPO algorithm. Therefore, when maintaining a certain distance between the vehicle and the environment object, no requirements are made on the angle relationship between the vehicle and the environment object. Only the distance information of the environment object is used to calculate rewards to train the algorithm, and the angle information is only used to

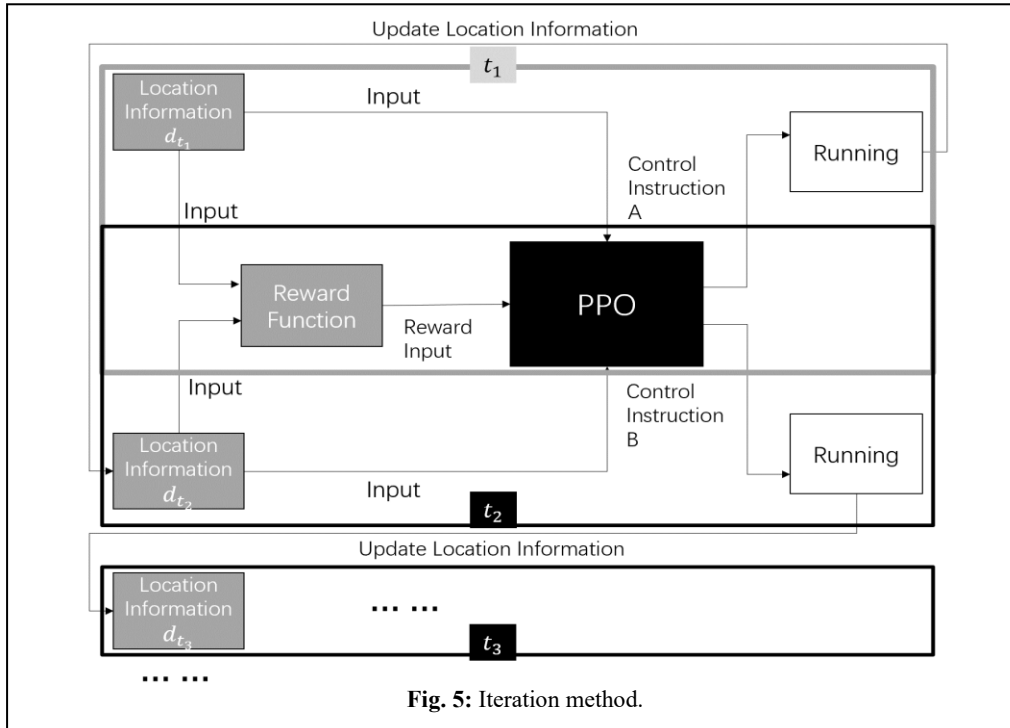
help the algorithm determine the approximate angle position of the current environment object and to assist the algorithm in selecting a direction of motion.

3.3 Training method

The iteration method of the PPO algorithm based on the collected environment information is shown in Fig. 5. At time t_1 , the laser radar obtains the distance information of the environment object, d_{t_1} , which is input into the PPO algorithm to obtain control command A, and the vehicle runs while saving d_{t_1} to the reward function. After the vehicle runs, the laser radar obtains the distance information d_{t_2} , which is input into the reward function to calculate the reward obtained by the control command A issued by the PPO algorithm at time t_1 . The obtained reward is used to evaluate the control effect, and the evaluation result is input into the PPO algorithm. The algorithm will improve and optimize according to the evaluation result and issue a new control command B based on d_{t_2} , which makes the vehicle run again. This completes one training process of the PPO algorithm.

sensitive to the position information scanned by the radar at the current and previous moments. If there is any human interference during the vehicle's operation, it will affect the convergence speed of the algorithm and may even prevent the algorithm from converging, making it impossible to complete the algorithm training.

In this paper, the aforementioned method will be employed to train the algorithm in the environment depicted in Fig. 6. The rationale for training in a real-world environment is to more accurately reflect the challenges and uncertainties present in the real world. All objects in the figure, except for the vehicle, are considered environmental objects. During the training process, the vehicle is allowed to explore the environment freely without any interference, which contributes to the development of better generalization capabilities and robustness for the algorithm when faced with complex environments and uncertainties in the real world.



By continuously running the vehicle and training the PPO algorithm, when the re-ward calculated based on the reward function converges, it is considered that the PPO algorithm has completed its training.

During actual algorithm training, the vehicle is randomly placed in any position in the environment as the start of training. Once the training begins, the vehicle's movement will not be interfered with in any way, allowing it to run completely autonomously. This is because the reward calculated by the reward function and the instructions issued by the PPO algorithm are extremely



Fig. 6: Training environment

4. Experiment and data processing

4.1 Device

The main computing platform used in this study is the Jetson Nano, a small GPU computing platform developed by NVIDIA specifically for robots. Its main parameters are presented in Table 1.

Table 1. Parameters of Jetson Nano.

Item	Parameters
CPU	4 Core ARM A57 @1.43GHz
GPU	128 Core Maxwell 472 GFLOPs (FP16)
Memory	4 GB 64 bit LPR4 25.6GB/s
Storage	16 GB eMMC

movement of the vehicle.

Table 2. Parameters of RPLIDAR A1.

Item	Parameters
Measurement range	0.15m-12m
Scanning angle	0° -360°
Measurement resolution	<0.5mm
Angular resolution	<=1°
Single measurement time	0.5ms
Measurement frequency	>=4000Hz
Scanning frequency	5.5Hz

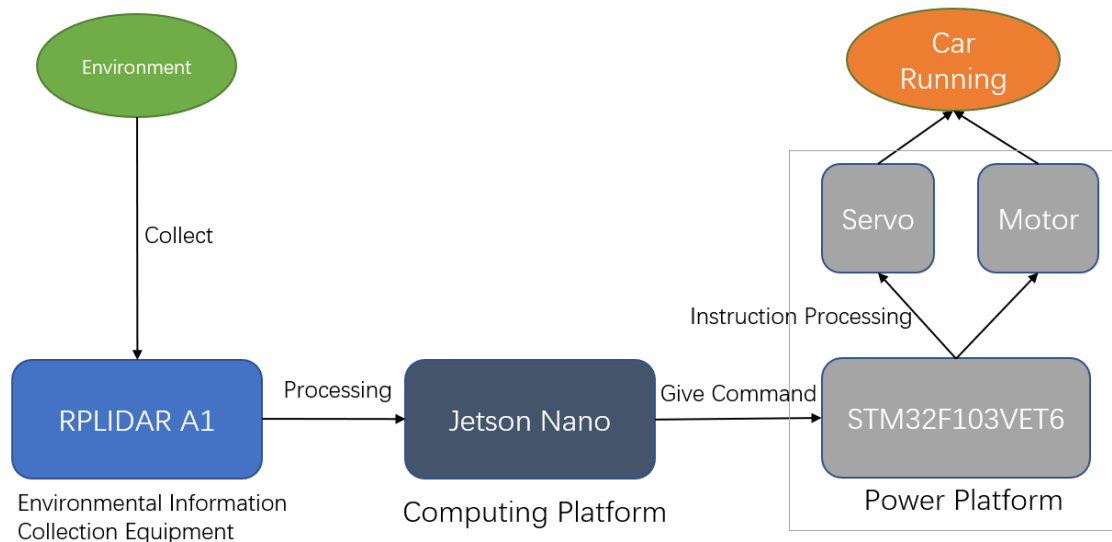


Fig. 7: Devices.

The RPLIDAR A1 LIDAR sensor, responsible for collecting environmental information, has its main parameters presented in Table 2. The vehicle's main power is provided by two DC motors, while its steering function is provided by a servo motor. The DC motors and servo motor are directly controlled by an STM32 microcontroller (STM32F103VET6), which together form the power platform.

The relationship between the environmental information collection device, the main computing platform, and the power platform is illustrated in Fig. 7. After the 1 LIDAR sensor collects environmental information, the information is sent to the computing platform for processing. The computing platform then sends instructions to the power platform's control core, which further processes the received instructions to make the servo and motors run, ultimately resulting in the

4.2 Method of processing environmental information.

The information collection method of the LiDAR is shown in Fig. 8. During each scan of the surrounding environment, the LiDAR simultaneously scans the positions of environmental objects at 360 angles. Each position information consists of distance data describing the distance between the environmental object and the vehicle, and angle data describing the angular position of the object relative to the vehicle. If the data is not processed, the PPO algorithm processes at least 720 pieces of data in a single iteration, significantly increasing the algorithm's computational complexity and training time. To reduce the computational complexity of the PPO algorithm and accelerate training speed, the in-formation

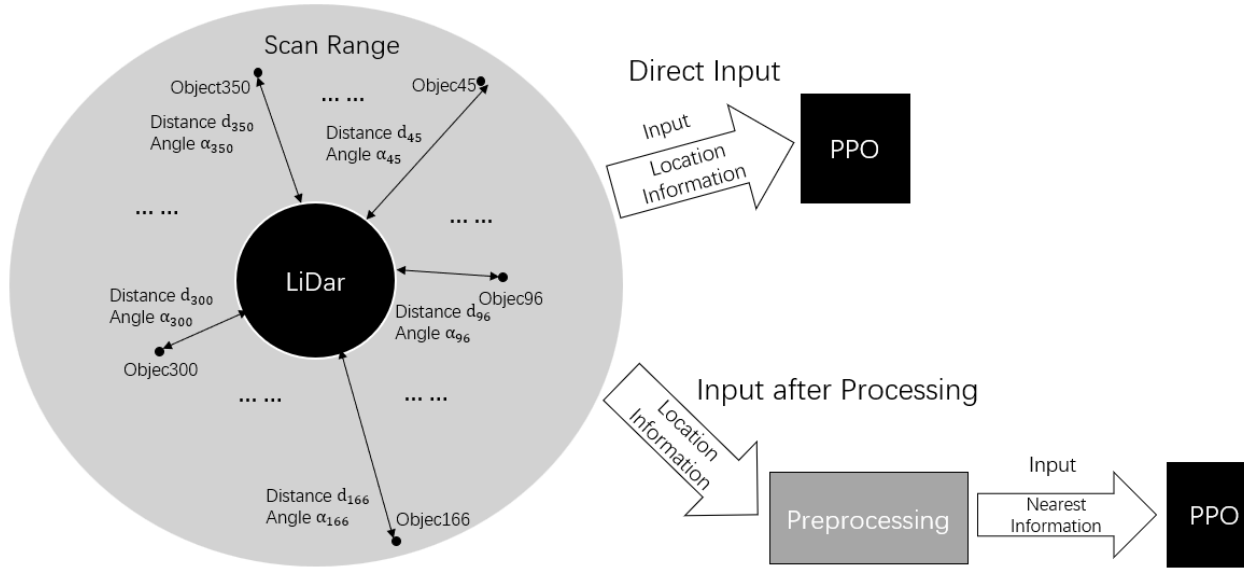


Fig. 8: Environmental information collection method.

collected by the LiDAR needs to be preprocessed. Under the control objective, only the distance to the closest object in the environment and its corresponding angle need to be obtained. After filtering the distance information, only the position information of the nearest object is input into the PPO algorithm.

4.3 Algorithm Settings

The action space refers to the angular velocity and linear velocity of vehicle movement. The advantage of the PPO algorithm lies in its support for continuous output actions. Therefore, the action space is set within a certain range, taking into account both radar detection and algorithmic calculation delays. The vehicle's movement speed should not be too fast. Therefore, the range of linear velocity for vehicle movement is $[-0.2\text{m/s}, 0.2\text{m/s}]$, and the range of angular velocity is $[-0.2\text{rad/s}, 0.2\text{rad/s}]$. The ranges of linear velocity and angular velocity constitute the action space of the PPO algorithm.

When the action command is a linear velocity of 0.2m/s and an angular velocity of 0.2rad/s , the vehicle will move as shown in Fig. 9.

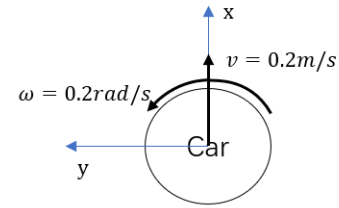


Fig. 9: Diagram of vehicle movement.

The control target is to maintain a certain distance, d , between the vehicle and environmental objects. In this chapter, d is set to 0.3m . The algorithm's control commands are evaluated based on the reward value computed by determining whether the distance between the vehicle and environmental objects at each time step is closer to d . The maximum reward is obtained when the current distance, d' , between the vehicle's current position and the environmental object is 0.3m .

Table 3 shows the reward function settings. Due to the measurement error of the lidar, the optimal reward range is limited to between 0.3m and 0.34m . The maximum reward is obtained at 0.3m - 0.32m , and a fault tolerance interval is set between 0.32m - 0.34m . This setting makes

Table 3. Reward function.

d	Movement Reward (r_d)	Time Punishment (r_t)	Total Reward
$0.3\text{m} < d' \leq 0.32\text{m}$	10	-	10
$0.2\text{m} < d' < 0.3\text{m}$	$(d' - d'') \times 20$	$\frac{0.2 \times (d' - d'')}{\text{abs}(t' - t'')}$	$r_d + r_t$
$0.34\text{m} < d'$	$-(d' - d'') \times 20$	$\frac{-0.2 \times (d' - d'')}{\text{abs}(t' - t'')}$	$r_d + r_t$
$d' < 0.2\text{m}$	$-0.9 \cdot (d' - d'') \times 5$	-	r_d

the rewards obtained by the vehicle clearer. d' denotes the distance between the vehicle and environmental target at the current step, while d'' denotes the distance between the vehicle and environmental target at the previous step. t' denotes the absolute time of the system at the current step, while t'' denotes the absolute time of the system at the previous step.

In this task, it is desired to encourage the algorithm to take more favorable actions by setting the maximum reward value to be greater than the penalty value (negative reward). Therefore, the maximum reward value is set to 10, while the maximum penalty value is around -1. Moreover, due to the algorithm's running speed and the vehicle's linear velocity, $abs(d' - d'')$ is usually less than 0.1. Therefore, different coefficients are used in different situations to amplify $(d' - d'')$ and balance its weight in the reward calculation process.

To avoid collisions between the vehicle and environmental objects, the minimum distance between the vehicle and environmental objects is set to 0.2m. If the algorithm's control command causes the vehicle to move to a position where d' is less than the minimum distance, a very large penalty will be given, regardless of how the vehicle moves. If d' is greater than d_{min} and has not yet reached the optimal distance d , the movement reward rd will be given based on $d' - d''$ (where d'' is the distance between the vehicle and the environmental object in the previous control step) to judge whether d' is approaching the control target. Additionally, to promote algorithm exploration and prevent the algorithm from falling into local optima, a time penalty r_t is added. The reward function calculates the penalty based on the distance traveled and the time between steps.

What's more, due to measurement errors or control commands that cause the vehicle to almost not move, there may be situations where $r_d = 0$. Therefore, an additional term is added to give a penalty once $r_d = 0$, to encourage the algorithm to move the vehicle.

The episodic approach is adopted to accelerate the convergence speed of the algorithm. The reward obtained within an episode and the total return will be saved, and when the vehicle's running meets certain conditions, it is considered as completing an episode. After completing multiple episodes, the reward data within these episodes will be used to iteratively update the algorithm. The specific process is shown in Fig. 10. When the distance between the vehicle and environmental objects is exactly d , or when the number of instructions (i.e., steps) issued by PPO reaches a certain value n , regardless of the vehicle's position, it is considered as completing an episode.

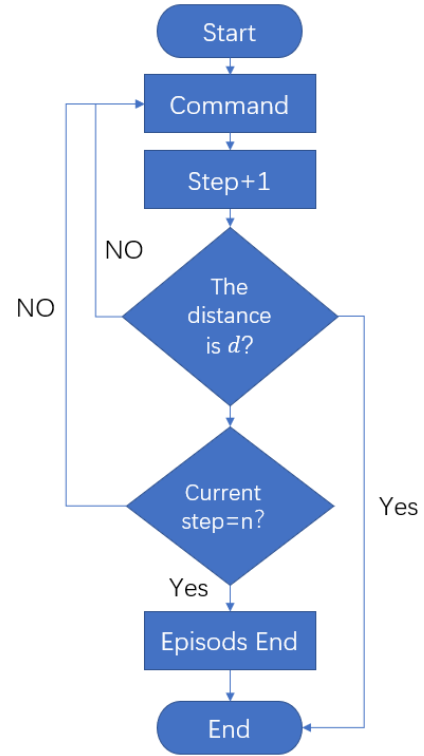


Fig. 10: Illustration of the algorithm updating process

Other parameter settings are shown in Table 4. The maximum number of steps per episode is set to 50 steps, and the algorithm will be iteratively updated once every 1000 steps, which is one round of updates.

Table 4. Parameter Settings.

Item	Meaning	Value
lr	Learning Rate	0.0003
Betas	Optimizer Hyperparameters	(0.9, 0.99)
Gamma	Reward Discount Factor	0.99
K_epochs	-----	50
update_timestep	-----	1000
max_timesteps	-----	50
eps_clip	Clipping Parameters	0.2

5. Result and discussion

5.1 Training Result

Figure 11 shows the change in total reward per episode after 400 episodes, during which the algorithm experienced approximately 14 iterations of updates based on the algorithm parameters. The gray line represents the real data of the average reward per episode, while the black line represents the result after exponential smoothing of the real data. It can be seen that after about

200 episodes, the average reward per episode was always at the maximum value of the reward function. This indicates that the algorithm has converged, and at this point, the algorithm has experienced approximately four iterations of updates.

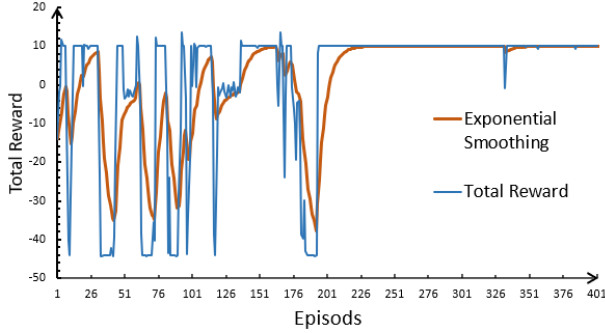
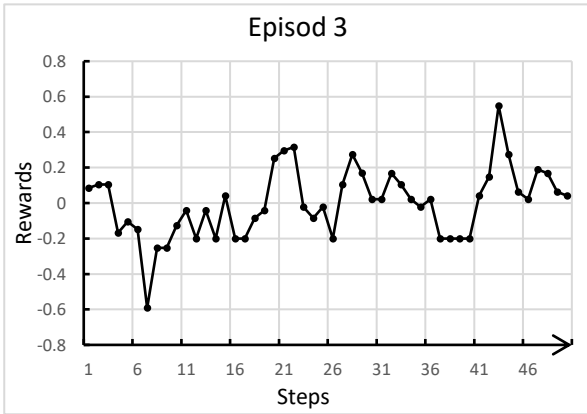
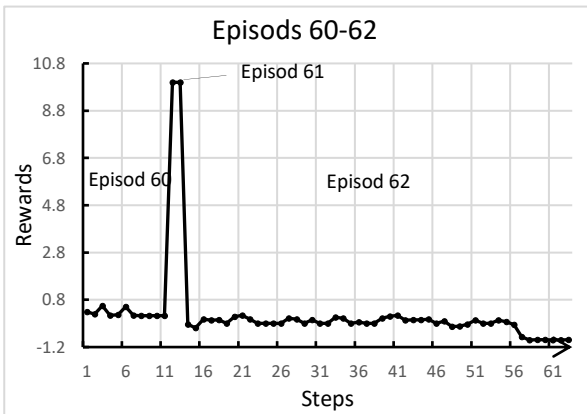


Fig. 11: Training result.

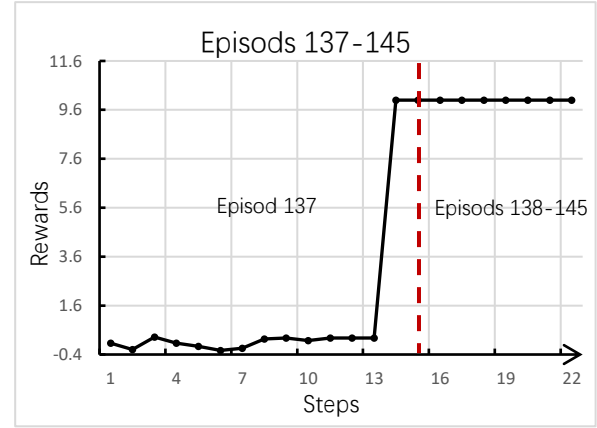
To further illustrate the algorithm training process, we observe the reward obtained by the algorithm for each step of the episode 3, episodes 60-62, and episodes 137-145, which represent the beginning, middle, and late stages of the training process, respectively, as shown in Fig. 12.



(a)



(b)



(c)

Fig. 12: Rewards of rounds: (a) Rewards for each step of the episode 3; (b) Rewards for each step of episodes 60-62; (c) Rewards for each step of episodes 137-145

It can be seen that in the episode 3, which represents the beginning of the training stage, the algorithm was only exploring, and the reward value was fluctuating. In the middle stage of the training process (episodes 60-62), the algorithm quickly reached the maximum reward in episode 60, indicating that the algorithm had gradually learned how to achieve the control goal through different instructions. However, after maintaining the goal for one episode (episode 61), the algorithm achieved the goal within 50 steps of episode 62, indicating that the algorithm had not yet completed its training. In the late stage of the algorithm training (episodes 137-145), it can be seen that the algorithm can achieve the goal with a small number of steps in a single episode, and can continue to maintain the goal over multiple episodes. This indicates that the algorithm has mastered the relationship between different instructions and control objectives, and the algorithm is close to convergence.

5.2 Validation and discussion

The Figure 13 shows the trajectory of the trained model when the environmental objects are approaching and moving away from the vehicle, indicating that the vehicle can achieve the control objectives set under the control of the PPO algorithm. Moreover, the vehicle can also achieve the control objectives in complex environments, as shown where the vehicle is stuck between two obstacles, and the vehicle must turn to get out of the situation. Note that due to the limitations of the environmental information in the following figure. Figure 14(a) shows a scenario where a large obstacle exists on the right side of the vehicle and a small obstacle is present in front of it. This means that no matter whether the vehicle moves forward or backward, the processed environmental information always tells the algorithm that the vehicle is in a state of being close to the environmental object. The escape method can only move the vehicle to the left and away from the obstacles or move it to the left and forward to bypass the obstacle in front.

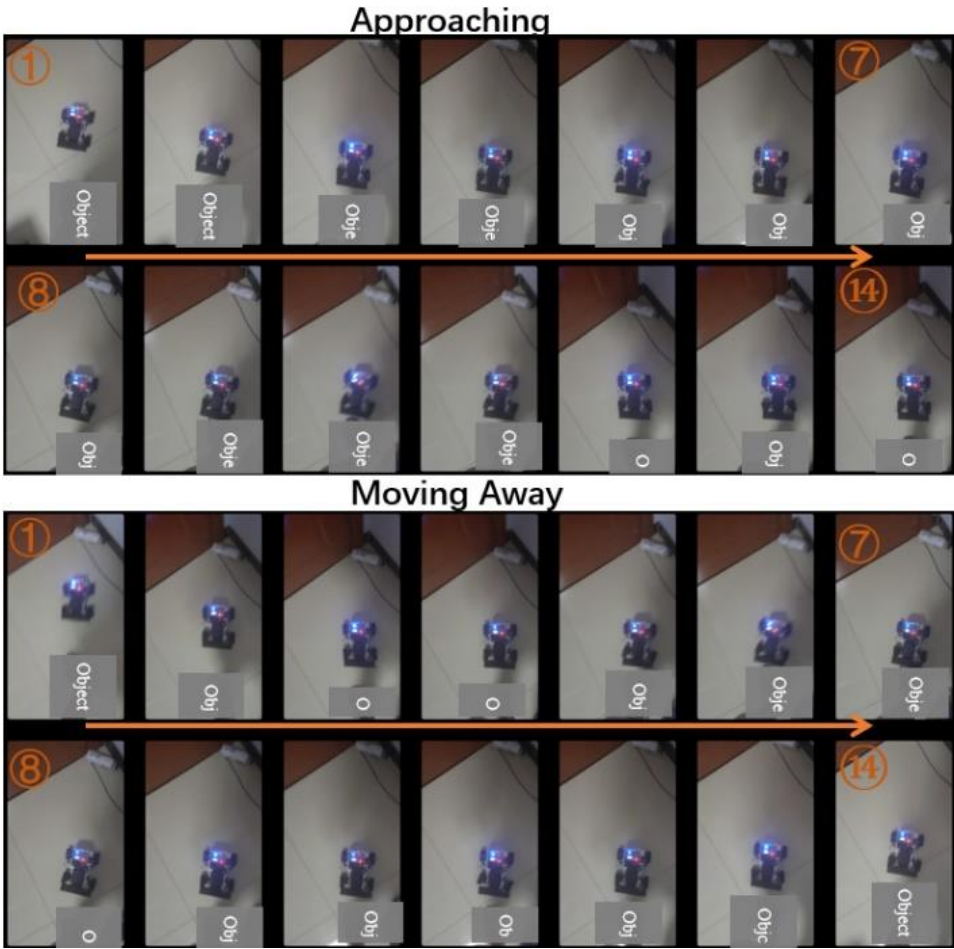


Fig. 13: Vehicle's motions.

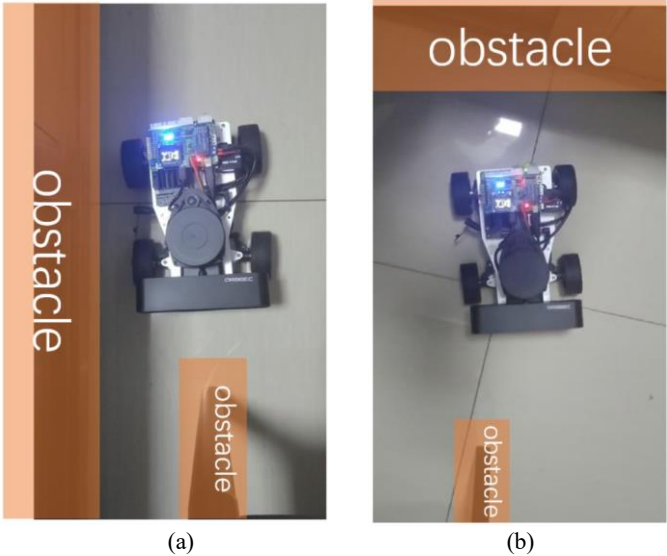
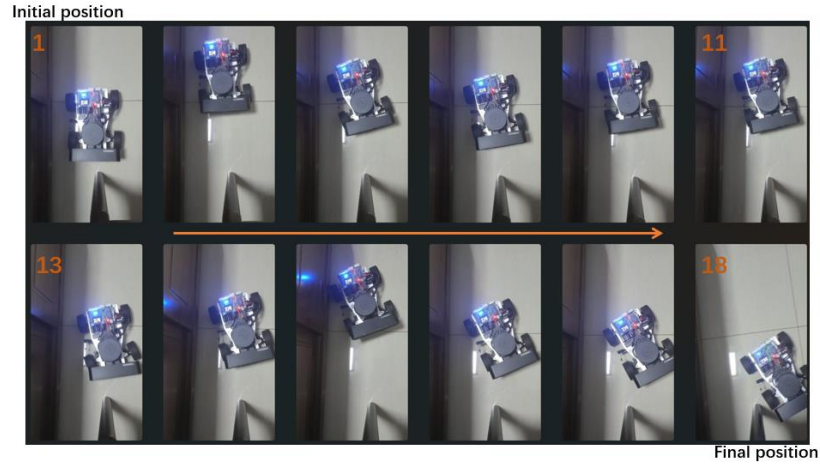
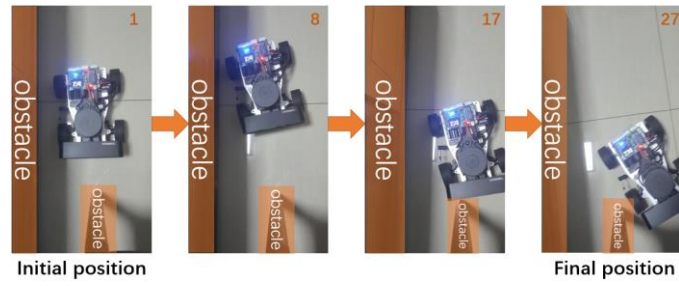


Fig. 14: Different environmental: (a) A large obstacle on the right side of the vehicle and a small obstacle in front; (b) Vehicle stuck between two obstacles.

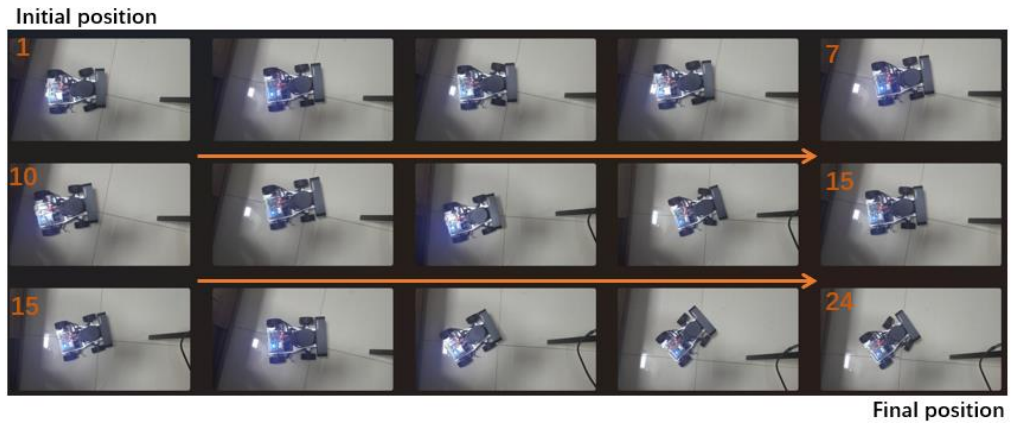


(a)

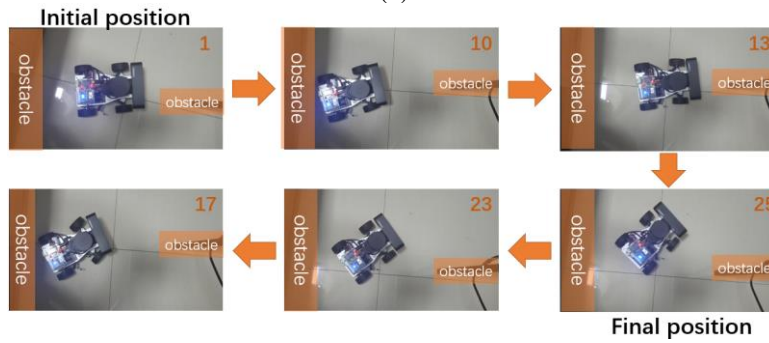


(b)

Fig. 15: Vehicle's motions under Fig. 14(a): (a) Vehicle's motions; (b) Selected vehicle motions.



(a)



(b)

Fig. 16: Vehicle's motion under Fig. 14(b): (a) Vehicle's motions; (b) Selected vehicle motions.

Figure 14(b) shows a scenario where the vehicle is stuck between two obstacles, and the vehicle must turn to get out of the situation. Note that due to the limitations of the environmental information acquisition method, the closest to the vehicle at each time, and cannot obtain the overall environmental information.

Figures 15 and 16 respectively illustrate the vehicle's motions under the circumstances depicted in Fig. 14 (a) and Fig. 14 (b). It is evident that the vehicle does not overcome challenging situations through a one-off movement process. Instead, it achieves its goal only after a certain amount of back-and-forth movement. The number of these oscillatory movements varies depending on the environment.

This is because, to simplify computation, certain treatments and definitions were applied to the environmental information in this study. Only the coordinates closest to the vehicle are input into the PPO algorithm. Therefore, the PPO algorithm cannot fully obtain the surrounding environmental information of the vehicle. Consequently, in the initial stage of operation, the PPO algorithm does not know how to resolve such challenging situations. However, after performing a series of actions, the PPO algorithm continuously evaluates the entire policy under the current situation and makes adjustments, ultimately selecting the most beneficial policy. For instance, it may choose to move left and forward in the case of challenging situation (a). This policy evaluation partially compensates for the algorithm's inability to obtain a comprehensive view of the environmental information, thereby enabling the algorithm to control the vehicle to undertake more complex tasks.

6. Conclusions

In this study, the PPO algorithm was directly trained in a real-world environment, aiming to avoid the limitations of simulating real environments in virtual settings. In the course of actual training, the algorithm is required to tackle a variety of challenges including sensor precision, execution of the algorithm, and latency in control commands. Nevertheless, the experimental results indicate that the PPO algorithm is capable of effective training and demonstrates rapid convergence. This suggests that the PPO algorithm does not require strict environmental conditions and input data during the training process, and can tolerate a certain degree of uncertainty, noise, and interference in the training environment.

When trained in a real-world environment, the PPO algorithm can enable the vehicle to achieve control objectives effectively. Notably, in the application environment of this study, the vehicle can only obtain the nearest environmental information at each time step. Nevertheless, the PPO algorithm can still generate optimal policies to achieve goals when dealing with challenging situations. This indicates that the PPO algorithm has good

robustness and generalization capabilities.

In summary, the PPO algorithm demonstrates strong robustness during the training process, as well as good generalization and robustness in practical applications for vehicle motion control tasks. Its rapid convergence, adaptability, and ability to achieve control objectives in the face of training environment uncertainty, noise, and interference highlight the potential of the PPO algorithm for widespread practical applications. Future work may consider extending the algorithm to more complex environments or incorporating more input information, such as depth maps or vehicle motion state data, for training.

References

- 1) C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving," 2015. <http://deepdriving.cs.princeton.edu>.
- 2) ZHANG Xinyu, GAO Hongbo, and ZHAO Jianhui, "Overview of deep learning intelligent driving methods," *Journal of Tsinghua University (Science and Technology)*, 58(4) 438–444 (2018).
- 3) M. Ayundyahrini, Danar Agus Susanto, H. Febriansyah, Fariz Maulana Rizanulhaq, and Gama Hafizh Aditya, "Smart farming: integrated solar water pumping irrigation system in thailand," *Evergreen*, 10 (1) 553–563 (2023). doi:10.5109/6782161.
- 4) K. Sujatha, N.P.G. Bhavani, V.S. George, T.K. Reddy, N. Kanya, and A. Ganesan, "Innovation in agriculture industry by automated sorting of rice grains," *Evergreen*, 10 (1) 283–288 (2023). doi:10.5109/6781076.
- 5) A. Arunika, J.F. Fatriansyah, and V.A. Ramadheena, "Detection of asphalt pavement segregation using machine learning linear and quadratic discriminant analyses," *Evergreen*, 9 (1) 213–218 (2022). doi:10.5109/4774236.
- 6) H.K. Chaudhary, K. Saraswat, H. Yadav, H. Puri, A.R. Mishra, and S.S. Chauhan, "A real time dynamic approach for management of vehicle generated traffic," *Evergreen*, 10(1) 289–299 (2023). doi.org/10.5109/6781078
- 7) P. Panwar, P. Roshan, R. Singh, M. Rai, A.R. Mishra, and S.S. Chauhan, "DDNet- a deep learning approach to detect driver distraction and drowsiness," *Evergreen*, 9 (3) 881–892 (2022). doi:10.5109/4843120.
- 8) V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, 518 (7540) 529–533 (2015). doi:10.1038/nature14236.

- 9) D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, 529 (7587) 484–489 (2016). doi:10.1038/nature16961.
- 10) K. Arulkumaran, A. Cully, and J. Togelius, "Alphastar: An evolutionary computation perspective," in: *GECCO 2019 Companion - Proceedings of the 2019 Genetic and Evolutionary Computation Conference Companion*, Association for Computing Machinery, Inc, 2019: pp. 314–315. doi:10.1145/3319619.3321894.
- 11) C. You, J. Lu, D. Filev, and P. Tsiotras, "Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning," *Rob Auton Syst*, 114 1–18 (2019). doi:10.1016/j.robot.2019.01.003.
- 12) J. Xin, H. Zhao, D. Liu, and M. Li, "Application of deep reinforcement learning in mobile robot path plan-ning," in: *Proceedings - 2017 Chinese Automation Congress, CAC 2017*, Institute of Electrical and Elec-tronics Engineers Inc., 2017: pp. 7112–7116. doi:10.1109/CAC.2017.8244061.
- 13) S. Aradi, "Survey of deep reinforcement learning for motion planning of autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, 23 (2) 740–759 (2022). doi:10.1109/TITS.2020.3024655.
- 14) M. Zhu, Y. Wang, Z. Pu, J. Hu, X. Wang, and R. Ke, "Safe, efficient, and comfortable velocity control based on reinforcement learning for autonomous driving," *Transp Res Part C Emerg Technol*, 117 (2020). doi:10.1016/j.trc.2020.102662.
- 15) G. Chen, Y. Lu, X. Yang, and H. Hu, "Reinforcement learning control for the swimming motions of a beaver-like, single-legged robot based on biological inspiration," *Rob Auton Syst*, 154 (2022). doi:10.1016/j.robot.2022.104116.
- 16) T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J.A. Ojea, E. Solowjow, and S. Levine, "Residual reinforcement learning for robot control," in: *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019: pp. 6023–6029.
- 17) J. Chen, B. Yuan, and M. Tomizuka, "Model-free deep reinforcement learning for urban autonomous driving," (2019). <http://arxiv.org/abs/1904.09503>.
- 18) J. Chen, S.E. Li, and M. Tomizuka, "Interpretable end-to-end urban autonomous driving with latent deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, 23 (6) 5068–5078 (2022). doi:10.1109/TITS.2020.3046646.
- 19) R. Liu, F. Nageotte, P. Zanne, M. de Mathelin, and B. Drespe-Langley, "Deep reinforcement learning for the control of robotic manipulation: a focussed mini-review," *Robotics*, 10 (1) 1–13 (2021). doi:10.3390/robotics10010022.
- 20) W. Zhao, J.P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," (2020). doi:10.1109/SSCI47803.2020.9308468.
- 21) B.R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A.A.A. Sallab, S. Yogamani, and P. Perez, "Deep reinforcement learning for autonomous driving: a survey," *IEEE Transactions on Intelligent Transportation Systems*, 23 (6) 4909–4926 (2022). doi:10.1109/TITS.2021.3054625.
- 22) J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," (2017). <http://arxiv.org/abs/1707.06347>.
- 23) P. Wolf, C. Hubschneider, M. Weber, A. Bauer, J. Hartl, F. Durr, and J.M. Zollner, "Learning how to drive in a real world simulation with deep Q-Networks," in: *IEEE Intelligent Vehicles Symposium, Proceedings*, Institute of Electrical and Electronics Engineers Inc., 2017: pp. 244–250. doi:10.1109/IVS.2017.7995727.
- 24) T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," (2015). <http://arxiv.org/abs/1509.02971>.
- 25) A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learn-ing to drive in a day," in: *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019: pp. 8248–8254.
- 26) M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, "Explaining how a deep neural network trained with end-to-end learning steers a car," (2017). <http://arxiv.org/abs/1704.07911>.
- 27) J. Kim, and J. Canny, "Interpretable Learning for Self-Driving Cars by Visualizing Causal Attention," 2017.
- 28) J. Chen, T. Wu, M. Shi, and W. Jiang, "PORF-ddpg: learning personalized autonomous driving behavior with progressively optimized reward function," *Sensors (Switzerland)*, 20 (19) 1–19 (2020). doi:10.3390/s20195626.
- 29) X. Liang, T. Wang, L. Yang, and E. Xing, "CIRL: Controllable Imitative Reinforcement Learning for Vi-sion-based Self-driving," n.d.