IAES International Journal of Artificial Intelligence (IJ-AI)
Vol. 12, No. 1, March 2023, pp. 422~431
ISSN: 2252-8938, DOI: 10.11591/ijai.v12.i1.pp422-431

Autonomous driving system using proximal policy optimization in deep reinforcement learning

Imam Noerhenda Yazid, Ema Rachmawati
School of Computing, Telkom University, Bandung, Indonesia

Article Info

Article history:

Received Mar 1, 2022
Revised Aug 26, 2022
Accepted Sep 24, 2022

Keywords:

Agent
Autonomous driving
Deep reinforcement learning
Proximal policy optimization
Unity3D

ABSTRACT

Autonomous driving is one solution that can minimize and even prevent accidents. In autonomous driving, the vehicle must know the surrounding environment and move under the provisions and situations. We build an autonomous driving system using proximal policy optimization (PPO) in deep reinforcement learning, with PPO acting as an instinct for the agent to choose an action. The instinct will be updated continuously until the agent reaches the destination from the initial point. We use five sensory inputs for the agent to accelerate, turn the steer, hit the brakes, avoid the walls, detect the initial point, and reach the destination point. We evaluated our proposed autonomous driving system in a simulation environment with several branching tracks, reflecting a real-world setting. For our driving simulation purpose in this research, we use the Unity3D engine to construct the dataset (in the form of a road track) and the agent model (in the form of a car). Our experimental results firmly indicate our agent can successfully control a vehicle to navigate to the destination point.

Corresponding Author:

Ema Rachmawati
School of Computing, Telkom University
Jl. Telekomunikasi, Bandung, Indonesia
Email: emarachmawati@telkomuniversity.ac.id

1. INTRODUCTION

Nowadays, technology is developing so fast, one of them being in the transportation field. Everyone wanted to do something instantly. Even though people are already supported by high technology, a chance for a horrible incident might still happen because of human negligence factors [1]. In the case of driving a car, with a lot of technology implemented in a vehicle, there are still many car accidents in this world. According to this condition, many researchers have researched to solve the vehicle accident problem. One is autonomous driving [1], a car that can move without a driver. Not just moving, the car must avoid an accident in the environment. The vehicle is forced to know about its environment and move based on some rules and situations. In fact, collecting and analyzing information related to driver behavior while driving is essential as an input for the automation system connected to the agent learning process [2].

The rapid advances in autonomous driving technology are mainly supported by advances in the area of deep learning and artificial intelligence, especially the increasing use of convolutional neural network (CNN) dan deep reinforcement learning (DRL) [3]. Some researchers integrated CNN in their proposed autonomous driving and related system [4]–[9] for processing spatial information and can be viewed as image feature extractors. Tian *et al*. [4] conducted the autonomous driving experiment using a deep neural network (DNN) by incorporating several inputs such as steering angle, brake, acceleration, light detection and ranging (LIDAR) input, and infrared (IR) sensor input; thus, there will be many variances of the possibility observed. Li *et al*. [5] proposed a human-like driving system to give autonomous vehicles the ability to make decisions like humans using CNN to detect, recognize and abstract the information in the input road scene captured by the onboard sensors. Instead of using color images, in terms of red-green-blue (RGB) images, which is unstable as they claimed, they used only depth information in their system. Wu *et al.* [6] proposed convolutional layers to extract feature maps and compute bounding boxes and class probabilities as the output layer. They claimed that their model leads to a small model size and better energy efficiency as these constraints were needed in the real-time inference of autonomous driving. Navarro *et al.* [7] also proposed several end-to-end DNN architectures to generate vehicle speed and steering wheel angle control actions. Teti *et al.* [8] studied using several CNN architectures on an end-to-end steering system. They showed their findings by reporting each network's lap completion rate and path smoothness. Also, Dai and Lee [9] used fully CNN to detect curve lanes in the real-time lane detection for their autonomous driving simulation system

Meanwhile, the use of DRL is equally essential, along with the increasing complexity of learning policies in high-dimensional environments [10], [11]. DRL cannot be separated from reinforcement learning (RL), where RL is a promising solution, especially in driving policy, predictive perception, and path and motion planning. Also, the integration of DRL in the autonomous driving system is heavily influenced by the advantages of exploration and exploitation features in RL [12]–[17]. Yu [13] experimented with using DRL to capture the environment object using a camera and detect the distance between the agent and the object using a LIDAR sensor. They produce output in terms of Q-Value. The reward and punishment value will be recorded within Q-Value into experience replay. In the end, the experience replay will have a role in updating the weight in the network, which will affect the agent in finding the destination point. Sallab *et al.* [12] incorporated recurrent neural network (RNN) for information integration in their proposed DRL framework for autonomous driving. They used long short-term memory (LSTM) networks to model the long-term dependencies on previous states as an addition to the current observation.

Legrand *et al.* [17] investigated whether DRL can train efficient self-driving cars by applying DRL in single and multi-agent settings. In his experiment, Legrand *et al.* [17] tried and identified what kind of neural networks perform well and how single-agent models can improve multi-agent learning. Yurtsever *et al.* [14] proposed a hybrid approach for integrating a path planning pipe into a vision-based DRL framework. They trained the agent to follow the path planner’s waypoints and defined a reward function to give the penalty to the agent when he strayed away from the path or had a collision. Wang *et al.* [15] applied the deep deterministic policy gradient (DDPG) algorithm [18] in their proposed DRL framework, designing their network architecture for both actors and critics inside DDPG. They used the open racing car simulator (TORCS) as a simulation environment. Further, Pérez-Gil *et al.* [16] compared the implementation of deep Q-network (DQN) [19] and DDPG in the DRL framework. They used the open-source simulator car learning to act (CARLA) as a simulation environment.

Meanwhile, managing the trade-off between exploration and exploitation has become one of the main challenges in reinforcement learning. The learning agent must not only explore the unknown to make better choices for future actions but must also exploit what he already knows to gain a reward. Based on this condition, the proximal policy optimization (PPO) algorithm makes the exploration and exploitation process more efficient [20]. In addition, using PPO through a DRL framework has become a promising approach to controlling multiple autonomous vehicles [21]–[23]. On the other hand, several simulation approaches have been used in autonomous driving research to overcome the problem of training and validating the driving control models [16], [24], [25].

This paper proposed an autonomous driving system using PPO-based DRL in a simulation environment, Unity3D [24]. The agent's mission in this system is to reach the destination point fastest without hitting the wall, besides dealing with the challenges of having many road intersections in getting to the destination point. We defined three kinds of reward values the agent obtained for each action it took which may affect the application of PPO. This paper is organized: section 2 explains our proposed system. Section 3 describes the experimental result and analysis. Finally, the conclusion is given in section 4.

2. PROPOSED SYSTEM

This research aims to build an autonomous driving system that can make a car move without a driver and not cause an accident. The agent is said to succeed if he meets three conditions, which are i) the agent must not hit the wall; ii) the agent reaches the destination; iii) the agent reaches the destination in the fastest way. The explanations of those three conditions are:

a) The agent must not hit the wall. The agent must stay on track and not hit the wall or the initial point in a specified time step parameter.
b) The agent reaches the destination. The agent can reach the destination point successfully in a specified time step parameter.
c) The agent can find the fastest way to reach the destination point. The agent must reach the destination point and get a reward value as much as possible in a specified time step parameter. The reward value

might take effect in measuring the distance to the destination point or the number of time step the agent has taken.

Since we implement the system in a simulation environment, the term 'wall' can represent many road conditions that a car must not hit, such as pedestrian paths, buildings, or other vehicles. The stage of the process of our proposed autonomous driving system can be seen in Figure 1.

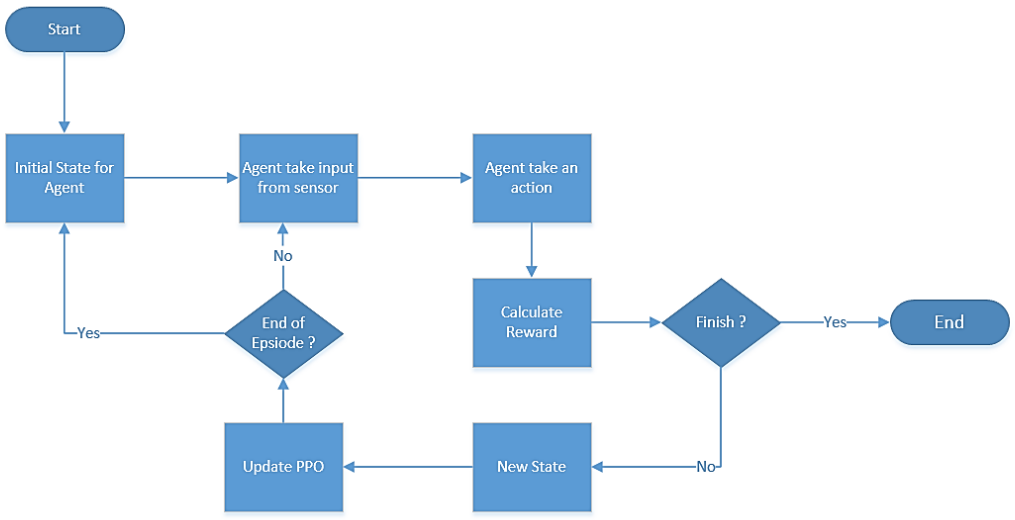


Figure 1. Proposed autonomous driving system

2.1. Setting up the initial state for agent

In this step, every agent has its track by default. Each track has five initial points, as depicted in Figure 2. The maximum agent we use is up to ten, and the minimum agent is one. Each agent must move along the track until he reaches the destination point from its initial point. To give the agent variance information about their initial state, we take at least two agents at the same initial point in the experiment.

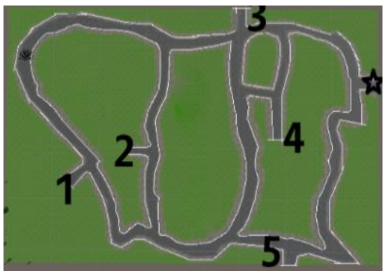


Figure 2. The initial points are depicted with numbers 1, 2, 3, 4, and 5, while the destination point is shown with a 'star' sign

2.2. Agent received input from sensor

In this stage, the agent receives inputs from each sensor the agent has, with the value of each sensor being random. Based on those values, the agent observes to choose the best action, which is assumed to have contributed to maximizing the reward. The observation process will be supported by the value of the policy gradient known as the policy parameter or instinct.

2.3. Agent takes an action

This step will be conducted after the agent observes the sensor input value and predicts the best action the agent might take. The agent will execute the action to get the reward value and create a new state. There are 3 (three) types of actions that the agent might use, namely:

a) Accelerate: the value ranges from 0 to 1; 0 for not using acceleration at all and 1 for using maximum acceleration.
b) Hit the brake: the value ranges from 0 to 1; 0 for not using the brake and 1 for using a maximum brake.
c) Turn the steer: the value ranges from -1 to 1; -1 for rotating the steer to the left at maximum and 1 for rotating the steer to the right at maximum.

2.4. Calculating reward

Each action the agent carries out will obtain a reward value. We designed 3 (three) kinds of reward values based on the conditions the agent must meet, as mentioned at the beginning of section 2. In subsection 2.4.1, 2.4.2, and 2.4.3 we explained the formula used for determining the reward.

2.4.1. Regarding distance to radius

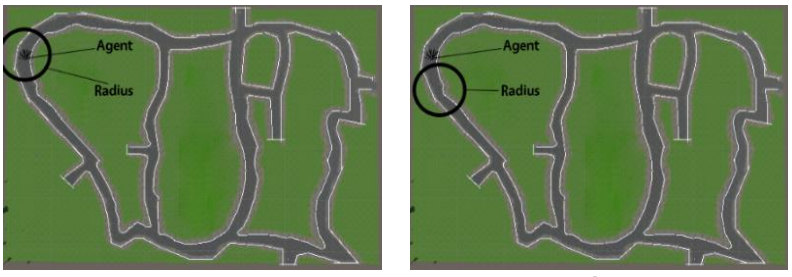
Formula (1) encourages the agent to move forward based on the agent's reward value. The reward depends on the position of the agent regarding a particular radius Rad value in time step t, with $t = 0, 1, 2, \dots, T$. A_t represents the agent position from radius Rad of time step t, which also shows whether the agent is in the coverage of the radius determined.

$$Reward = \begin{cases} \frac{-1}{\sqrt{\sum_{t=1}^{T}(A_t - A_{t-1})^2}} & , A_t < Rad \\ 0 & , A_t \geq Rad \end{cases} \tag{1}$$

Suppose the agent is still inside the radius coverage. In that case, the agent will get a penalty based on the Euclidean distance value from the current agent position A_t with the previous agent position A_{t-1}. The closer the agent to the radius edge, the more minor the penalty will be. The example of the reward value obtained by the agent regarding the agent's distance to radius is shown in Table 1. To clarify, in Figure 3, we illustrate the agent condition inside and outside the radius. Figure 3(a) shows the agent's position inside the radius. Meanwhile, in Figure 3(b), the agent appears outside the radius.

Table 1. Example of reward values regarding agent's distance to the radius

Distance to radius	Reward value	Explanation
50	-0.02	Inside the radius, near the radius edge
1	-1	Inside the radius, far from the radius edge
≥ 60	0	Outside the radius



(a) (b)

Figure 3. The agent position (a) the agent inside the radius and (b) the agent is outside the radius. A black circle represents the radius

2.4.2. Regarding maximum time step

In the running simulation, the agent is given a limit in carrying out the action. The time step is a variable that shows each action performed by the agent to produce a new state. We configure the reward value for each time step the agent takes to encourage the agent to find the destination.

$$Reward = -1 + \left(-0.01 * \sqrt{(A_c - A_f)^2}\right) \tag{2}$$

Formula (2) is used to calculate the reward when the agent reaches the maximum time step by considering the distance between the agent's current position vector A_c and the finish position vector A_f using

Euclidean distance, where c and f represent the coordinate position of the agent A. The more distant the agent from the destination point, the smaller the reward value obtained. If the agent reaches the maximum time step and the distance between the agent and the finish point is too far, the agent will get a significant penalty value. This applied to the opposite.

2.4.3. Regarding collision

Our simulation track has three types of walls: wall, initial point, and destination point. We define different reward values if the agent hits those walls, as shown in (3):

a) If the agent hits the destination point, the agent will get the maximum reward (which is 15), reduced by the proportion of the total steps that the agent used.
b) If the agent hits the wall, the agent will get reward -1, added with the proportion of the total steps the agent used and Euclidean distance value between the agent's current position A_c and finish position A_f.
c) If the agent hits the initial point, the agent will get a reward value of -10, added with the proportion of the total steps used.

The proportion of total steps is calculated by comparing the last time step t (where the agent was at the time of the collision) with the maximum time step T.

$$Reward = \begin{cases} 15 - \frac{t}{T} & , agent\ hits\ the\ destination\ point \\ \left(-1 + \frac{t}{T}\right) + \left(-0.01 * \sqrt{(A_c - A_f)^2}\right) & , agent\ hits\ the\ wall \\ -10 + \frac{t}{T} & , agent\ hits\ the\ initial\ point \end{cases} \quad (3)$$

2.5. Checking the end of training process

The training process will be finished when it reaches the maximum number of episodes specified in the experiment. We set the maximum number of episodes to five million episodes. In that condition, if the agent does not hit the track wall or does not yet reach the maximum step, the agent will go back to observe the input in the new state. Contrarily, the agent will be spawned again to the initial point.

2.6. Starting a new state

The agent will create a new state after the agent does the action and has got the reward value based on the action that has been taken. The agent will record the previous action when the agent has a new state. In our proposed system, there will be a chance for the action and step to be used again as we were using PPO for the policy gradient. The explanation of why the same action might be reused in PPO will be explained in the following process.

2.7. Updating PPO

The agent will update the gradient using the PPO algorithm in this stage. The agent uses policy parameters or instincts to choose an action based on this gradient. The process in policy gradient is shown in Figure 4. The explanation of Figure 4 is: i) the agent takes action based on instinct (π); ii) the agent does the action A_t; iii) the agent has formed a new state; iv) the agent adjusts the instinct based on the total reward R_t; and v) the agent takes further action based on the observed state S_t. The main point in the policy gradient is configuring instinct. To get the instinct value, we must define a set of parameters (θ). The total reward $J(\theta)$ for a given trajectory (or time step) $r(\tau)$ is calculated using (4).

$$J(\theta) = \mathbb{E}_{\pi}[r(\tau)] \quad (4)$$

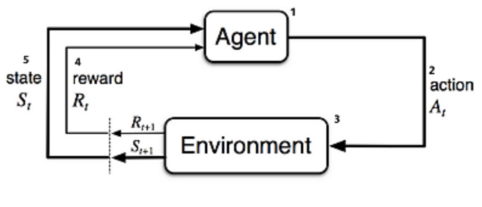


Figure 4. Policy gradient flow

Formula (4) shows that the policy gradient maximizes the expected reward following a parametrized policy $\mathbb{E}_{\pi}$. PPO tries to optimize the policy parameters by using a clipped surrogate objective ($L^{CLIP}(\theta)$), which follows the calculation as shown in (5) [20]. θ is the policy parameter, $\hat{E}_t$ denotes the empirical expectation over time step, $r_t(\theta)$ is the probability ratio between the old and new policies, and $\hat{A}_t$ is the estimated advantage at time step t. PPO imposes the constraint by forcing $r_t(\theta)$ to stay within a small interval around 1, precisely $[1-\varepsilon, 1+\varepsilon]$, where ε is a hyperparameter, usually 0.1 or 0.2.

$$L^{CLIP}(\theta) = \hat{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\right)\right] \quad (5)$$

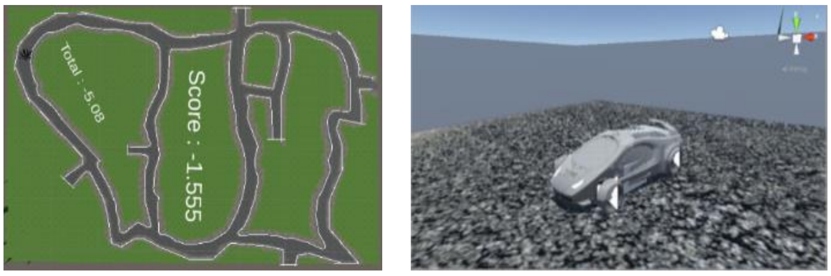
Based on the total reward that the agent obtained, the PPO formula will choose between using the old policy parameter or updating the new policy parameter. After the policy parameter is updated, the instinct will also be updated. That is why it will give the agent better instinct [20].

3. EXPERIMENTAL RESULTS AND DISCUSSION

This section explains the result and analysis of simulating our proposed autonomous driving system in Unity3D autonomous driving simulation [24]. We also provide the data and the parameter set used in our simulation. We also explain the various conditions of the agent when trying to reach the destination point in the simulation we designed.

3.1. Data and parameter setting

We create a simulation environment representing the actual track of one area in Bandar Lampung, Indonesia, as seen in Figure 5. The challenge of this track is that it has many intersections, as seen in Figure 5(a). Each time the agent encounters an intersection, the agent must take corrective actions to reach the destination. As agents, we use the car model depicted in Figure 5(b). The number of agents used in the experiment ranges from 1 to 10. Each agent has five sensors, which are used to detect the environment, such as walls, initial point, and destination point. We use the Raycast provided by Unity3D software [24] for the sensor to represent LIDAR. In our simulation, the agent can move forward, hit the brake, and take a turn based on the wheeling steer.



(a) (b)

Figure 5. The illustration of the simulation environment in Unity3D (a) sample track and (b) the agent

We conducted ten scenarios using the parameter set as shown in Table 2. Each scenario is applied to one track, so in total, we have ten similar tracks in conducting the experiment. This parameter setting affects the process in PPO, like a chance to maintain the old policies and update the instinct. The parameter values used in each column other than "*observation_vector* (P8)" follow the recommendations from Unity3D. The description of each parameter used in our experiment is: a) *Use_curiosity*: the agent will have more varied steps; we set this parameter to *true* for all experiments, b) *Gamma (P1):* discount factor for future rewards, c) *Learning_rate (P2):* the strength of each step to update gradient descent, d) *Batch_size (P3):* the number of experiences in one iteration, e) *Buffer_size (P4):* used to store the experiences conducted by agents before the model update process, f) *Beta (P5):* entropy that affects the exploration level of the agent, g) *Time_horizon (P6):* number of experience step, h) *Max_step (P7):* number of the maximum time step for the agent, i) *Observation_vector (P8):* input value that observed for choosing action value to get the maximum possible reward. The value in the *observation_vector* parameter comes from the five sensors with each sensor value in $(x, y)\ coordinate$, represents the interaction value between each sensor and the wall, initial point, and destination point, acceleration value, brake value, and steer value.

In the training process, the agent learns how to move forward and find a way to reach the destination point. Each agent is positioned on a different track and at other initial points. We use ten tracks in the training process, each having 1 to 10 agents. The result of this training process is a model with information about the agent's steps to get the maximum reward. We also use ten tracks in the testing process. In the testing, how the agent moves forward will differ depending on the model resulting from the training process. Placing the agent at a random initial point can indicate whether the agent is succeeded or not in reaching the destination point.

Table 2. Parameter setting for ten scenarios. *P1=gamma, P2=learning rate, P3=batch size,P4=buffer size, P5=entropy, P6=time horizon, P7= maximum time step, P8=obervation_vector*

No	P1	P2	P3	P4	P5	P6	P7	P8	#agent
1	0.8	1.00E-05	512	2048	1.00E-04	32	7000	28	1
2	0.8	1.00E-05	512	2048	1.00E-04	32	7000	28	10
3	0.8	1.00E-05	512	2048	1.00E-04	32	7000	28	10
4	0.85	1.00E-05	512	2048	1.00E-04	32	5000	28	10
5	0.85	1.00E-04	512	2048	1.00E-04	32	5000	30	10
6	0.95	1.00E-04	512	5120	1.00E-04	2048	3000	30	10
7	0.9	1.00E-05	2024	5120	1.00E-03	2048	3000	31	10
8	0.9	1.00E-05	2024	5120	1.00E-03	2048	3000	31	10
9	0.9	1.00E-05	512	5120	1.00E-03	1280	3000	32	10
10	0.9	1.00E-05	2024	20240	1.00E-03	1280	3000	32	10

3.2. Result and analysis

Here we show some visualization of screen-captured the agent's position in the track in time step t. Figure 6 until Figure 9 shows the agent movement, which starts at 1st initial point and succeeds in reaching the destination point, with its total score obtained. Figure 6 shows the visualization of agent position at time step $t = 10$. It can be seen in Figure 7 that the agent has moved position at time step $t = 30$. Furthermore, at the time step $t = 50$ in Figure 8, the agent has moved closer to the destination point. Finally, in Figure 9, at the time step $t = 70$, the agent managed to reach the destination point.

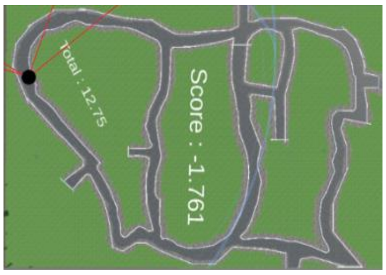


Figure 6. Agent (black circle) with ray beam (red) from its sensors at $t = 10$

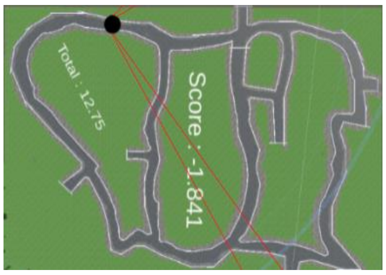


Figure 7. Agent (black circle) with ray beam (red) from its sensors at $t = 30$

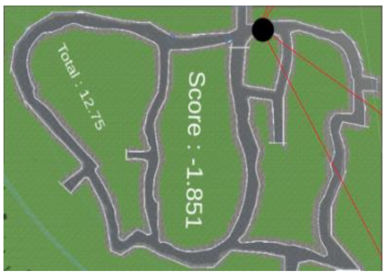


Figure 8. Agent (black circle) with ray beam (red) from its sensors at $t = 50$

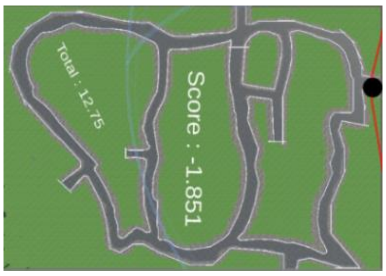


Figure 9. Agent (black circle) with ray beam (red) from its sensors at $t = 70$

As shown in Figures 6 to 9, we use 2 (two) scores in the experiment, namely 'score' and 'total'. 'Score' is a value that provides information about the reward agents obtained in each time step. 'Total' is a value that provides information about the total reward obtained after the agent hits the wall or reaches the maximum step limit. An agent's maximum score is 15, and the minimum score an agent can get is an infinite negative. This score can be assumed to determine whether the agent succeeded in the training process. The example of score value when the agent hits the wall is shown in Table 3.

Table 3. Example of score value regarding a collision

Distance to finish point	Last_step	Score value
500	500	14.9
500	3500	14.3
20	500	14.9
20	3500	14.3

Table 4 shows the result of our experiments consisting of ten scenarios with the parameter values for each scenario shown in Table 2. Max_reward: maximum total reward value that agent ever obtained; min_reward: minimum total reward value that agent ever obtained; #success_agent: the number of agents that successfully reach the destination point, #total agent: the number of agents used in each track. The explanation of each scenario's result is as follows:

Result of scenario #1. The agents only stay in place. Even if agents remain in the area, it does not mean they do nothing. The value of the acceleration action on the agent never exceeds the value of the brake action on the agent. This condition makes the agent cannot move forward.

Result of scenario #2. The agent slowly moves forward but still hits the wall. In this experiment, we configure an agent's reward and then increase the number of agents. Compared to scenario #1, this scenario is better as the agent succeeds in moving forward even with prolonged speed. However, the agent still cannot turn left or right correctly. In addition to not successfully turning, agents tend to crash into the "wall" before finding a turning trajectory. This experiment does better because of the number of agents added to the training process. Each agent has its environment to explore. These conditions help agents to distinguish between the track and the wall. For the case of an agent who crashes too quickly, this is still due to the provision of less-than-optimal rewards. In the end, the agent is stuck at the local maximum.

Result of scenario #3. The agent slowly moves forward but succeeds in turning to the right. We focus on configuring the reward for this experiment, such as not giving too much penalty or extreme reward value when hitting the destination. The agent learns how to turn with the same parameter setting but is still very slow to move forward.

Result of scenario #4. The agent can move forward but hit the wall as soon as possible. We encourage the agent to move faster in this experiment, giving a time limit and radius. If the agent reaches the time limit and is still inside the radius, the agent will get a reduction in a reward. This can be beneficial or harmful based on the size of the radius or the ability agent to explore. We found in this experiment that the agent cannot move forward like in the previous experiment.

We observed that the agent is trapped in a local maximum because the penalty value obtained when hitting the wall before the time limit is more minor than after the time limit. There are 2 (two) conditions of why the agent chooses to hit the wall as soon as possible: a) the agent reaches the time limit and is still inside the radius, then the agent hits the wall or reaches the maximum step limit (the agent get 2 (two) punishment); b) The agent is inside the radius but hit the wall or reach maximum step limit before agent reach the time limit (the agent get one punishment). Based on the comparison between (a) and (b), the agent tends to choose the action which leads to (b) just because the (b) condition is more profitable to the agent.

Result of scenario #5. The agent can move forward but not towards the destination. We add acceleration observation in this experiment, in addition, to reward values to encourage agents to move toward the destination point. Acceleration is the value of the agent speed at each step, calculated based on the x and y vector using the velocity formula. Observation of acceleration is used to encourage the agent to move forward at the proper speed. The results obtained in this scenario are pretty satisfactory because the agent fulfills the first objective, not to hit the wall. However, the agent cannot achieve the second goal, which is not driving toward the destination point.

Result of scenario #6. The agents normally run, but only 2 (two) agents reach the destination point. For this experiment, we change some parameters, such as learning rate, buffer_size, time_horizon, and max_step. The 2 (two) agents that succeeded in exploring and finding destinations are the agents who came from the same initial point. The other agent starts from a different initial point does not move towards the destination.

Result of scenario #7. The agents regularly run; only 2 (two) agents that reach the destination, but the cumulative reward value is better than before. In this experiment, we increase one of the parameters, namely batch_size, which helps use each iteration experience. Besides that, we add a new observation, which is step observation, which affects the agent's reward. The more steps, the bigger the penalty value the agent obtained. It makes the agent use the step more efficiently.

Result of scenario #8. All agents are initiated at one initial point and are failed to reach the destination. By creating the agent at the same and farthest initial point, we expect it will make it easier for the training process to be closest to the destination point. But the result obtained is far worse than before; the agent is trapped at the local maximum. We observed that if the agents are placed in different initial points, the agent's input value will be more varied than those set only in one initial point.

Result of scenario #9. The agents normally run, and all agents move to the destination, but only 2 (two) agents reach the destination. In this experiment, we decreased the parameter value for batch_size and time_horizon. This change is to make training more stable and focus on future rewards. Besides, we add a new observation value: the distance between the agent and the destination. The distance between the agent and destination is also used to calculate rewards. This value is applied to make the agent know the distance toward each step's destination. With that agent will try to move towards the destination. The result obtained for this change is quite good. Like the previous scenarios, only two agents managed to hit the destination. But the difference is that this agent comes from an initial point different from before. In this scenario, all agents also moved towards the destination, although some are stuck on the local maximum.

Result of scenario #10. The agents normally run, and all agents move to the destination, but only 5 (five) agents reach the destination point. We modify the parameter and give some obstacles on several tracks. It is intended for the agent to explore a new environment and influence other agents. We observed that each agent managed to find a destination point given an obstacle. Meanwhile, other agents that do not have barriers at their track are stuck at the local maximum.

Table 4. The result of ten scenarios of our experiment

No.	Max_reward	Min_reward	#Success_agent	#Total_agent
1	-1	-1	**0**	1
2	-3.52	-23.5	**0**	10
3	-2.07	-17.64	**0**	10
4	-2.73	-15.823	**0**	10
5	-2.07	-18.74	**0**	10
6	-0.3749	-14.67	**2**	10
7	1.877	-12.35	**2**	10
8	-5.325	-16.99	**0**	10
9	-3.31	-10.13	**2**	10
10	-2.05	-13.73	**5**	10

4. CONCLUSION

In this study, we successfully demonstrated the application of our proposed reward rules in the PPO as the policy gradient in the autonomous driving simulation system. As we all know, autonomous driving is a very complex problem. Based on the result of our experiment, we observe that it is necessary to have a proper technique for giving reward value so that the training process can go properly. For further studies, we expect to extend our work to a more complicated road track to represent high-dimensional state and action spaces, which is crucial in the autonomous driving system using deep reinforcement learning.

REFERENCES

[1] K. Bengler, K. Dietmayer, B. Farber, M. Maurer, C. Stiller, and H. Winner, "Three decades of driver assistance systems: review and future perspectives," *IEEE Intelligent Transportation Systems Magazine*, vol. 6, no. 4, pp. 6–22, 2014, doi: 10.1109/MITS.2014.2336271.

[2] L. Fridman *et al.*, "MIT advanced vehicle technology study: large-scale naturalistic driving study of driver behavior and interaction with automation," *IEEE Access*, vol. 7, pp. 102021–102038, 2019, doi: 10.1109/ACCESS.2019.2926040.

[3] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020, doi: 10.1002/rob.21918.

[4] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: automated testing of deep-neural-network-driven autonomous cars," *Proceedings of the 40th International Conference on Software Engineering*, vol. 7, no. 1, pp. 37–72, Aug. 2017, doi: 10.1145/3180155.

[5] L. Li, K. Ota, and M. Dong, "Humanlike driving: empirical decision-making system for autonomous vehicles," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 8, pp. 6814–6823, 2018, doi: 10.1109/TVT.2018.2822762.

[6] B. Wu, F. Iandola, P. H. Jin, and K. Keutzer, "SqueezeDet: unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2017, vol. 2017-July, pp. 446–454, doi: 10.1109/CVPRW.2017.60.

[7] P. J. Navarro, L. Miller, F. Rosique, C. Fernández-Isla, and A. Gila-Navarro, "End-to-end deep neural network architectures for speed and steering wheel angle prediction in autonomous driving," *Electronics*, vol. 10, no. 11, p. 1266, May 2021, doi: 10.3390/electronics10111266.

[8] M. Teti, W. E. Hahn, S. Martin, C. Teti, and E. Barenholtz, “A controlled investigation of behaviorally-cloned deep neural network behaviors in an autonomous steering task,” *Robotics and Autonomous Systems*, vol. 142, 2021, doi: 10.1016/j.robot.2021.103780.
[9] Y. Dai and S.-G. Lee, “Perception, planning and control for self-driving system based on on-board sensors,” *Advances in Mechanical Engineering*, vol. 12, no. 9, Sep. 2020, doi: 10.1177/1687814020956494.
[10] B. R. Kiran *et al.*, “Deep reinforcement learning for autonomous driving: a survey,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4909–4926, 2022, doi: 10.1109/TITS.2021.3054625.
[11] F. Ye, S. Zhang, P. Wang, and C. Y. Chan, “A survey of deep reinforcement learning algorithms for motion planning and control of autonomous vehicles,” *IEEE Intelligent Vehicles Symposium, Proceedings*, vol. 2021-July, pp. 1073–1080, 2021, doi: 10.1109/IV48863.2021.9575880.
[12] A. El Sallab, M. Abdou, E. Perot, and S. Yogamani, “Deep reinforcement learning framework for autonomous driving,” *IS and T International Symposium on Electronic Imaging Science and Technology*, pp. 70–76, 2017, doi: 10.2352/ISSN.2470-1173.2017.19.AVM-023.
[13] A. Yu, “Deep reinforcement learning for simulated autonomous vehicle control,” *Course Project Reports*, vol. 42, no. 1–4, pp. 229–233, Jun. 2016.
[14] E. Yurtsever, L. Capito, K. Redmill, and U. Ozgune, “Integrating deep reinforcement learning with model-based path planners for automated driving,” in *IEEE Intelligent Vehicles Symposium, Proceedings*, 2020, pp. 1311–1316, doi: 10.1109/IV47402.2020.9304735.
[15] S. Wang, D. Jia, and X. Weng, “Deep reinforcement learning for autonomous driving,” *Multimedia Tools and Applications*, vol. 81, no. 3, pp. 3553–3576, Nov. 2018, doi: 10.1007/s11042-021-11437-3.
[16] Ó. Pérez-Gil *et al.*, “Deep reinforcement learning based control for autonomous vehicles in CARLA,” *Multimedia Tools and Applications*, vol. 81, no. 3, pp. 3553–3576, Jan. 2022, doi: 10.1007/s11042-021-11437-3.
[17] D. R. M. Legrand, R. Radulescu and A. Nowe, “The SimuLane highway traffic simulator for multi-agent reinforcement learning — Vrije Universiteit Brussel,” in *Proceedings of the 29th Benelux Conference on Artificial Intelligence*, Apr. 2017, pp. 394–396, doi: 10.48550/arXiv.1604.06778.
[18] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, “Benchmarking deep reinforcement learning for continuous control,” *33rd International Conference on Machine Learning, ICML 2016*, vol. 3, no. December, pp. 2001–2014, Apr. 2016, doi: 10.48550/arXiv.1901.00137.
[19] J. Fan, Z. Wang, Y. Xie, and Z. Yang, “A theoretical analysis of deep Q-learning,” in *Proceedings of the 2nd Conference on Learning for Dynamics and Control, PMLR*, Jan. 2020, vol. 120, pp. 1–6.
[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv:1707.06347*, 2017, doi: 10.48550/arXiv.1707.06347.
[21] D. Q. Tran and S. H. Bae, “Proximal policy optimization through a deep reinforcement learning framework formultiple autonomous vehicles at a non-signalized intersection,” *Applied Sciences (Switzerland)*, vol. 10, no. 16, 2020, doi: 10.3390/app10165722.
[22] Y. Wu, S. Liao, X. Liu, Z. Li, and R. Lu, “Deep reinforcement learning on autonomous driving policy with auxiliary critic network,” *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–11, 2021, doi: 10.1109/TNNLS.2021.3116063.
[23] F. Ye, X. Cheng, P. Wang, C. Y. Chan, and J. Zhang, “Automated lane change strategy using proximal policy optimization-based deep reinforcement learning,” in *IEEE Intelligent Vehicles Symposium, Proceedings*, 2020, pp. 1746–1752, doi: 10.1109/IV47402.2020.9304668.
[24] A. Juliani *et al.*, “Unity: a general platform for intelligent agents,” *arXiv:1809.02627*, 2018, doi: 10.48550/arXiv.1809.02627.
[25] G. Rong *et al.*, “LGSVL simulator: a high fidelity simulator for autonomous driving,” 2020, doi: 10.1109/ITSC45102.2020.9294422.

BIOGRAPHIES OF AUTHORS

Imam Noerhenda Yazid received the B.Sc. in Informatics Engineering from Telkom University, Indonesia in 2019. Focused and detail-oriented Unity3D engine expert with a focus on video game development, and experienced in C# progamming languages. Creative and independent individual dedicated to innovation, exposition, and efficiently resolving project issues. Currently, he pursued his career as Game Programmer in Algorocks. He can be contacted at LinkedIn: linkedin.com/in/imamnoerhenda, email: imamnoerhenda@gmail.com.

Ema Rachmawati received the B.Sc. in Informatics Engineering from Institut Teknologi Bandung (ITB), Indonesia in 2004, the M.Sc in Informatics Engineering from Institut Teknologi Bandung (ITB), Indonesia in 2008, and the Ph.D. in Electrical Engineering and Informatics from Institut Teknologi Bandung (ITB), Indonesia in 2018. Since 2010, she joined Telkom University as a lecturer in the School of Computing. Her research interests include machine learning, object recognition, and image understanding. She can be contacted at e-mail: emarachmawati@telkomuniversity.ac.id.