electronics

MDPI

Article

Autonomous Driving Control for Passing Unsignalized Intersections Using the Semantic Segmentation Technique

Jichiang Tsai *, Yuan-Tsun Chang, Zhi-Yuan Chen and Zhehao You

Department of Electrical Engineering, Graduate Institute of Communication Engineering, National Chung Hsing University, Taichung 402202, Taiwan; d110064007@mail.nchu.edu.tw (Y.-T.C.); g110064009@mail.nchu.edu.tw (Z.-Y.C.); g111064801@mail.nchu.edu.tw (Z.Y.)
* Correspondence: jichiangt@nchu.edu.tw

Abstract: Autonomous driving in urban areas is challenging because it requires understanding vehicle movements, traffic rules, map topologies and unknown environments in the highly complex driving environment, and thus typical urban traffic scenarios include various potentially hazardous situations. Therefore, training self-driving cars by using traditional deep learning models not only requires the labelling of numerous datasets but also takes a large amount of time. Because of this, it is important to find better alternatives for effectively training self-driving cars to handle vehicle behavior and complex road shapes in dynamic environments and to follow line guidance information. In this paper, we propose a method for training a self-driving car in simulated urban traffic scenarios to be able to judge the road conditions on its own for crossing an unsignalized intersection. In order to identify the behavior of traffic flow at the intersection, we use the CARLA (CAR Learning to Act) self-driving car simulator to build the intersection environment and simulate the process of traffic operation. Moreover, we attempt to use the DDPG (Deep Deterministic Policy Gradient) and RDPG (Recurrent Deterministic Policy Gradient) learning algorithms of the DRL (Deep Reinforcement Learning) technology to train models based on the CNN (Convolutional Neural Network) architecture. Specifically, the observation image of the semantic segmentation camera installed on the self-driving car and the vehicle speed are used as the model input. Moreover, we design an appropriate reward mechanism for performing training according to the current situation of the self-driving car judged from sensing data of the obstacle sensor, collision sensor and lane invasion detector. Doing so can improve the convergence speed of the model to achieve the purpose of the self-driving car autonomously judging the driving paths so as to accomplish accurate and stable autonomous driving control.

Keywords: self-driving cars; deep reinforcement learning (DRL); deep deterministic policy gradient (DDPG); recurrent deterministic policy gradient (RDPG); CARLA (CAR Learning to Act)

Citation: Tsai, J.; Chang, Y.-T.; Chen, Z.-Y.; You, Z. Autonomous Driving Control for Passing Unsignalized Intersections Using the Semantic Segmentation Technique. *Electronics* **2024**, *13*, 484. https://doi.org/10.3390/electronics13030484

Academic Editor: Mohamed Karray

Received: 26 December 2023
Revised: 19 January 2024
Accepted: 22 January 2024
Published: 24 January 2024

1. Introduction

In recent years, the advent of Artificial Intelligence (AI) and rapid advancements in chip technologies have revolutionized the field of autonomous driving [1]. AI-powered systems, combined with sophisticated sensors and efficient processing units, have propelled the development of self-driving vehicles, representing a significant milestone in the transportation industry. This intersection of AI and chip technologies has not only enhanced the capabilities of autonomous vehicles but has also paved the way for a safer, more efficient and sustainable future in transportation.

The escalation of traffic accidents at intersections, especially those without traffic signals in densely populated urban areas, underscores the pressing need for enhanced safety measures in the realm of autonomous driving. Statistics reveal a staggering 35% increase in the incidence of fatalities and injuries at unsignalized intersections [2], highlighting the urgency for autonomous driving technologies to possess an elevated capacity to

adapt to unpredictable scenarios. These areas present complex and dynamic environments, demanding that autonomous driving technologies possess an elevated capacity to adapt to unpredictable scenarios. The intricacies of navigating through uncontrolled intersections and densely trafficked cityscapes require autonomous vehicles to swiftly process and respond to real-time changes, making split-second decisions to ensure the safety of passengers and pedestrians alike.

In the past, numerous experts and scholars have proposed the feasibility assessments of Intelligent Transportation Systems (ITS) and autonomous driving technologies [3]. These propositions emphasized the potential of establishing vehicle-to-infrastructure communication networks, fostering collective action and accident prevention. Such systems were envisioned to enhance overall traffic convenience, minimize commuting time and costs and reduce the occurrence of accidents [4]. However, due to the substantial costs associated with the initial infrastructure setup and network integration, achieving widespread adoption and implementation has posed significant challenges [5,6].

As a result, the current focus primarily revolves around the integration of Advanced Driver Assistance Systems (ADAS) and blind-spot detection systems [7] as mainstream solutions. These systems serve to alert drivers and offer limited, passive driving assistance capabilities. Yet, they fall short of meeting the aforementioned requirements, thus highlighting the necessity for further advancements in autonomous driving technologies.

To better facilitate the application of autonomous driving technologies in heavily congested urban areas, researchers have made notable strides in integrating deep learning models and simulated environments [8]. Chen et al. [9] utilized ConvNet [10] feature extractors to simulate multi-lane highway driving behaviors in the TORCS simulator [11]. Building upon this work, Sauer et al. [12] extended the application by implementing a Conditional Affordance-Based Learning Model based on VGG16 [13]. Here, the traditional PID controller serves as the decision module, mapping the model's predicted affordance indicators into actions, thereby addressing the limitations highlighted in [9] and achieving advanced autonomous driving behavior control in urban areas.

Furthermore, several studies have endeavored to integrate Deep Reinforcement Learning (DRL) with the field of autonomous driving [14]. Wolf et al. [15] employed DQN to teach self-driving cars lane-keeping behaviors in simulated environments. Kendall et al. [16] utilized forward-facing camera imagery, current vehicle speed and steering angles as inputs to train lane-keeping policies, demonstrating the applicability of DRL in real-world self-driving scenarios. Agarwal et al. [17] proposed DRL strategies for urban driving tasks, encompassing lane tracking, intersection navigation and adherence to relevant traffic rules. Their training process incorporated top-view semantic segmentation images and traffic light states, enabling the learning of driving strategies to facilitate decision-making in urban environments.

In many cases, research endeavors in the field of autonomous vehicles have leveraged simulators such as CARLA [18], TORCS [11], AirSim [19] and AWS DeepRacer [20] to minimize costs and streamline development efforts. Considering the environmental requisites and the feasibility assessment for implementation, this paper ultimately opts to utilize CARLA as the platform for self-driving car training to accomplish the goals outlined in this research.

The structure of this paper is divided into five sections. Section 1 describes the purpose and motivation for writing this paper, along with discussing related research. Section 2 provides the theories and knowledge required for this paper, e.g., DQN, the DDPG, the RDPG and so on. In Section 3, we carefully formulate an appropriate reward-generation mechanism and elaborate the two DRL models proposed by us. Section 4 constructs the simulation environments required for training and testing of our models and analyzes the experimental results. Finally, Section 5 provides a summary of this paper.

2. Preliminaries

2.1. Modular Pipeline

Modular pipeline (MP) is an approach in autonomous driving systems that comprises a series of modules, primarily including perception, decision-making and motion planning modules. Each module is tasked with specific responsibilities within the overall architecture. The perception module collects data from onboard sensors such as cameras, LiDAR and RADAR, which creates a high-dimensional state representation of the vehicle's surroundings and includes tasks such as semantic segmentation [21], object tracking [22], object detection [23] and traffic sign identification [24].

The decision-making module is responsible for receiving the vehicle's state and making decisions such as lane-keeping, left-lane changing and right-lane changing. Meanwhile, the motion planning module adheres to the decision commands and state displays, controlling actions such as steering, throttle and braking based on the instructions. Given the interdependencies between the modules, integrating them effectively becomes crucial for achieving optimal behavioral control in the concatenated applications.

2.2. Reinforcement Learning

Reinforcement learning (RL) focuses on improving an autonomous agent's behavior in accomplishing assigned tasks through interactions with the environment, resembling human or biological learning patterns. The functioning of reinforcement learning is illustrated in Figure 1, where the agent observes each step (t) and obtains the environment's state ($S_t \in S$). Based on the input state, the agent selects discrete or continuous actions ($A_t \in A$) as the output. After executing the action and interacting with the environment, the agent acquires a new state (S_{t+1}) and receives a reward ($R_{t+1} \in R$).

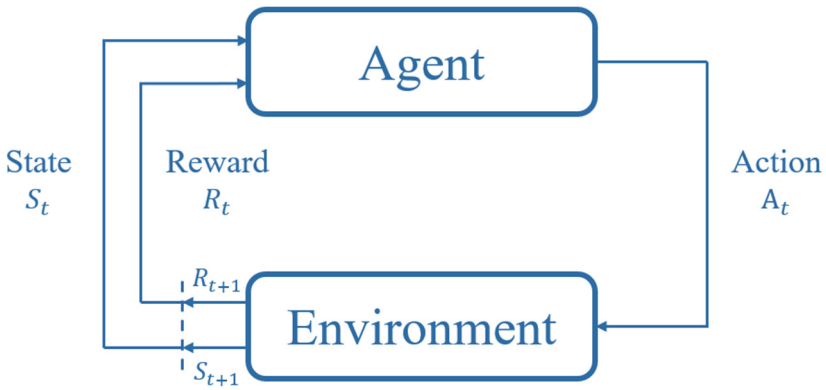


Figure 1. The procedure of reinforcement learning.

The Markov decision process (MDP) elucidates the theoretical framework and standards followed by the agent in dealing with decision-making problems within reinforcement learning. MDP is composed of a set of states (S), a set of actions (A), transition probabilities (P) and a reward function (R), denoted collectively as (S, A, P, R).

The performance of the RL agent is evaluated by the reward mechanism, making the design of such a mechanism a crucial aspect in aligning the actions executed by the agent with the desired outcomes. The goal of the agent is to maximize the cumulative reward obtained during its lifetime, which can be represented by Equation (1) [25]:

$$R = \sum_{k=0}^{H} \gamma^k r_{t+k} \quad (1)$$

Here, γ represents the discount factor that signifies the discounting value of future rewards, ranging between 0 and 1. Specifically, a value close to 0 implies a focus on immediate actions, with the agent considering short-term rewards. Conversely, a value closer to 1 signifies a more forward-thinking approach, considering future rewards to

achieve better long-term rewards. On the other hand, H refers to the time steps in the MDP, where $H = \infty$ under unrestricted conditions. However, in typical training contexts, H is often set as a finite value since scenarios terminate either after a fixed number of time steps or when the agent reaches a designated goal state, referred to as a terminal state.

When the state (s) is input, an action (a) is chosen to enter the environment, resulting in a new state (s'), a transition probability $p(s',r \mid s,a) \in (0,1)$ and a reward $R(s,a)$. The Stochastic Policy π: $S \rightarrow P(A)$ maps probabilities from the state space to the action set. The goal of reinforcement learning is to find the optimal policy π^* that maximizes the expected cumulative discounted reward, as denoted in Equation (2) [25]:

$$\pi^* = \underset{\pi}{argmax}\mathbb{E}_\pi\left\{\sum_{k=0}^{H-1}\gamma^k r_{k+1}|s_0 = s\right\} \tag{2}$$

2.3. Q Learning

Q learning is an off-policy algorithm which applies the temporal-difference (TD) [26] equation. At the beginning, the Q function and the initial state s are randomly initialized, and action a is selected according to ε-greedy. After the selected action a is applied to the environment, a reward R and a new state s′ will be obtained, and subsequently the Q value of the pair (s, a) is updated according to the following equation:

$$Q(s,a) = Q(s,a) + \alpha\left[R + \gamma \underset{a'}{max} Q(s',a') - Q(s,a)\right] \tag{3}$$

After updating the Q function, we can continue to choose a random action with ε-greedy or choose the action a with the largest Q value, repeating the above steps until the final state is reached. More specifically, the update method of Q-Learning is to continually update the Q function with the largest Q value of the next state. That is, $Q_\pi(s, a)$ represents the expected cumulative return after taking action a from state s under the policy π, as shown in Equation (4) [25]:

$$Q_\pi(s,a) = \mathbb{E}_\pi\left\{\sum_{k=0}^{H-1}\gamma^k r_{k+1}|s_0 = s, a_0 = a\right\} \tag{4}$$

2.4. DQN

In environments with large state spaces, traditional Q-Learning struggles with the impracticality of learning and storing Q-values for each state. The deep Q-Network (DQN) [27] addresses this by integrating deep learning with Q-Learning by using a deep neural network (DNN) to approximate Q-values for high-dimensional states so as to manage the vast computational complexity and storage issues. More specifically, this approach approximates the Q-value distribution using a function, as depicted in the equation below:

$$Q(s,a,\omega) \approx f(s,a,\omega) \tag{5}$$

In the above equation, $Q(s, a,\omega)$ represents the approximate Q-value with the parameter ω of the DNN, which needs to be precisely adjusted during the raining procedure.

By leveraging the power of deep learning, DQN efficiently handles complex and high-dimensional state spaces, contributing significantly to the advancement of reinforcement learning in various applications, including autonomous driving systems. Particularly, in advanced DQN versions, a target network that is identical in structure to the main network but with periodically updated parameters is used to estimate the target value for enhancing training stability, as the architecture shown in Figure 2.

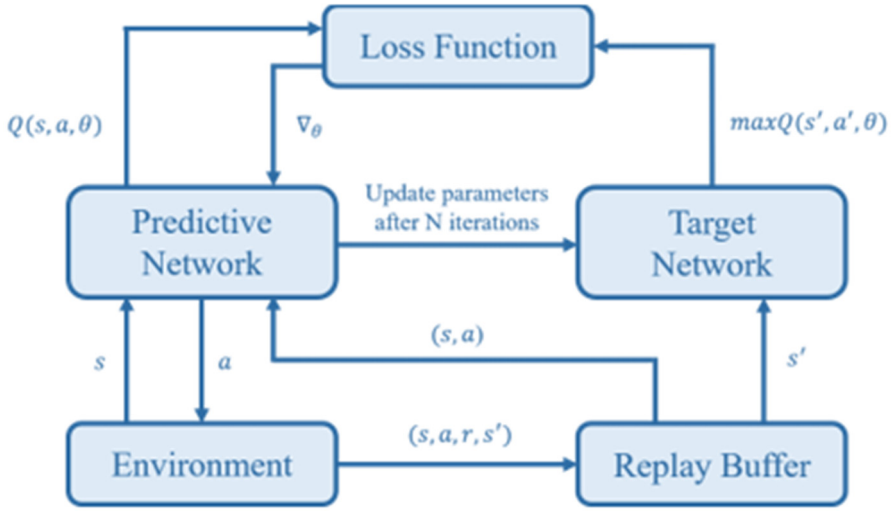


Figure 2. The architecture of DQN.

2.5. Policy Gradient

Policy Gradient and Q-Learning are two different reinforcement learning strategies. The latter seeks to learn the best actions based on the highest expected rewards using a method like ε-greedy; while the former directly learns the best policy through a probabilistic approach, determining actions based on a learned probability distribution. Within Policy Gradient, Stochastic Policy Gradient (SPG) [28] is used for discrete action spaces, adjusting policy based on probability distribution; while Deterministic Policy Gradient (DPG) [29] is better for continuous action spaces, aiming for a deterministic policy solution.

2.6. Actor–Critic

Actor–Critic [30] is an algorithm that combines Policy Gradient and value function algorithms, i.e., a hybrid of Policy Gradient and value function-based algorithms. It employs two networks for training: Actor and Critic. The Actor is trained using the Policy Gradient method, determining which action to take based on the input state. The Critic, on the other hand, uses the value function method. It updates network parameters based on the current state and action from the Actor, as well as the received reward. Based on this, it evaluates actions using the value function. Finally, the Actor improves its actions based on the Critic's evaluations.

2.7. DDPG

Deep Deterministic Policy Gradient (DDPG) [31], introduced by DeepMind in 2016, is an algorithm derived from the Actor–Critic architecture, enabling learning through both the Q-function and policy methods, effectively addressing problems with continuous action spaces. More specifically, the DDPG algorithm combines deep learning techniques with DPG, structured into the Actor and Critic networks. The Actor component utilizes a deep neural network to approximate a deterministic policy function for predicting actions; while the Critic component employs a deep neural network to approximate the state-action Q function for action value prediction. This approach resembles the methodology of DQN. Hence, DDPG can be seen as an extension of DQN tailored for continuous action spaces, which is rooted in the principles of the Actor–Critic method. Moreover, the Actor–Critic architecture is further segmented into the target network and the online network for enhancing training stability, just like DQN.

To further enhance the stability and convergence speed of the algorithm, DDPG employs the Experience Replay mechanism. This technique involves storing a collection of past experiences and then randomly sampling from this collection during training. By learning from a diverse set of experiences, the algorithm can improve its robustness and overall performance. Furthermore, OU Noise (Ornstein–Uhlenbeck Process) is typically combined

with DDPG to aid in learning deterministic policies [32]. During the training phase, OU Noise is incorporated into the actions to enhance the agent's exploration rate. Particularly, OU Noise assists the agent in exploring temporal correlations between neighboring actions, thereby enhancing the exploration intensity in the selected direction.

The structural layout of the DDPG algorithm is shown in Figure 3, emphasizing the interplay between the Actor and Critic networks and the integration of Experience Replay. Note that $\mu(\cdot)$ and $\mu'(\cdot)$ denote the output of the online network and the target network according to their input, respectively. Moreover, y_i is the temporary target Q value predicted by the Target Critic network for the Online Critic network to update its future evaluation on the Q value of the current state-action pair.

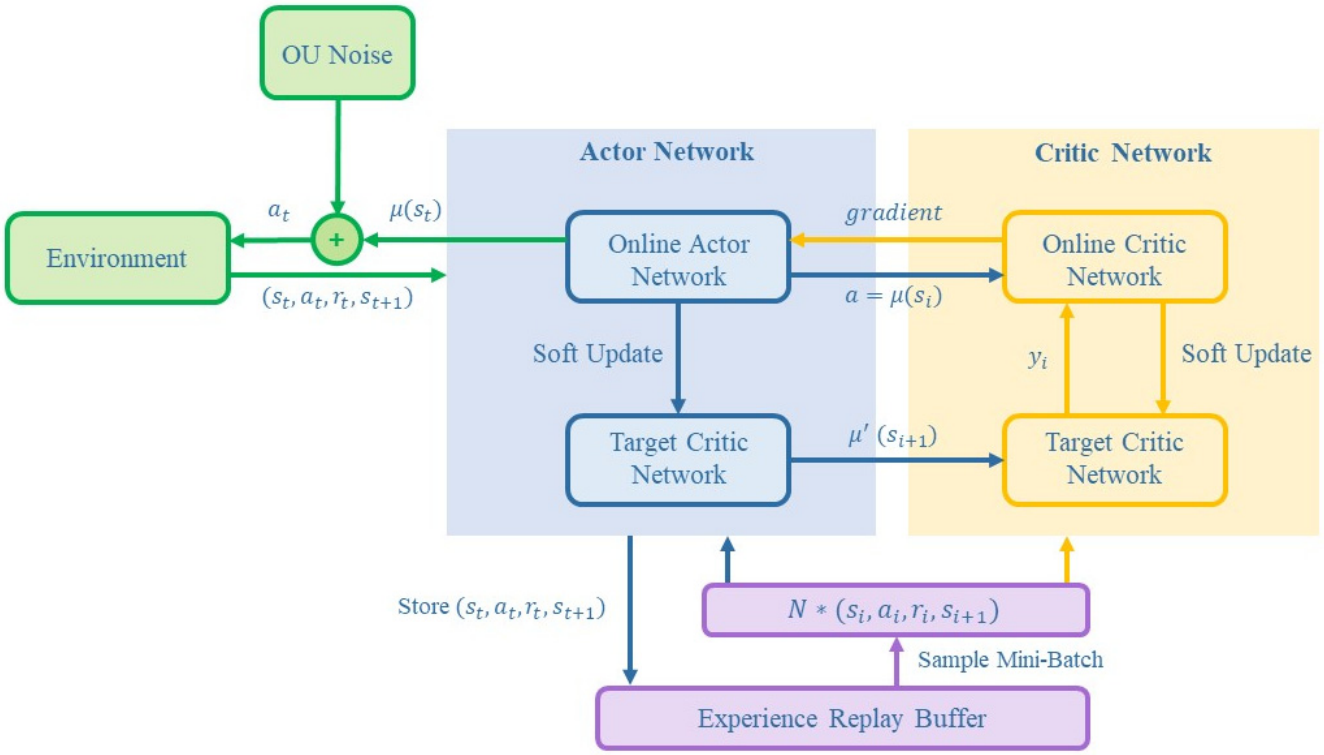


Figure 3. The architecture of DDPG.

2.8. RDPG

RDPG (Recurrent Deep Deterministic Policy Gradient) [33] is an advancement built upon the DDPG algorithm, incorporating two key modifications. First, RDPG replaces the hidden layer neurons in DDPG with long short-term memory (LSTM) cells [34]. Specifically, an LSTM cell involves three critical gate units that regulate information flow: Forget gate, Input gate and Output gate. The Forget gate determines which information to discard or retain, while the Input gate controls the incorporation of current input and new memory units. Both of these stages utilize Sigmoid functions to generate values in the [0, 1] range, dictating the proportion of data flow. Finally, the Output gate decides which information needs to be output, with the values constrained to the $[-1, 1]$ range via the Tanh function. The architecture of an LSTM cell is depicted in Figure 4.

Secondly, RDPG introduces historical states h_t as inputs, enabling the model to capture the dynamic nature and long-term influences of the environment. These enhancements equip RDPG with the capability to adapt to complex and evolving scenarios more effectively. The structural layout of the RDPG algorithm is shown in Figure 5.

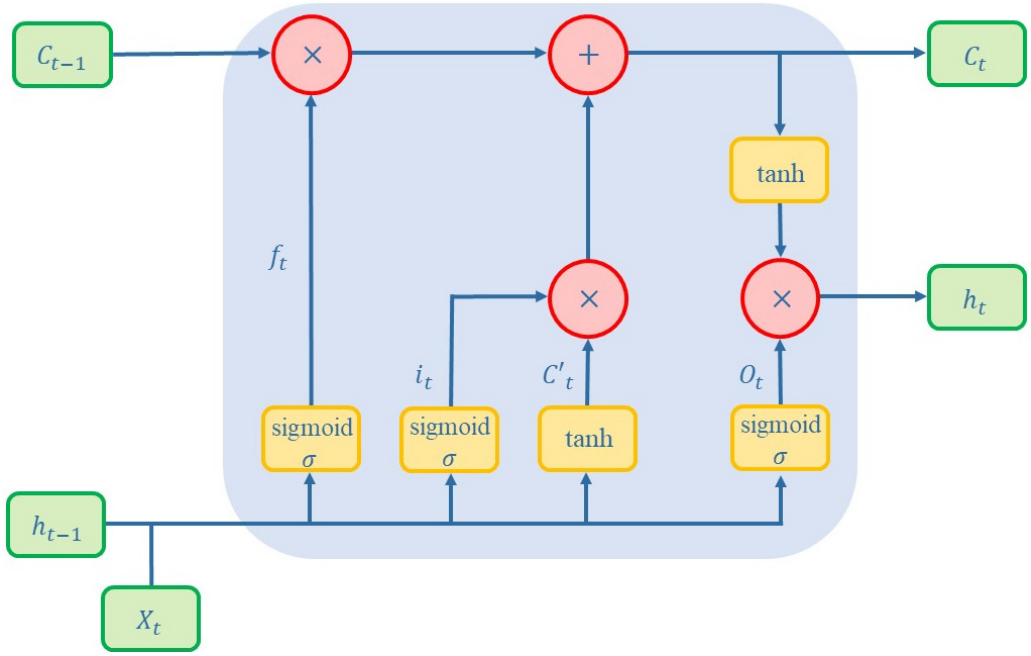


Figure 4. The architecture of an LSTM cell.

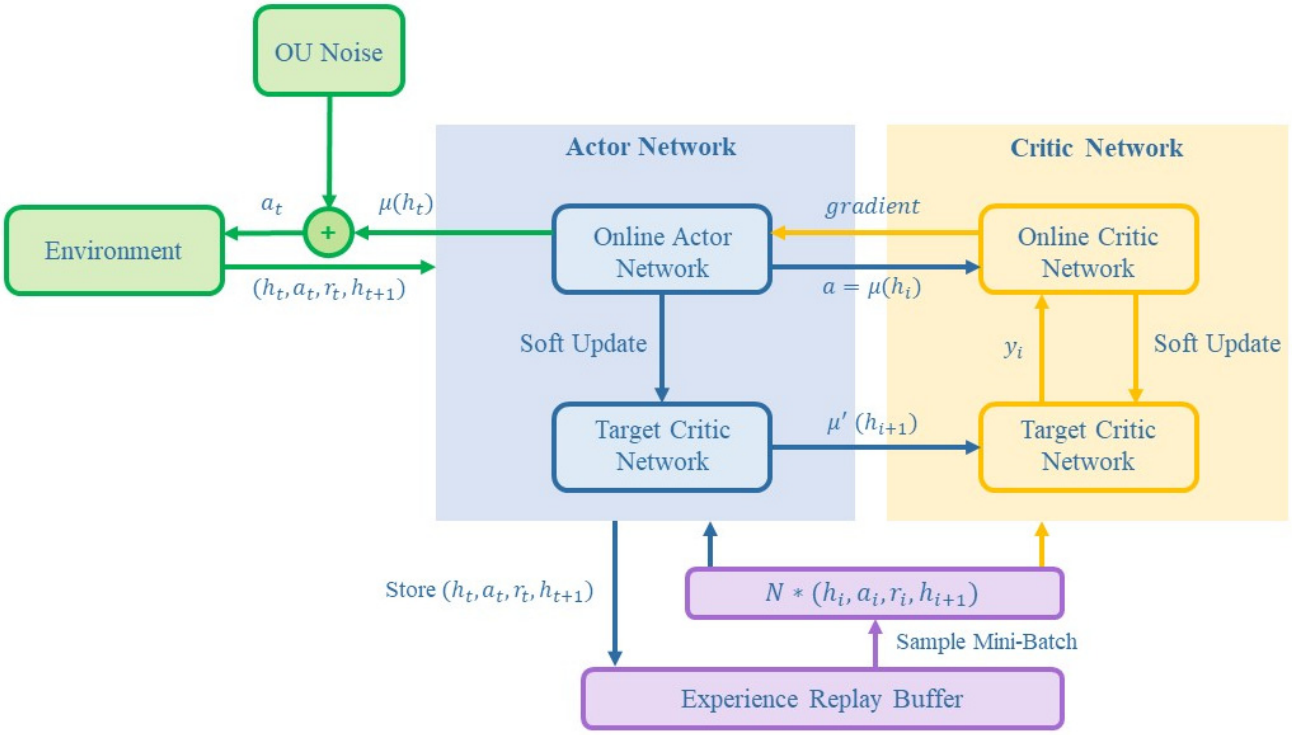


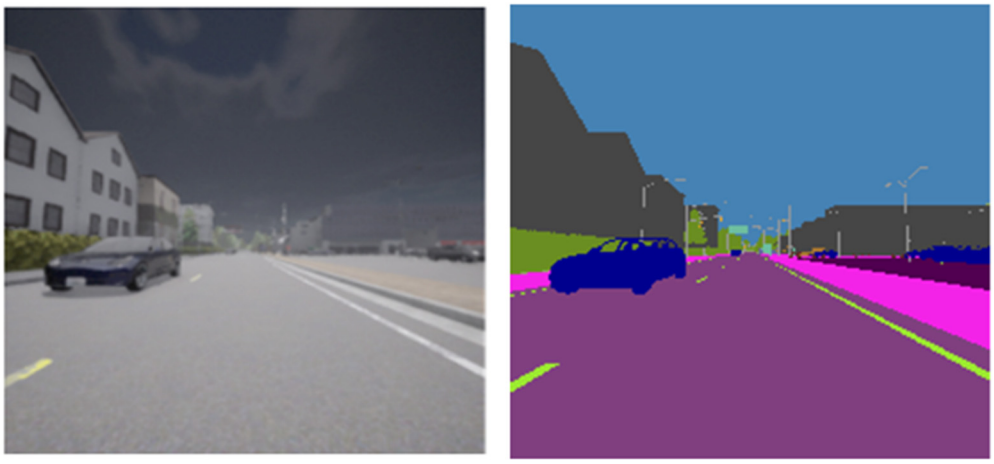
Figure 5. The architecture of RDPG.

2.9. CARLA Simulator

CARLA [18] is an open-source autonomous driving simulator focusing on urban environments. It offers detailed urban content like city layouts, vehicles, pedestrians and road signs, using the Unreal Engine for realistic scene rendering and physical behavior simulation. The simulator includes both static and dynamic 3D objects and allows for environmental customization with 18 weather and lighting conditions.

2.10. Driving Scenes

To train autonomous driving behaviors effectively in dynamic traffic environments [35,36], we employ a semantic segmentation camera on the training vehicle. This camera captures the driving scene and uses semantic segmentation to classify each pixel, simplifying the visual data by focusing on relevant features and reducing complexity. This technique's significant impact on training efficiency in deep reinforcement learning is noted. However, due to GPU memory constraints, image frames are resized to 224 × 224 pixels and undergo preprocessing to match the neural network's input format. Specifically, images from the CARLA sensor are initially in RGBA format and are converted to RGB by removing the alpha channel, which reduces image transparency but does not affect the semantic segmentation's ability to differentiate objects based on color, as shown in Figure 6 below.



(**a**) (**b**)

Figure 6. An example driving vision captured by (**a**) the ordinary camera and (**b**) the semantic segmentation camera.

3. Our DRL Models

In this section, we begin to clearly explain how we devised and trained our two DRL models for controlling a self-driving car to perform the behavior of crossing an unsignalized intersection, including turning right, going straight and especially turning left. In particular, both our models are constructed mainly based on the existing Convolutional Neural Network (CNN) architecture. Although CNN architecture is well known in building models for extracting critical features from an image to facilitate decision making, we still need to decide how many convolution layers and pooling layers should be used for our application. On the one hand, the two DDPG and RDPG learning algorithm proposed in the literature are exploited for training our DRL models. However, we have to carefully design the reward mechanism for these two algorithms to train the models to be faster and more accurate. Note that the above two issues both must be manipulated by building a proper experimental learning environment and carrying out a long process of trial and error. This is our work's main contribution. Here, the block diagram shown in Figure 7 is exploited to illustrate the relationship between CARLA and our driving-control DRL models. Thanks to the architecture of such a simulation tool, for the different DRL learning algorithms, such as DDPG and RDPG, we just need to replace the source code in the DRL model with another one. Furthermore, CARLA can generate realistic sensed data and send them directly to the DRL model for the training procedure. After the training has finished, upon receipt of the required sensed data, the DRL model can continuously provide decision actions to CARLA for the purposes of emulating the process of crossing an unsignalized intersection autonomously.

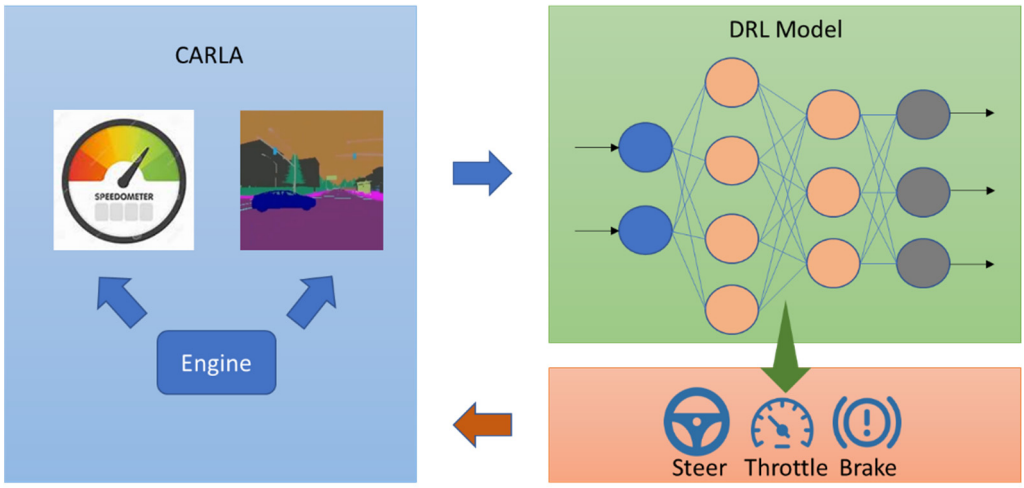


Figure 7. The relationship between CARLA and the DRL model.

3.1. The Reward Mechanism

The reward mechanism plays a key role in training the DRL model since it determines the convergence speed and the decision accuracy. Here, our reward mechanism is composed of six main constituents for accurate driving control, including velocity control and steering control. Their numeric values are derived according to the current situation of the self-driving car judged from sensed data of certain sensors, like the obstacle sensor, lane invasion detector and collision sensor, that are additionally installed on the car module. Note that the above sensors are only utilized for evaluating the reward obtained by performing the chosen action during the training procedure. Their sensed data are not actually fed into the DRL model. This means that we need not to really mount these sensors on a self-driving car for normal operations. Only the observation image of the road ahead captured by the semantic segmentation camera, installed beside the rearview mirror of the car, and the car speed are employed as the inputs of our DRL models for the training procedure and decision making in real applications. Through the reward mechanism, the DRL model will learn how to transform information obtained by sensors used for training into that obtained by sensors used for rewarding.

First of all, the concept of navigation with waypoints used in [35] is adopted to prevent the training car from deviating from the driving path properly arranged when it crosses the intersection. Before each behavior, a sequence of waypoints will be generated via navigation in advance, as seen in the example shown in Figure 8 for the going-straight behavior of crossing the intersection. The car will follow these waypoints to obtain a higher score and thus accomplish correct path keeping by designing an appropriate reward constituent $R_{waypoint}$ for this objective, expressed as follows:

$$R_{waypoint} = \left(T_{waypoint} - \left|P_{car} - P_{waypoint}\right|^2\right) \times w_{waypoint} \quad (6)$$

where $T_{waypoint}$ is the threshold for deciding whether the car deviates from the waypoint too much, P_{car} is the position of the car, $P_{waypoint}$ is the position of the next waypoint and waypoint is the regulation weight for the deviation distance. Specifically, because the car is required to pass through a waypoint as closely as possible, the method for the above equation to calculate the score is to measure the straight-line distance between the car and the next waypoint and then subtract the square of such a value from a predefined baseline to get the result.

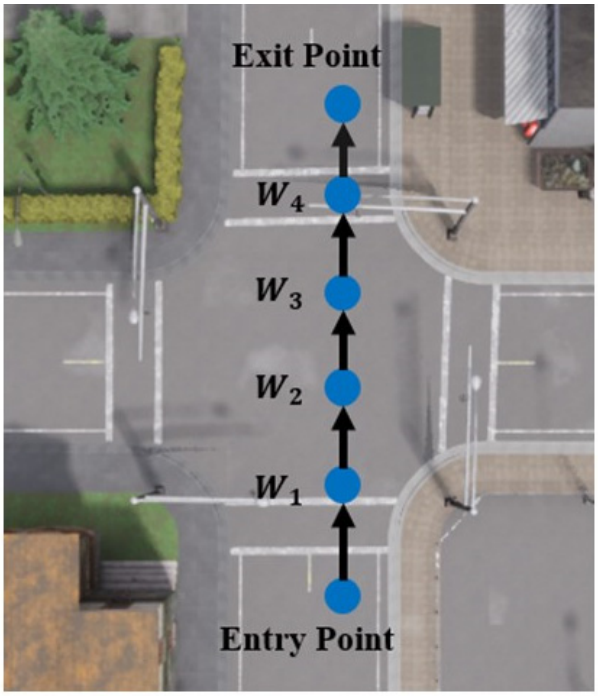


Figure 8. An example sequence of waypoints generated via navigation for the going-straight behavior of crossing the intersection.

Next, the self-driving car cannot run into any obstacle upon its driving. Hence, an obstacle sensor is mounted on the car for detecting the obstacle ahead on the driving path, like the building, vehicle, pedestrian and so on. Such a sensor can provide the distance to the obstacle detected and thus is utilized to calculate the score. If the safe distance is too short, the training car must be rewarded a negative score to ensure driving safety. The resultant equation for this new reward constituent $R_{obstacle}$ is expressed in the following:

$$R_{obstacle} = \begin{cases} C_{obstacle} \times w_{obstacle}, & D_{obstacle} < T_{obstacle} \\ -C_{obstacle} \times w_{obstacle}, & Otherwise \end{cases} \tag{7}$$

where $C_{obstacle}$ is a constant denoting the baseline magnitude of this constituent affecting the total reward, $w_{obstacle}$ is the regulation weight for $C_{obstacle}$, $T_{obstacle}$ is the threshold for deciding whether the car is keeping a safe distance from the obstacle ahead and $D_{obstacle}$ is the distance between the car and the detected obstacle.

Furthermore, the training car must keep running on the lane assigned to it in a way that it cannot deviate from its navigation route and move into another lane or even onto the sidewalk. To detect such a scenario of traffic regulations violation, a lane invasion detector is installed on the self-driving car, which can also assist the car in identifying the driving area so as to improve the control accuracy of the autonomous driving system and reduce the risk of accidents. The equation of this constituent R_{lane} is shown below:

$$R_{invasion} = \begin{cases} -C_{invasion} \times w_{invasion}, & Lane\ Invasion \\ 0, & otherwise \end{cases} \tag{8}$$

where $C_{invasion}$ is a constant denoting the baseline magnitude of the above constituent affecting the total reward and $w_{invasion}$ is the regulation weight for $C_{invasion}$.

On the other hand, another reward constituent $R_{collision}$ is used to indicate whether the self-driving car really collides with a car or an obstacle and should consequently be awarded a negative score to prevent such a scenario from occurring. To identify this

fatal problem, a collision sensor is installed on the car. The equation of this constituent is shown below:

$$R_{collision} = \begin{cases} -C_{collision} \times w_{collision}, & Obstacle\ Collision \\ 0, & otherwise \end{cases} \quad (9)$$

where $C_{collision}$ is a constant denoting the baseline magnitude of this new constituent affecting the total reward, and $w_{collision}$ is the regulation weight for $C_{collision}$.

The last two constituents concern the motion stability of the self-driving car, the purpose of which is to train the car to run more steadily, that is, without much deviation in its driving direction and velocity, so as to make the passengers within the car feel more comfortable. If the current change has not deviated much compared with the previous one, the reward score is higher; otherwise, it is lower. The equations of these two reward constituents R_{steer} and $R_{throttle}$ are expressed as below:

$$R_{steer} = (T_{steer} - |Sr_t - Sr_{t-1}|) \times w_{steer} \quad (10)$$

$$R_{throttle} = (T_{throttle} - |Th_t - Th_{t-1}|) \times w_{throttle} \quad (11)$$

In Equation (10), T_{steer} is the threshold for deciding whether the car has changed its direction suddenly, Sr_t and Sr_{t-1} separately denote the steering angles at times t and $t-1$ $(t > 0)$ and w_{steer} is the regulation weight for the fifth constituent. In Equation (11), $T_{throttle}$ is the threshold for deciding whether the car has changed its velocity abruptly, Th_t and Th_{t-1} denote the throttle values at times t and $t-1(t > 0)$ and $w_{throttle}$ is the regulation weight for the sixth constituent.

Eventually, the total reward R_{total} is constructed by summing the above six reward constituents, and thus its equation is expressed as follows:

$$R_{total} = R_{waypoint} + R_{obstacle} + R_{invasion} + R_{collision} + R_{steer} + R_{throttle} \quad (12)$$

3.2. System Architecture

This paper proposes a hybrid framework, as shown in Figure 9. It shows a stack of semantic segmentation images as input to a Convolutional Neural Network (CNN). In addition to the CNN, various measurements such as speed, lane invasion, collision, obstacles and waypoints are also input to a combined module. Such a module is then connected to a DDPG (Deep Deterministic Policy Gradient) or RDPG (Recurrent Deterministic Policy Gradient) agent. The agent processes these inputs and decides on actions such as steering angle, brake and throttle. The output of the agent's actions is then evaluated in the environment, resulting in a reward value that is fed back to the agent for updating future decisions. This setup illustrates a closed-loop system where the agent learns from the environment to improve its driving policy.

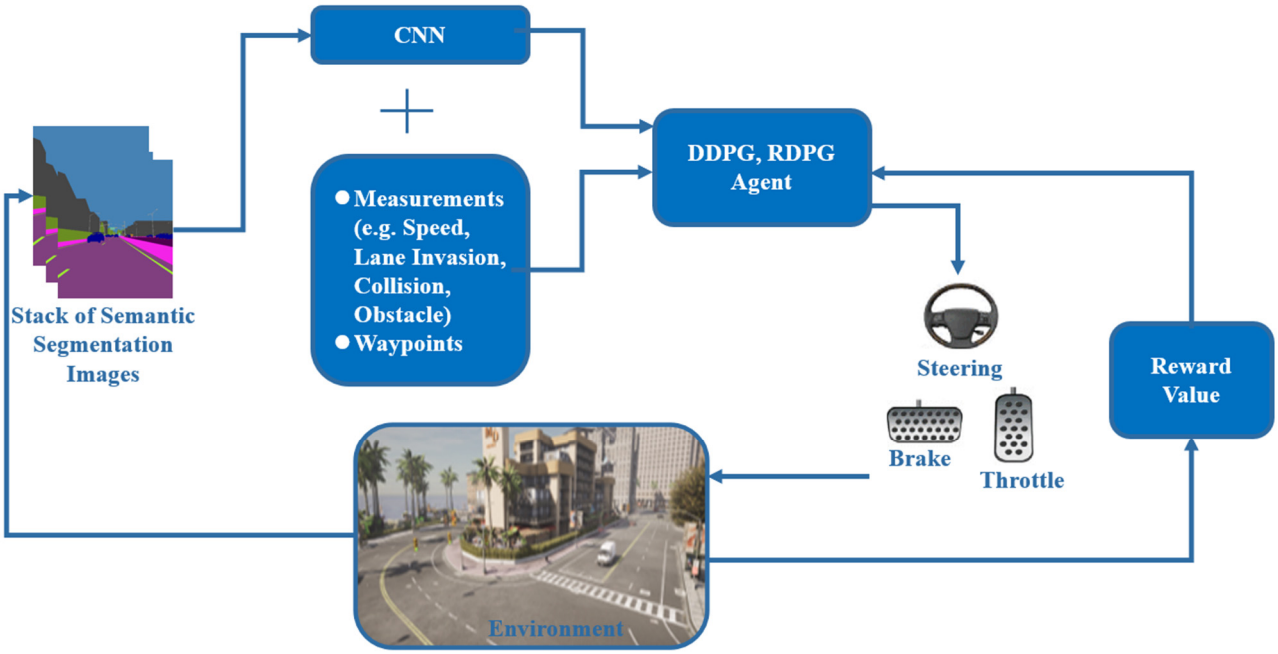


Figure 9. The system architecture of our hybrid framework.

3.3. Training Process

DDPG and RDPG differ only in their input dimensions and the structure of the hidden layers in their neural networks. The training stages are identical. More specifically, the steps of the training process are listed as follows:

1. First, the simulation environment is initialized. Multiple reference vehicles are generated at specified locations, and a training vehicle is placed at the starting point. A reference vehicle autonomously selects a path to enter the intersection, while the training vehicle must travel a short initial section to achieve a speed of over 20 km per hour. This is to prevent a low speed from triggering the environment's reset mechanism.
2. After obtaining the current environmental state (s_t), it is input into the two Actor networks of DDPG and RDPG. The Actor networks then generate an action output (a_t) based on the input state, including throttle, brake and steering control values. These actions are applied to the simulation environment, leading to the acquisition of the next state (s_{t+1}) and the cumulative reward (r_t). The collection of experiences (s_t, a_t, r_t, s_{t+1}) is stored in the Replay Buffer accordingly.
3. Once the Replay Buffer stores a set amount of training data, a batch of N experiences (s_t, a_t, r_t, s_{t+1}) is selected and fed into the training model for the update of the Actor and Critic networks.
4. To maintain the correct operation of the training vehicle, reset conditions are established. These include detecting unstable acceleration and deceleration behaviors of the vehicle, sensing collisions with other objects and whether the vehicle reaches the final goal. If any of these conditions are met, the training environment is reset and corresponding penalties are applied. In contrast, if the conditions are normal, the vehicle's driving actions continue to be controlled by the Online Actor network.

The overall training process flow is thus depicted in Figure 10.

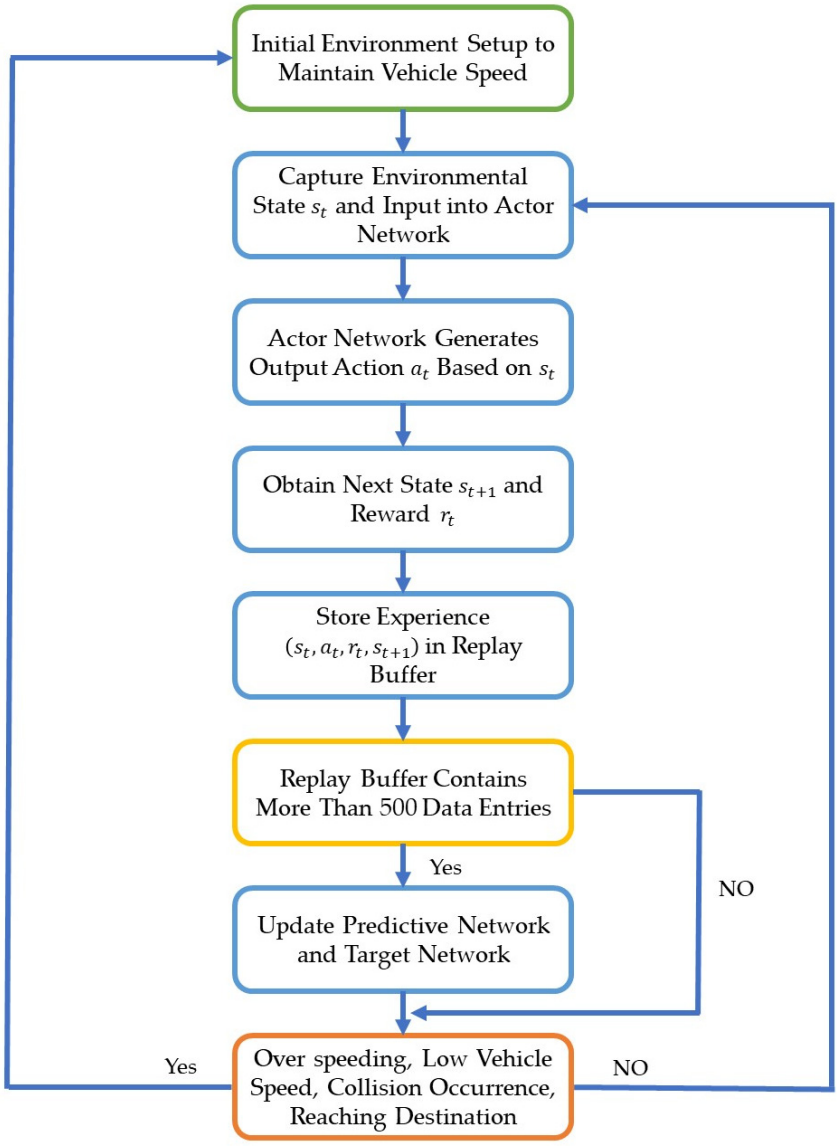


Figure 10. The flowchart of the DDPG and RDPG training processes.

4. Experimental Results and Discussion

4.1. Training Environment

The Town04 map of CARLA, as shown in Figure 11, is the environment adopted by us for the training of autonomous driving. We select an intersection on this map to serve as the training spot, which is a crossroad where two-way single-lane roads converge.

The map setup for the training spot is further detailed in Figure 12, which shows specific points of interest relevant to the training exercises. The starting point for the autonomous driving training is marked with a red dot on the map. This is the location from which the autonomous vehicle begins its navigation training. The endpoints, which are the goals the autonomous vehicle aims to reach during training, are marked with dots in different colors corresponding to different driving maneuvers. The endpoint for a right turn is indicated with a purple dot, the endpoint for continuing straight is marked with a yellow dot and the endpoint for a left turn is shown with a green dot. Each of these colored dots represents the final coordinates that the autonomous vehicle is trained to reach, depending on the control training it undergoes, i.e., the training is for making a right turn, going straight or making a left turn at the intersection.

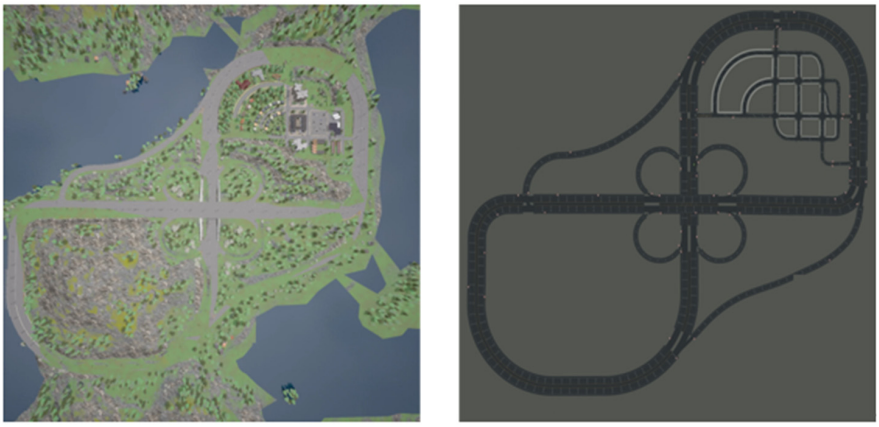


Figure 11. The top-down view of the Town04 map in CARLA.

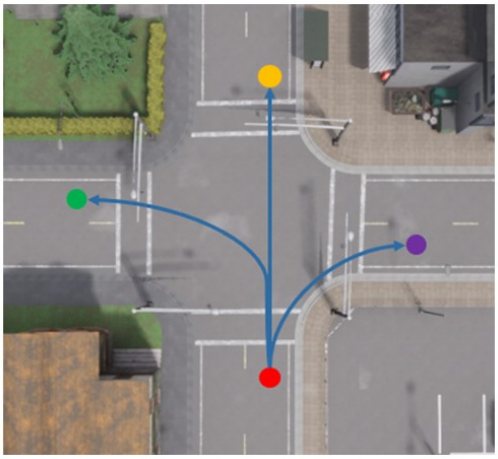


Figure 12. The setup for the autonomous driving training at the intersection.

This paper enhances the reality of intersection simulations by using the CARLA simulator's Traffic Manager to control reference vehicles. These vehicles operate autonomously, adhering to traffic rules, maintaining safe distances and observing speed limits. Unlike the main autonomous vehicle being trained, they are programmed to disregard traffic lights, as this study focuses on unsignalized intersection control. The reference vehicles navigate randomly chosen paths on the intersection to simulate the continuous traffic flow at the intersection, as the scenario shown in the Figure 13 below.

Figure 13. Random positioning and navigation of reference vehicles generated.

4.2. Hyperparameter Configuration

Adjusting hyperparameters is a crucial step in deep reinforcement learning, used to optimize the performance and generalization ability of the model. Hyperparameter values must be set manually before model training, as these parameters cannot be learned directly from the training. Therefore, experiments must be conducted to find the best combination of hyperparameters to improve model the effectiveness.

First, the configuration for the Experience Replay Buffer is shown in Table 1. Note that due to the difference in the input state dimensions between DDPG and RDPG, the capacity of the buffer set up for RDPG is smaller than that for DDPG. On the other hand, in the experiments with DDPG and RDPG, an Epsilon decay strategy is employed so that the proportion of the noise (OU Noise) added to the action can be adjusted over time. The initial value of Epsilon is set to 1, and after the Replay Buffer stores 500 pieces of data, the subsequent training process will decay Epsilon by 0.999 every 100 steps, until Epsilon decays to 0.01. The hyperparameter configuration used in this paper is listed in Table 2:

Table 1. Experience Replay Buffer configuration.

The Parameter Name	Value
DDPG Replay Buffer Size	16,000
DDPG Batch Size	150
DDPG Replay Buffer training threshold	500
RDPG Replay Buffer Size	6000
RDPG Batch Size	150
DDPG Replay Buffer training threshold	500

Table 2. Hyperparameter configuration.

The Parameter Name	Value
Learning rate α	Actor = 0.0001; Critic = 0.001
Discount factor γ	0.9
Tau (τ)	0.005
OU Noise: θ	All = 0.35
OU Noise: σ	Throttle = 0.1; Other = 0.2
OU Noise: μ	Throttle = 0.2; Other = 0
Epsilon (ε) start	1
Epsilon (ε) decay	0.99
Epsilon (ε) min	0.01

4.3. Training Results

In this thesis, two different algorithms, DDPG and RDPG, are used to train the autonomous driving control mechanism with the CARLA simulator. During the training process, multiple reference vehicles are generated to simulate the dynamic scenarios of real intersections. Moreover, the training car exploits the sensing modules to detect the environmental changes and explores the appropriate driving strategies to complete the task of crossing an unsignalized intersection. In particular, the focus of this research is to train the self-driving car to complete the three basic driving maneuvers: turning right, turning left and going straight through the intersection. In addition, by observing the convergence rate and cumulative rewards of different training models, we can understand the efficiency performance of the agents under the training of both models. The training results are visualized clearly by showing the cumulative rewards of the three maneuvers in Figure 14, as well as the convergence rate in Figure 15. Each chart corresponds to a distinct driving behavior, where the cumulative reward and loss convergence of the DDPG algorithm is shown in as orange line, while the results of the RDPG algorithm is shown as the blue line.

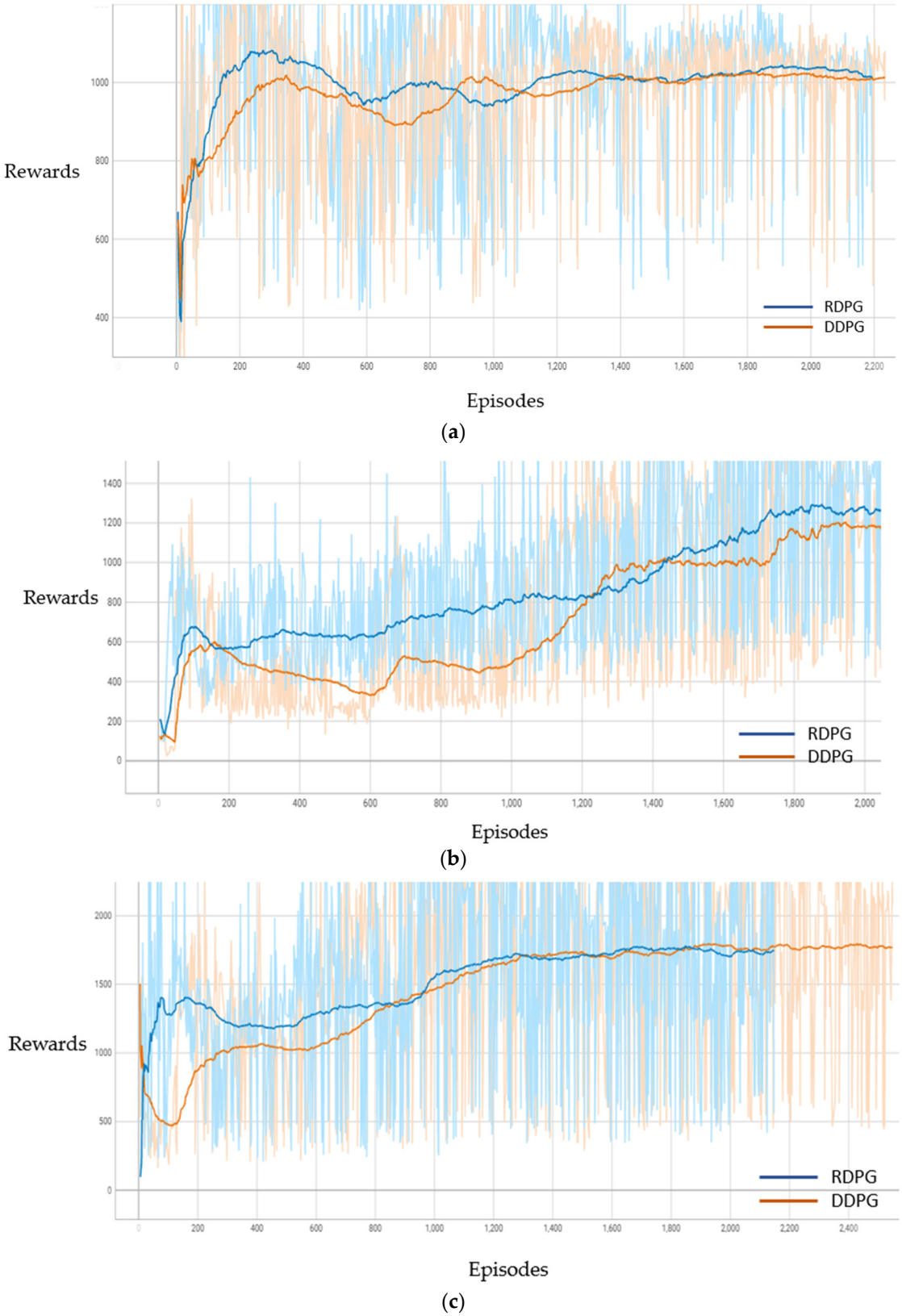
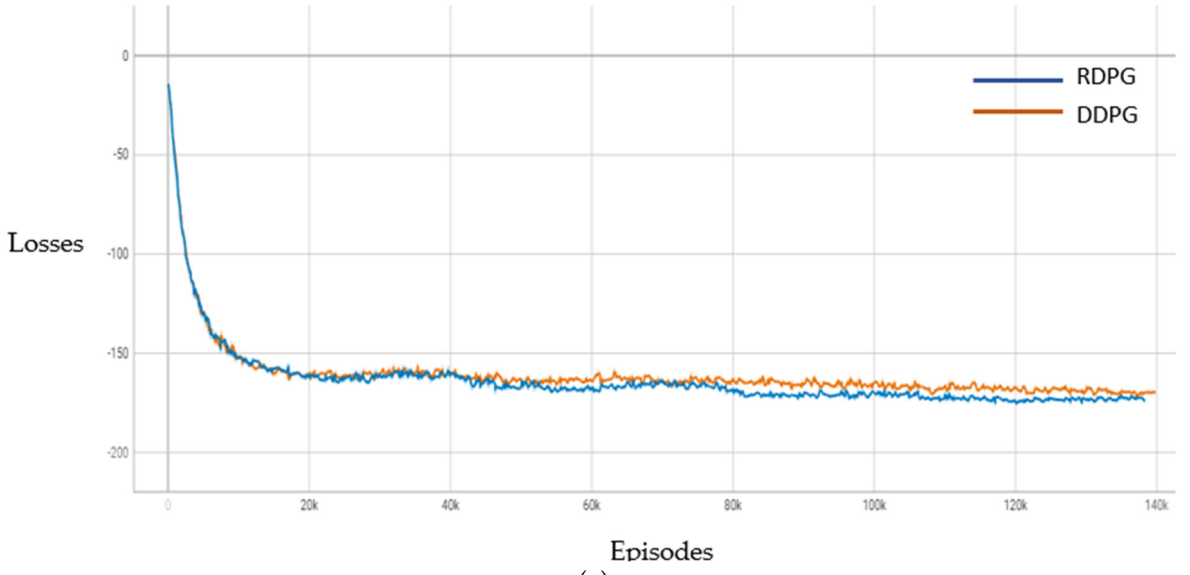
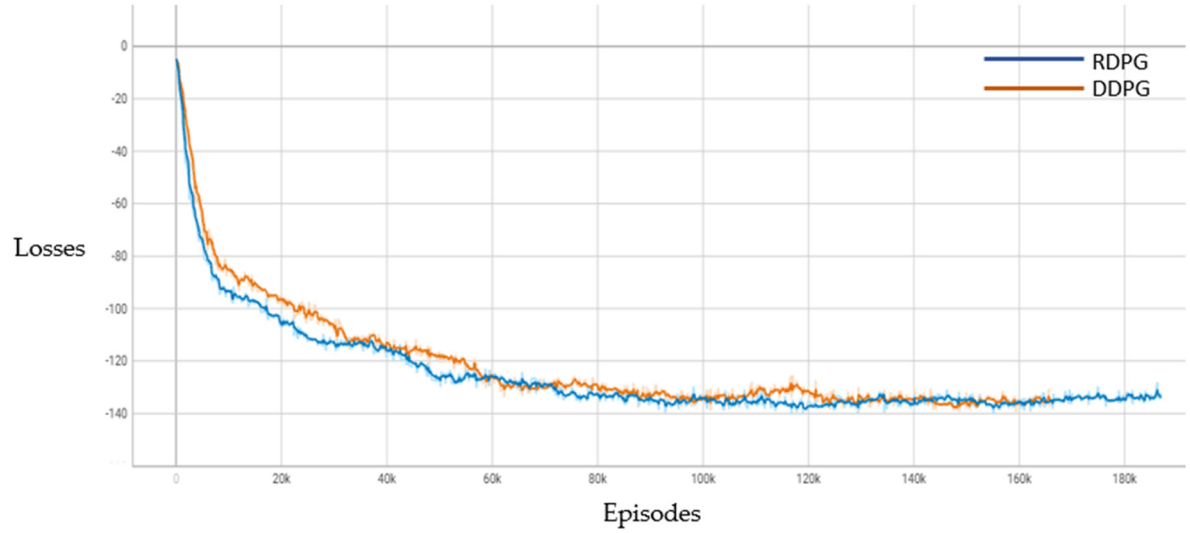


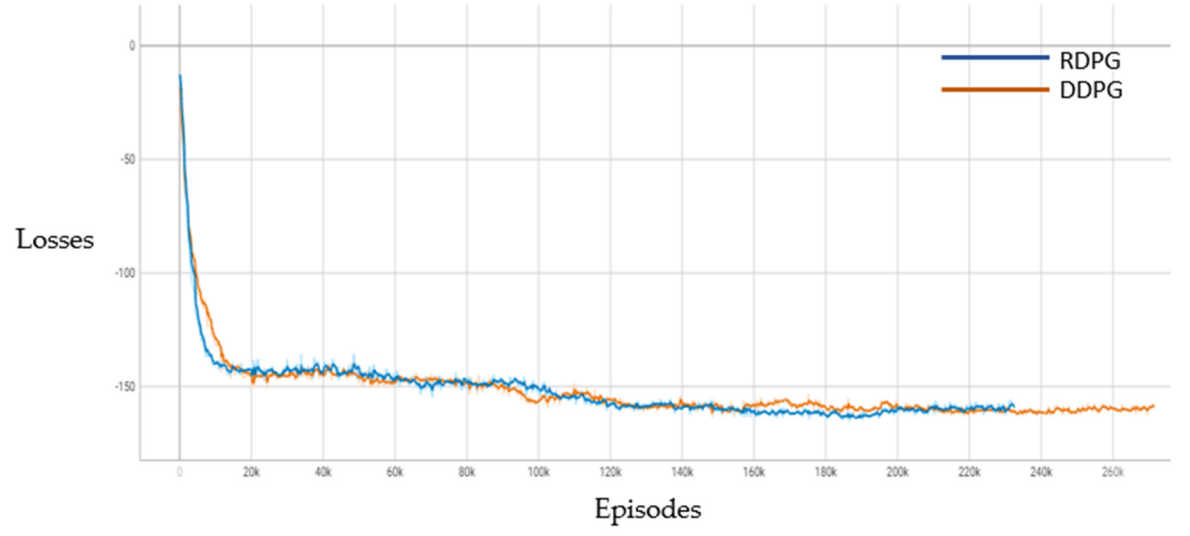
Figure 14. The cumulative rewards: (**a**) turning right; (**b**) going straight; (**c**) turning left.



(**a**)



(**b**)



(**c**)

Figure 15. The loss convergence: (**a**) turning right; (**b**) going straight; (**c**) turning left.

4.4. Discussion

According to Figure 14, we can find that the cumulative rewards obtained from the RDPG training for the three target actions are all superior to those from the DDPG training, especially for straight movements and left turns. Specifically, after 200 episodes, the rewards obtained from RDPG gradually increased, whereas DDPG still shows significant fluctuations, and the final maximum reward value obtained is also lower than that of the RDPG method. These differences in performance can be also reflected in the loss values of the two models shown in Figure 15. Although the two algorithms consistently exhibit good performance, it can be observed that the RDPG algorithm begins to converge for right turns after about 1300 training episodes and for straight movements around 1700 episodes, while the DDPG algorithm begins to converge for right turns and straight movements during much later episodes, specifically at the 1400th and 1900th episodes, respectively. This indicates that DDPG requires a larger number of episodes to stabilize, revealing slower learning efficiency. Moreover, from the training results, we can see that the convergence speed for learning a right turn is faster compared to the other two maneuvers. This is primarily because interferences from other vehicles upon making a right turn are much fewer than those encountered during the other two maneuvers. Finally, we can conclude that since the RDPG algorithm is capable of processing time-series data, it exhibits better performance compared to the DDPG algorithm.

5. Conclusions

In this paper, we introduced a hybrid architecture that combines the DDPG and RDPG algorithms with the perception module of a modular pipeline to address the driving task of traversing intersections without traffic signals. This includes driving maneuvers for right turns, going straight and left turns. Additionally, to avoid collisions with other dynamic vehicles and intrusions into prohibited pedestrian areas, a reward mechanism is elaborately designed to enable the DRL models to generate appropriate action control values for accurate and smooth autonomous driving.

Moreover, the experimental results show that the RDPG method outperforms the DDPG one. Such a performance promotion is mainly attributed to the incorporation of neural cell with the long-term memory capability, which significantly improves the model's ability to recognize environmental factors and then quickly formulate effective driving strategies under dynamic traffic conditions. Finally, the use of semantic segmentation technology combined with strategic reinforcement learning must have low-latency sensing and high-accuracy recognition capabilities for application in real-world autonomous vehicles. Future developments in this field are definitely expected to be able to meet these requirements of substantial computational power so as to really accomplish the methods presented in this study after fine tuning.

Author Contributions: Conceptualization, J.T. and Z.-Y.C.; methodology, J.T and Z.-Y.C.; software, Z.-Y.C.; validation, J.T., Y.-T.C., Z.-Y.C. and Z.Y.; formal analysis, J.T., Y.-T.C., Z.-Y.C. and Z.Y.; writing—original draft preparation, J.T. and Y.-T.C.; writing—review and editing, J.T. and Y.-T.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

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

References

1. Yurtsever, E.; Lambert, J.; Carballo, A.; Takeda, K. A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access* **2020**, *8*, 58 443–58 469. [CrossRef]
2. Road Traffic Safety Site. Available online: https://168.motc.gov.tw/ (accessed on 17 December 2023).
3. Bertozzi, M.; Broggi, A.; Cellario, M.; Fascioli, A.; Lombardi, P.; Porta, M. Artificial vision in road vehicles. *Proc. IEEE* **2002**, *90*, 1258–1271. [CrossRef]

4. Talebpour, A.; Mahmassani, H.S. Influence of connected and autonomous vehicles on traffic flow stability and throughput. *Transp. Res. Part C Emerg. Technol.* **2016**, *71*, 143–163. [CrossRef]
5. Paden, B.; Čáp, M.; Yong, S.Z.; Yershov, D.; Frazzoli, E. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Trans. Intell. Veh.* **2016**, *1*, 33–55. [CrossRef]
6. Schwarting, W.; Alonso-Mora, J.; Rus, D. Planning and decision-making for autonomous vehicles. *Annu. Rev. Control Robot. Auton. Syst.* **2018**, *1*, 187–210. [CrossRef]
7. Claussmann, L.; Revilloud, M.; Gruyer, D.; Glaser, S. A review of motion planning for highway autonomous driving. *IEEE Trans. Intell. Transp. Syst.* **2020**, *21*, 1826–1848. [CrossRef]
8. Iftikhar, S.; Zhang, Z.; Asim, M.; Muthanna, A. Deep learning-based pedestrian detection in autonomous vehicles: Substantial issues and challenges. *Electronics* **2022**, *11*, 3551. [CrossRef]
9. Chen, C.; Seff, A.; Kornhauser, A.; Xiao, J. DeepDriving: Learning affordance for direct perception in autonomous driving. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 2722–2730.
10. Iandola, F.; Moskewicz, M.; Karayev, S.; Girshick, R.; Darrell, T.; Keutzer, K. DenseNet: Implementing efficient ConvNet descriptor pyramids. *arXiv* **2014**, arXiv:1404.1869.
11. Espié, E. Torcs: The Open Racing Car Simulator. 2000. Available online: https://api.semanticscholar.org/CorpusID:16920486 (accessed on 25 May 2023).
12. Sauer, A.; Savinov, N.; Geiger, A. Conditional affordance learning for driving in urban environments. *arXiv* **2018**, arXiv:1806.06498.
13. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 1–9.
14. Chang, C.-C.; Tsai, J.; Lin, J.-H.; Ooi, Y.-M. Autonomous driving control using the DDPG and RDPG algorithms. *Appl. Sci.* **2021**, *11*, 10659. [CrossRef]
15. Wolf, P.; Hubschneider, C.; Weber, M.; Bauer, A.; Härtl, J.; Dürr, F.; Zöllner, J.M. Learning how to drive in a real world simulation with deep Q-Networks. In Proceedings of the 2017 IEEE Intelligent Vehicles Symposium (IV), Los Angeles, CA, USA, 11–14 June 2017; pp. 244–250.
16. Kendall, A.; Hawke, J.; Janz, D.; Mazur, P.; Reda, D.; Allen, J.M.; Lam, V.D.; Bewley, A.; Shah, A. Learning to drive in a day. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 8248–8254.
17. Agarwal, T.; Arora, H.; Schneider, J. Learning urban driving policies using deep reinforcement learning. In Proceedings of the 2021 IEEE International Intelligent Transportation Systems Conference (ITSC), Indianapolis, IN, USA, 19–22 September 2021; pp. 607–614.
18. Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; Koltun, V. CARLA: An open urban driving simulator. In Proceedings of the Conference on Robot Learning, California, MV, USA, 13–15 November 2017.
19. Shah, S.; Dey, D.; Lovett, C.; Kapoor, A. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics: Results of the 11th International Conference*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 621–635.
20. AWS Deep Racer. Available online: https://aws.amazon.com/jp/deepracer/ (accessed on 17 December 2023).
21. Liu, S.; Jia, J.; Fidler, S.; Urtasun, R. SGN: Sequential grouping networks for instance segmentation. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017; pp. 3496–3504.
22. Wojke, N.; Bewley, A.; Paulus, D. Simple online and realtime tracking with a deep association metric. In Proceedings of the 2017 IEEE International Conference on Image Processing (ICIP), Beijing, China, 17 September 2017; pp. 3645–3649.
23. Redmon, J.; Farhadi, A. YOLO9000: Better faster stronger. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 7263–7271.
24. Zhu, Z.; Liang, D.; Zhang, S.; Huang, X.; Li, B.; Hu, S. Traffic-sign detection and classification in the wild. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 2110–2118.
25. Wiering, M.; Otterlo, M. *Reinforcement Learning: State-of-the-Art*; Springer: Berlin, Germany, 2012.
26. Watkins, C.J.; Dayan, P. Technical note: Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]
27. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
28. Paternain, S.; Bazerque, J.A.; Small, A.; Ribeiro, A. Stochastic policy gradient ascent in reproducing kernel hilbert spaces. *IEEE Trans. Autom. Control* **2021**, *66*, 3429–3444. [CrossRef]
29. Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; Riedmiller, M. Deterministic policy gradient algorithms. In Proceedings of the 31st International Conference on Machine Learning (ICML-14), Beijing, China, 22–24 June 2014; pp. 387–395.
30. Bhatnagar, S.; Sutton, R.S.; Ghavamzadeh, M.; Lee, M. Natural actor critic algorithms. *Automatica* **2009**, *45*, 2471–2482. [CrossRef]
31. Jesus, J.C.; Bottega, J.A.; Bottega, J.A.; Cuadros, M.A.S.L.; Gamarra, D.F.T. Deep deterministic policy gradient for navigation of mobile robots in simulated environments. In Proceedings of the 2019 19th International Conference on Advanced Robotics (ICAR), Belo Horizonte, Brazil, 2–6 December 2019; pp. 362–367.
32. Li, X.; Liu, H.; Wang, X. Solve he inverted pendulum problem base on DQN algorithm. In Proceedings of the 2019 Chinese Control and Decision Conference (CCDC), Nanchang, China, 3–5 June 2019; pp. 5115–5120.
33. Heess, N.; Hunt, J.; Lillicrap, T.; Silver, D. Memory-based control with recurrent neural networks. *arXiv* **2015**, arXiv:1512.04455.

34. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
35. Tsai, J.; Chang, C.-C.; Li, T. Autonomous driving control based on the technique of semantic segmentation. *Sensors* **2023**, *23*, 895. [CrossRef] [PubMed]
36. Tsai, J.; Chang, Y.-T.; Chuang, P.-H.; You, Z. An autonomous vehicle-following technique for self-driving cars based on the semantic segmentation technique. In Proceedings of the 16th IEEE International Symposium on Robotic and Sensors Environments, Yokohama, Japan, 16–19 November 2023.