information

Article

Deep Reinforcement Learning for Autonomous Driving in Amazon Web Services DeepRacer

Bohdan Petryshyn, Serhii Postupaiev, Soufiane Ben Bari and Armantas Ostreika *

Department of Multimedia Engineering, Faculty of Informatics, Kaunas University of Technology, LT-51368 Kaunas, Lithuania; bohdan.petryshyn@ktu.edu (B.P.); serhii.postupaiev@ktu.edu (S.P.); soufiane.ben@ktu.edu (S.B.B.)

* Correspondence: armantas.ostreika@ktu.lt; Tel.: +370-614-21-727

Abstract: The development of autonomous driving models through reinforcement learning has gained significant traction. However, developing obstacle avoidance systems remains a challenge. Specifically, optimising path completion times while navigating obstacles is an underexplored research area. Amazon Web Services (AWS) DeepRacer emerges as a powerful infrastructure for engineering and analysing autonomous models, providing a robust foundation for addressing these complexities. This research investigates the feasibility of training end-to-end self-driving models focused on obstacle avoidance using reinforcement learning on the AWS DeepRacer autonomous race car platform. A comprehensive literature review of autonomous driving methodologies and machine learning model architectures is conducted, with a particular focus on object avoidance, followed by hands-on experimentation and the analysis of training data. Furthermore, the impact of sensor choice, reward function, action spaces, and training time on the autonomous obstacle avoidance task are compared. The results of the best configuration experiment demonstrate a significant improvement in obstacle avoidance performance compared to the baseline configuration, with a 95.8% decrease in collision rate, while taking about 79% less time to complete the trial circuit.

Keywords: reinforcement learning; autonomous driving; simulation; object avoidance; AWS DeepRacer

Citation: Petryshyn, B.; Postupaiev, S.; Ben Bari, S.; Ostreika, A. Deep Reinforcement Learning for Autonomous Driving in Amazon Web Services DeepRacer. *Information* **2024**, *15*, 113. https://doi.org/10.3390/info15020113

Academic Editors: Jianbo Li, Junjie Pang and Antonio Comi

Received: 9 January 2024
Revised: 4 February 2024
Accepted: 6 February 2024
Published: 15 February 2024

1. Introduction

The topic of self-driving vehicles has been the focus of the automobile industry in recent years, with numerous technological advances paving the way for its realisation. However, despite significant progress, the path to achieving fully autonomous self-driving cars remains fraught with challenges. Developing sophisticated object detection and avoidance algorithms and safeguarding the security and reliability of these systems are paramount hurdles that must be overcome. Nevertheless, and while still facing many challenges to overcome before full-scale adoption, the ongoing breakthroughs in artificial intelligence and related fields are propelling us towards full vehicle autonomy, offering a tantalising glimpse into a future where transportation is safer, more efficient, and revolutionised. One of the most compelling benefits of self-driving cars is the potential to significantly reduce traffic accidents. More than 90% of road accidents are attributed to some degree of human error, including distractions, impairment, and flawed decision making. As autonomous driving technology matures, we can expect a dramatic decline in accidents [1]. The end goal of autonomous driving research is to create an automated driving system (ADS) that operates independently, increasing passenger safety and driver efficiency while simultaneously reducing accidents caused by human error. The widespread adoption of ADSs is projected to generate annual social benefits of nearly USD800 billion by 2050 in the USA alone, resulting from reduced congestion, decreased road casualties, lower energy consumption, and increased productivity from reclaimed driving time [2]. Automobile manufacturers are increasingly recognising self-driving software as a crucial competitive advantage, validating the feasibility of vehicle autonomy with current or near-term technologies. Although

researchers and manufacturers can use different approaches to address the problem of self-driving, common practices have emerged. Traditionally, ADSs have divided the task of autonomous driving, addressing each subcomponent individually. However, more recently, end-to-end methods have emerged as an alternative to modular approaches, leveraging artificial intelligence to create holistic solutions [3].

The purpose of this paper is to explore the capabilities of the Amazon Web Services (AWS) DeepRacer autonomous race car platform. This platform serves as a testing ground for investigating the viability of training end-to-end self-driving models with reinforcement learning (RL), with a specific emphasis on obstacle detection and avoidance. The insights gained from hardware and software configurations, as well as fine-tuning processes, contribute to the broader goal of transitioning self-driving technology from simulations to real-life scenarios. And, while DeepRacer provides robust support for deploying trained models to physical cars for evaluation and testing purposes, successful deployment in real-world environments often necessitates a multifaceted approach. Researchers commonly employ calibration techniques [4,5] to ensure integration between sensors and actuators, domain randomisation [6–8] to simulate a diverse range of real-world conditions and challenges, fine-tune using real-world data [9], and learn features through a mixture of simulation and real data [10].

The paper contributes to the field by conducting hands-on experimentation and comparing the performance of several hardware and software configurations in training the AWS DeepRacer autonomous car to navigate an obstacle course in a simulated environment. On the hardware side, the paper proposes the use of a full sensor suite, consisting of a single-lens camera with a light detection and ranging (LiDAR) sensor mounted atop the DeepRacer car, in combination with an optimised action space, software-wise, to provide a wider range of control options. Additionally, a continuous reward function is presented that allows for a more fine-grained feedback mechanism and enables the agent to make more precise adjustments to its behaviour in response to its environment.

2. State of the Art Review

In the rapidly progressing domain of artificial intelligence and autonomous driving, the field of RL has emerged with multiple techniques that were validated in simulated environments. Furthermore, during recent years, there have been accomplishments in the delivery of simulators for training an autonomous car agent [11,12] in a simulated environment such as Unity [13] or AWS DeepRacer [14]. This chapter delves into and reviews existing methodologies and pays attention to approaches, merits, and limitations in simulated autonomous driving scenarios.

Path planning is one of the main areas of applying RL in autonomous driving. The main goal is to ensure safe and smooth navigation, avoiding collisions with parts of the environment. To handle path planning issues, conventional approaches are A-star search [15], Rapidly Exploring Random Trees, and D-star algorithms [16].

However, these algorithms face limitations in terms of speed, when applied to large-scale environments, making them unsuitable for real-time usage. Therefore, many studies are based on alternative approaches, such as deep learning. Most recent solutions use deep RL, as using deep neural networks in combination with RL algorithms allows us to create more robust solutions that can learn complex tasks. For example, a double Deep Q-Network (DQN) was applied, to perform dynamic path planning for unknown environments [17], and then another researcher enhanced the previous solution and proposed a novel motion planning algorithm based on the double DQN, which demonstrated excellent generalisation abilities in the simulated environment [18]. Then, a combined approach was introduced, in which the DQN was merged with a Long Short-Term Memory (LSTM) network. In this way, the LSTM network can learn and store state information at moments before and after specific time points [19].

In addition to the above, there is an increasing rate of RL applications made not only for motion planning but also for the domain of autonomous racing agents training to navigate

in simulated environments [20,21]. These methods often utilise Soft Actor Critic (SAC) algorithms that are designed to train a model for assessing the value of specific actions in a given situation. This knowledge is then used to guide the choice of actions. Another group of authors proposed a novel approach that was built on top of distributional RL, with its policy optimisation maximising stochastic outcomes [22]. Consequently, they obtained significantly better motion planning behaviour compared to traditional RL algorithms. Another team of researchers used ResNet-34 [23] as an actor and critic network architecture in the SAC algorithm, to increase the sustainability of the implemented policy [24].

In addition to planning routes, autonomous driving agents also face the complex challenge of avoiding stationary or dynamically placed obstacles in their way. The research team introduced a Convolutional Deep Deterministic Policy Gradient (CDD-PG) approach [25], which employs a network composed of depth-wise separable convolutional layers. This method is designed to efficiently process images for obstacle avoidance tasks, that utilise an actor–critic network structure. Another article [26] proposed Coordinated Convolution Multi-Reward Proximal Policy Optimisation (CCMR-PPO), which decreases the dimension of the bird's eye view data within the convolution network and then feeds the processed data to the algorithm input to optimise the state space. Experiments show that the designed approach suggested a notable gain in performance; compared to the Proximal Policy Optimisation (PPO) algorithm, the number of tasks completed increased by 54%. The research [27] conducted experiments to evaluate the PPO and SAC algorithms in terms of success rate, performance, and training speed; the experiments conducted showed that the algorithms are comparable and performed similarly with a 91% success rate.

Another popular avenue for the application of RL in autonomous driving is navigating the agent in dynamic traffic highway environments. Notably, one approach proposed an LSTM-based framework to predict the potential paths of several nearby vehicles within a specific proximity of the agent [28]. Experiment results indicated the effectiveness of the proposed model in predicting the forthcoming trajectories around the agent, accurately generating the corresponding spatio-temporal map across various dynamic scenarios. Another noteworthy solution addressed agent navigation in densely simulated traffic environments with various driver behaviours [29]. The authors presented a simulation approach that enriches existing traffic simulators by integrating trajectories enhanced with diverse behaviours, generated through a driver behaviour modelling algorithm. Utilising this enhanced simulator, they trained a deep RL policy, allowing the ego-vehicle to navigate through dense traffic. Other researchers developed the decision-making approach for autonomous vehicles operating in a multilane environment [30]. This method establishes a high-level policy for safe tactical decision-making, addressing challenges related to collision prevention and the consideration of unobservable states caused by unpredictable behaviours of other agents. Furthermore, it is worth mentioning another set of authors proposed the ReinforcementDriving method [31], which involves the exploration of navigation skills and trajectories of a simulator for comprehensive road maintenance. Given that traffic navigation is relatively new, the group of researchers introduced the framework that incorporates Recurrent Neural Networks (RNN) for effective information integration, enabling the vehicle to navigate partially observable scenarios [32]. Another promising approach to consider is imitation learning. It allowed the enhancement of trajectory tracking precision in autonomous driving, by teaching effective driving skills to an intelligent agent and then adopting RL to optimize the agent's driving policy [33].

In continuation with the remarkable findings discussed earlier, another salient topic of investigation centres around mixed traffic scenarios where human-driven vehicles and autonomous vehicles (AVs) coexist. To contribute to the described field, the authors aim to improve travel efficiency and minimize congestion at intersections, ultimately reducing the total travel time for AVs. As a result, the novel hierarchical multi-agent RL framework was proposed to simultaneously regulate traffic signals and rerouting directions of AVs in a dynamic manner [34].

The high amount of research in simulated environments can also be explained by the fact that training RL policies in the real world is difficult due to the high complexity of sampling and safety issues. Simulation mitigates these problems and serves as a testing ground for evaluating software and experimenting with algorithms.

However, the gap between simulation and reality, or the Sim-to-Real gap, downgrades the effectiveness of RL policies when they are applied to actual robots. A common method to bridge this Sim-to-Real transfer gap is domain randomisation. In domain randomisation, simulation parameters are varied during the training phase. This technique has been effectively used in Sim-to-Real transfers for numerous robotic tasks. In general, approaches involve introducing noise into dynamics [6] and imagery [35], employing model ensembles [36], incorporating adversarial noise [37], and evaluating simulation bias [38].

Another way to handle the Sim-to-Real transfer gap is domain adaptation, which is also widely used to mitigate the visual domain gap [39,40].

With the constant growth of the area of autonomous driving, it has become essential to have platforms where these Sim-to-Real techniques can be tested and refined. Several simulation platforms, including CARLA [41], AirSim [42], TORCS [43], and SUMO [44], significantly contribute to the development and evaluation of deep RL autonomous systems by offering diverse environments and features to simulate real-world scenarios.

CARLA focusses on the testing of autonomous vehicles in realistic city settings. For example, a team of researchers applied the DQN method and used the CARLA platform to emulate the motion of a self-driving vehicle within a simulation environment, which includes an obstacle vehicle [45]. Other authors also used the given simulator and collected extensive data on human drivers' responses to road obstacles to apply behaviour-cloning network architecture with the modified loss [46]. In that paper, they confirmed that the end-to-end model that incorporates imitation learning successfully navigates through an urban environment simulation. In addition to that, a motion planning strategy for autonomous vehicles grounded in motion prediction and vehicle-to-vehicle (V2V) communication was designed, and its robustness was guaranteed using the CARLA simulator [47].

AirSim offers physically and visually realistic simulations for the development and testing of algorithms for autonomous vehicles. With their help, the authors validated the effectiveness of an online federated RL transfer process for real-time knowledge extraction to address the problem of the complex and time-consuming process of knowledge localisation [48]. Another group of researchers enhanced cloud computing technology to reduce the training time of deep RL models for autonomous driving by allocating the training process among a pool of virtual machines [49] using AirSim environments.

TORCS and SUMO are demonstrated to be prominent tools for developing and testing short-term trajectory planning approaches [50], solutions that handle complex state and action spaces in a continuous domain [51], multi-agent deep reinforcement learning methods for self-driving vehicles capable of navigating through traffic networks with uncontrolled intersections [52], and many other advances or methodologies.

Building upon this diverse landscape of simulation platforms, AWS DeepRacer emerges as a decent tool in this domain, offering a practical and versatile environment for experimentation. DeepRacer provides a basis for replicating and experimenting with Sim-to-Real techniques. Using the DeepRacer car, navigation varies from basic, slow-speed lane-following to more intricate activities like high-speed racing or navigating through traffic. It utilises nonrecurrent network architectures for RL policy, allowing to obtain real-world robustness to various cars, tracks, and environmental changes.

Despite growing interest in DeepRacer, the amount of research on the platform is relatively limited. Most ideas and findings are often found in the form of tutorials, blog posts, and open-source projects, rather than in peer-reviewed scientific papers. However, some notable contributions were observed and reviewed in the rest of this chapter.

Most of the articles focused on creating solutions with the DeepRacer agent for obstacle-free environments [53]. Another paper presented a novel system framework for the Vehicle Network Autonomous Racing Model (VNARM) [54]. The vehicle's ability to complete a

lap improved drastically, reducing the time by up to three times compared to the baseline solutions [14], while still achieving a high completion rate.

There was also research oriented on multi-agent autonomous navigation on the DeepRacer platform [55,56]. This article focusses on vehicle platoon state control strategy and scheduling, using Gazebo simulations [57], and introduces novel algorithms for seamless state switching, including strategies for convoy formation, disbandment, and reordering, ensuring accurate navigation in varied scenarios such as overpass transits.

Another group of authors even conducted experiments to introduce a cost-effective DeepRacer simulation on an EC2 instance [58], bypassing the need for hardware and reducing costs compared to the DeepRacer Console.

However, there is almost no research aimed at handling obstacle avoidance using DeepRacer. Only one research study was found [59], where the authors integrated two pathfinding algorithms, A-Star and Line-of-Sight, into the paradigm of autonomous driving and showed that the models developed using these methods exceeded the default AWS DeepRacer approach in terms of both learning speed and overall race performance.

The review conducted of existing solutions indicates a significant focus on applying deep RL to autonomous driving solutions in simulated environments, with a growing trend of applying RL to autonomous racing agents and notable advancements in obstacle avoidance techniques. While DeepRacer serves as a perspective platform for experimentation, there is a notable lack of research in obstacle avoidance using the mentioned environment, pointing to potential areas for future exploration.

3. Materials and Methods

This chapter describes the proposed methods for implementing a machine learning model for autonomous driving in a simulated environment. The underlying optimization algorithms, hardware and software systems, and reward functions are going to be defined in this chapter as well.

3.1. Policy Network

The policy network is a deep neural network model, responsible for the actions taken by the agent [60]. The model takes environment data as the input and generates actions as the output. In this case, the agent is a self-driving car, the environment is the racing track, the input data are the video frames from the cameras, and the point cloud forms the LiDAR. The actions the model can output are combinations of velocity and steering angle that are defined by the action space before training.

The model architecture (Figure 1) is defined by the AWS DeepRacer service and consists of multiple 2D convolutional neural network (CNN) image embedders for video frames processing, a 1D LiDAR input embedder, and a dense layer responsible for selecting actions based on the features extracted from the environment.

The first step is the input processing. The goal of this step is to extract features from the input sensor signals. The 2D CNNs are responsible for feature extraction from the front-facing cameras. The inputs from each camera are processed separately. The one-dimensional input of the LiDAR is processed by a 1D CNN.

The second step in the processing pipeline is to choose an action based on the features extracted from the inputs captured by the sensors. A neural network of fully connected layers is responsible for this. The network takes the combined output of the feature extraction networks as input and produces an action as output. The size of the output layer is defined by the action space size and type (discrete or continuous). In the case of a discrete action space, the output layer will have the same number of nodes as the number of actions in the action space. In the case of a continuous action space, the output layer will have two nodes producing continuous values: one for the velocity and one for the steering angle.

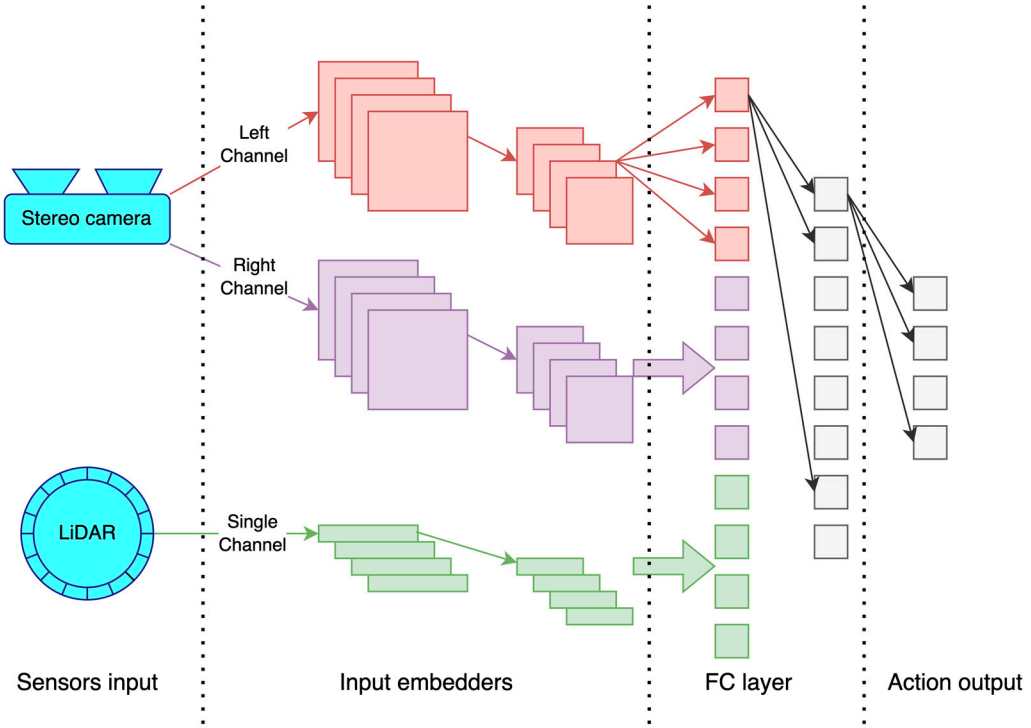

Figure 1. AWS DeepRacer policy network architecture.

3.2. Hardware and Sensors

The physical car AWS DeepRacer Evo can be equipped with different combinations of sensors: a single camera, a single camera with a LiDAR, a stereo camera, or a stereo camera with a LiDAR [61].

A forward-pointing monocular camera provides the agent with visual information on what is in front of the car. This information should be enough for the car to race without going off track. An agent with a single camera requires the least training time, since the policy network has the smallest input size. On the other hand, this configuration is known for demonstrating suboptimal results in obstacle avoidance and head-to-head racing modes, since it lacks information about the distances to the detected objects on track.

A forward-pointing stereo camera provides the agent with two images taken from slightly different perspectives. This enables the policy network to estimate the distances to the visible objects and show better results in obstacle avoidance and head-to-head racing. With this configuration, the model takes significantly more time to converge, but has the potential to achieve higher performance.

Both single and stereo camera configurations are limited to providing the agent with the information from the forward direction. A LiDAR can be added to increase the agent's awareness of the situation in other directions. This helps the car to perform manoeuvres such as lane changes or turns without crashing into an obstacle or another car that might be to the side and not visible to the front-facing cameras. The layout of the sensors on a Deep Racer model can be seen in Figure 2.

The AWS DeepRacer simulation environment was designed to closely emulate the specifications of a physical DeepRacer EVO car and minimise the Sim-to-Real performance gap.

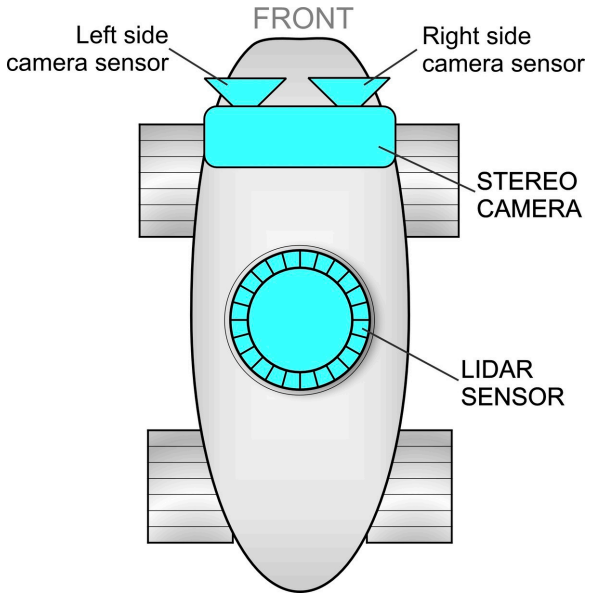

Figure 2. Sensors on the AWS DeepRacer EVO car [62].

3.3. Policy Optimization Algorithms

The AWS DeepRacer simulation platform provides tools and implements the reinforcement learning pipeline to train the policy network.

This section describes two policy optimisation algorithms available in the AWS DeepRacer simulation environment: PPO and SAC. The algorithms are used to optimise the policy network described earlier to perform actions with the highest possible reward according to the reward function, which is going to be defined later.

3.3.1. Proximal Policy Optimisation Algorithm

PPO is a policy-based algorithm capable of optimising policies with discrete and continuous action spaces [62]. The algorithm strives to make the optimisation of the policy stable. It is considered a successor to the Trust Region Policy Optimisation (TRPO) algorithm [63], which pursues the same goal, but is much more complex. The PPO algorithm has been proven to perform at least as well as TRPO, while being much simpler to implement.

At each optimisation step, the PPO algorithm tries to take the largest possible optimisation step without stepping too far to avoid causing optimisation collapse. The optimized policy parameters θ_{k+1} can be defined as follows:

$$\theta_{k+1} = \arg\max_{\theta} \underset{s,a\sim\pi_{\theta_k}}{\mathrm{E}} [L(s, a, \theta_k, \theta)] \tag{1}$$

where π_{θ_k} is the policy at the previous step and k, s, and a are the states and actions defined in the state and action spaces of the environment. The optimized policy is a policy that maximizes the expected value of $L(s, a, \theta_k, \theta)$ which is defined as follows [56]:

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_k}(a \mid s)} A^{\pi_{\theta_k}}(s, a), \mathrm{clip}\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_k}(a \mid s)}, 1-\epsilon, 1+\epsilon\right) A^{\pi_{\theta_k}}(s, a)\right) \tag{2}$$

where $\pi_\theta(a \mid s)$ is the probability of taking an action a in state s according to policy θ, $A^{\pi_{\theta_k}}(s, a)$ is a quantitatively expressed advantage of taking an action a in state s comparing to a baseline, and ϵ is a hyper parameter of the algorithm responsible for clipping the optimization step in the range from $1-\epsilon$ to $1+\epsilon$.

3.3.2. Soft Actor Critic Algorithm

SAC is an advanced off-policy algorithm designed to work with continuous action spaces [64]. The off-policy model makes the algorithm more data-efficient and improves sample efficiency. The algorithm incorporates an actor–critic architecture and an entropy regularisation technique that improves the stability of the training and the exploration capabilities.

SAC uses multiple value critics to estimate the state-action value function. This helps to reduce the variance in value estimates and contributes to more stable and accurate value function updates. The end value estimate $Q(s, a)$ is then given by an ensemble of critics, as follows:

$$Q(s,a) = \frac{1}{n}\sum_{i=1}^{n} Q_{\phi_i}(s,a) \tag{3}$$

The algorithm incorporates a regularisation of the entropy into the policy optimisation process. This improves the exploration abilities of the training process, preventing early convergence to suboptimal solutions. The entropy term is included in the training objective, which balances the trade-off between exploration and exploitation. The policy should, in each state, act to maximise the expected future return plus the expected future entropy [58], as follows:

$$\theta_{k+1} = \arg\max_{\theta} \mathop{\mathrm{E}}_{\tau\sim\pi_{\theta_k}}\left[\sum_{t=0}^{\infty}\gamma^t\left(R(s_t,a_t,s_{t+1}) + \alpha H\left(\pi_{\theta_k}(\cdot \mid s_t)\right)\right)\right] \tag{4}$$

where π_{θ_k} is the policy at the previous step k, $\alpha > 0$ is the trade-off coefficient, $H\left(\pi_{\theta_k}(\cdot \mid s_t)\right)$ is the entropy of the policy given the current state s_t, τ is the trajectory the agent experienced, γ^t is the discount factor at time t, and $R(s_t, a_t, s_{t+1})$ is the reward obtained by taking an action a_t in state s_t and transitioning to state s_{t+1}.

3.4. Reward Functions

In reinforcement learning, the reward function is responsible for providing quantitative feedback to the agent based on the actions performed. The reward function is at the core of the training process, as it defines the desired behaviour of the agent. In the case of a self-driving car in an obstacle avoidance scenario, the function must evaluate the following key behaviours of the agent: progressing through the racing track, staying inside the track, and staying away from the obstacles. This section describes the reward functions evaluated and analysed in this article.

3.4.1. Baseline Reward Function

The baseline reward function was taken from the AWS example collection [65]. This reward function rewards the agent for staying inside the track's borders and penalises it for getting too close to objects in front of it. The agent can move from lane to lane to avoid accidents. The total reward is a weighted sum of the reward and penalty. The function gives more weight to the penalty in an effort to avoid crashes. The function's pseudocode can be seen in Algorithm 1:

Algorithm 1: Baseline reward function

Input: reward function parameters
Result: reward
Initialize: *reward* = 1×10^{-3} *reward lane* = 1 *reward avoid* = 1
if car distance to the edge of the road < 0.05 m
    reward lane = 1
else
    reward lane = 1×10^{-3}
end if

Algorithm 1: *Cont.*

```
9:  if obstacle on the same lane as car
10:     if 0.5 ≤ distance to obstacle < 0.8
11:         reward avoid × = 0.5
12:     else if 0.3 ≤ distance to obstacle < 0.5
13:         reward avoid × = 0.2
14:     else if distance to obstacle < 0.3
15:         reward avoid = 1 × 10^−3
16:     end if
17: end if
18: reward = reward lane × 1 + reward avoid × 4
```

3.4.2. Extended Baseline Reward Function

After analysing the training and evaluation results (see Section 4.2) of the previous reward function, it can be concluded that the trained agent manages to stay on track well. It also avoids obstacles that are in front and on the same lane. Most crashes happen when the car needs to take a sharp turn and crashes into an obstacle from the side.

The results can be explained by the fact that the baseline reward function only handles the case where the car approaches an obstacle on the same lane and ignores the case where the car moves straight into an obstacle from a different lane.

This version of the reward function is based on the baseline version, but it punishes the agent not only when it approaches an obstacle on the same lane, but it also discourages it from being too close to an object on a different lane. According to the dimensions of an obstacle and the track published by AWS in the DeepRacer documentation [66], a circle with 0.4 m in radius around the centre of the box should be declared as a no-go zone for the agent. Entering an additional concentric circle with a radius of 0.5 m will be punished with a smaller decrease in reward so that the car can enter it when it is needed to make a turn on the track. The pseudocode of this function can be seen in Algorithm 2.

Algorithm 2: Extended baseline reward function

```
1:  Input: reward function parameters
2:  Result: reward
3:  Initialize: reward = 1 × 10^−3 reward lane = 1 reward avoid = 1
4:  if car distance to the edge of the road < 0.05 m
5:      reward lane = 1
6:  else
7:      reward lane = 1 × 10^−3
8:  end if
9:  if obstacle on the same lane as car
10:     if 0.5 ≤ distance to obstacle < 0.8
11:         reward avoid × = 0.5
12:     else if 0.3 ≤ distance to obstacle < 0.5
13:         reward avoid × = 0.2
14:     else if distance to obstacle < 0.3
15:         reward avoid = 1 × 10^−3
16:     end if
17: else
18:     if 0.4 ≤ distance to obstacle < 0.5
19:         reward avoid × = 0.5
20:     else if distance to obstacle < 0.4
21:         reward avoid = 1 × 10^−3
22:     end if
23: end if
24: reward = reward lane × 1 + reward avoid × 4
```

3.4.3. Continuous Reward Function

Another observation that can be captured from the experiments with the baseline and the extended baseline reward functions (see Sections 4.2 and 4.4) is that, when approaching an obstacle and entering a punishment zone, the agent does not do anything to exit the zone.

It can be assumed that such results are caused by the discrete nature of the baseline reward function. As the rewards returned by the function do not change gradually, the agent receives no guidance on what actions to take to immediately decrease the punishment. The PPO algorithm, which is used for policy optimisation, is based on gradient descent, which requires the rewards to be differentiable.

A possible solution to this problem could be to re-implement the reward function to return continuous reward values in the punishment zones. As the reward gradually decreases as the agent approaches an obstacle or an edge of the track, there will always be an action to take to stop being punished.

In this version of the reward function, it was also decided to remove the separation by lanes to simplify the solution and introduce less confusion during training. Each obstacle will have three concentric zones around it: a crash zone, a close zone, and a safe zone. In the crash zone, the minimal constant reward will be returned as the agent does not have an option to recover from entering this zone. In the close zone, a continuously decreasing reward will be returned as the agent gets closer to the centre. In the safe zone, no punishment will be applied, and the agent will receive the full reward. This will ensure that the agent receives guidance to leave the close zone and its behaviour is not affected by far-away obstacles. The pseudocode can be seen in Algorithm 3.

Algorithm 3: Continuous reward function

Input: reward function parameters
Result: reward
Initialize: *reward* = 1×10^{-3} *reward lane* = 1 *reward avoid* = 1
if distance from center is <0.35 × track width
    reward lane = 1
else if distance from center is <0.5 × track width
    reward lane = 3.33 × track width − 6.66 × track width × distance from center
else
reward lane = 1×10^{-3}
end if
if distance to closest obstacle < 0.25
    reward avoid = 1×10^{-3}
else if 0.25 ≤ distance to obstacle < 0.5
    reward avoid = (distance to obstacle − 0.25) × 4 + 1×10^{-3}
else
    reward avoid = 1
end if
reward = *reward lane* × 1 + *reward avoid* × 2

4. Results

4.1. Experimentation Settings

The evaluation of the experiments conducted was undertaken by comparing the DeepRacer agent simulation evaluation output. Given the limited existing research on obstacle avoidance, it was decided to focus the experiments on that area to better generalise to real-world tasks.

As there are two reinforcement learning algorithms available in the DeepRacer environment (PPO and SAC), the experiments were carried out using the default algorithms. Emphasis was placed on producing multiple reward functions by handling the limitations of the default ones. In addition, available sensors, environment configurations, training time, and hyperparameters were considered.

The research conducted can be divided into two groups, delineated by experiments involving reward functions and experiments with sensors (Table 1).

Table 1. Experiments plan.

Experiment No.	Reward Function	Sensors	Algorithm
1	Baseline	Stereo camera	PPO
2	Baseline	Stereo camera	SAC
3	Extended baseline	Stereo camera	PPO
4	Extended baseline	Single camera, LiDAR	PPO
5	Continuous reward function	Stereo camera	PPO
6	Continuous reward function	Single camera, LiDAR	PPO
7	Continuous reward function	Single camera, LiDAR	PPO, Reduced Action Space

All models were trained using the obstacle avoidance race type. In this race type, a vehicle races on a two-lane track, where three objects are randomly distributed across two lanes along the track at the beginning of each episode (Figure 3). All models were trained for an equal time period of three hours.

Then, the resulting models were evaluated on the same virtual track with three fixed obstacles located at 25%, 50%, and 75% of the track length (Figure 3).

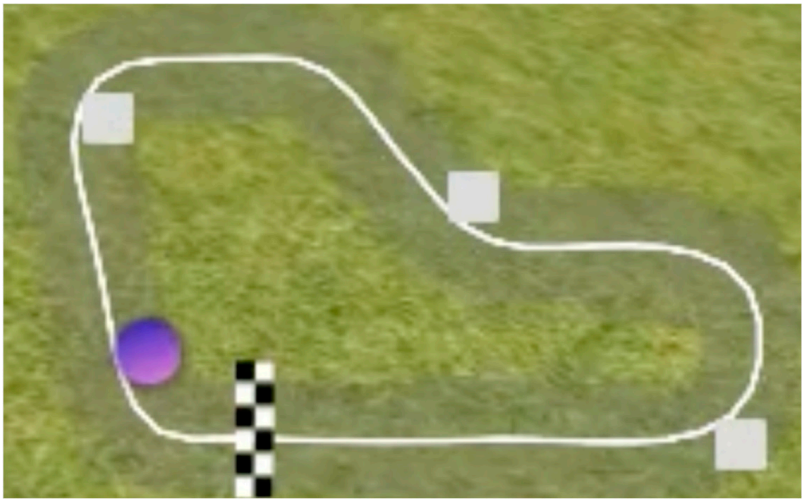

Figure 3. Simulation track map; the DeepRacer car represented by a purple circle; the three obstacles represented by white boxes.

4.2. Baseline Model

To start the experimentation, the AWS DeepRacer baseline reward function was selected. The setup involved the function itself, the PPO algorithm, stereo camera sensors, and used a discrete action space (Table 2). The model was trained for three hours.

The PPO algorithm was configured with the default set of hyperparameters (Table 3).

The results obtained were satisfactory to a certain degree, considering the training time and selected sensors. The reward graph depicted several extreme values during track completion in the evaluation (Figure 4). However, the evaluation data highlighted that the model stopped showing gradual improvement through the iterations, as the average percentage of completion in the training process stopped progressing after the 17th iteration.

Table 2. Baseline action space.

Action No.	Steering Angle (°)	Speed (m/s)
1	−30.0	0.50
2	−30.0	1.00
3	−15.0	0.50
4	−15.0	1.00
5	0.0	0.50
6	0.0	1.00
7	15.0	0.50
8	15.0	1.00
9	30.0	0.50
10	30.0	1.00

Table 3. Baseline PPO hyperparameters.

Hyperparameter	Value
Gradient descent batch size	64
Entropy	0.01
Discount factor	0.999
Loss type	Huber
Learning rate	0.0003
Number of experience episodes between each policy-updating iteration	20
Number of epochs	10

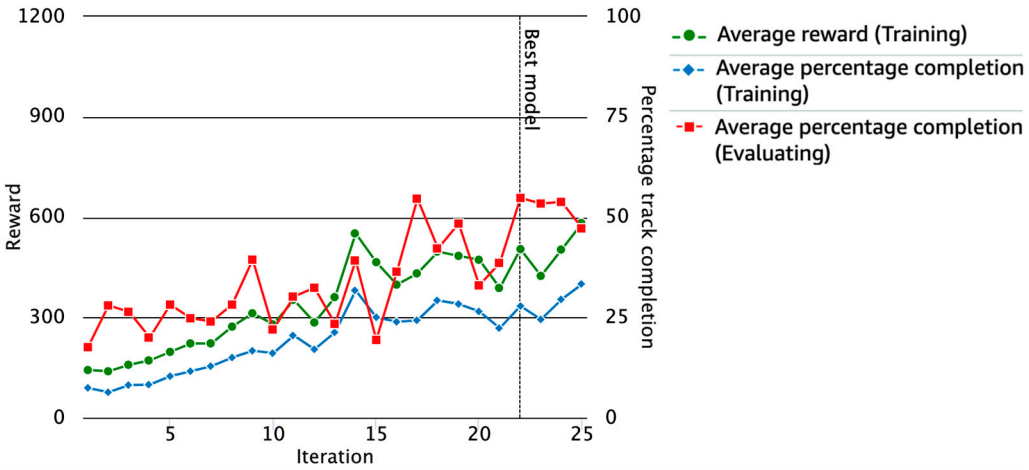

Figure 4. Baseline model reward graph.

The evaluation results were suboptimal, primarily due to the use of the baseline function of the environment, as it only addressed the scenarios when the agent faced an obstacle in the same lane and ignored the case when the car moved directly into an obstacle from the other lane. The agent completed the race with a total of 48 crashes and a race time of approximately 6:30 min (Table 4).

Table 4. Baseline model evaluation results.

Trial	Time (MM:SS.mmm)	Trial Results	Off-Track	Off-Track Penalty	Crashes	Crash Penalty
1	03:17.583	100%	0	--	26	130 s
2	02:09.535	100%	0	--	16	80 s
3	01:04.011	100%	0	--	6	30 s

Based on the evaluation analysis, the direction for the improvement of the reward function was defined. The agent failed to avoid obstacles in the opposite lane, turning

right into obstructions instead of avoiding them. These problems were addressed in the following experiments, to encourage the agent to complete the track without crashing into obstacles.

4.3. Baseline Model with Soft Actor Critic

Before continuing the experiments to improve the reward function, an experiment with the SAC algorithm was conducted to check the agent's behaviour with another RL algorithm using the baseline reward function. The default set of hyperparameters was used (Table 5) for the given experiment. The model was trained for three hours. The action space was continuous, as the SAC algorithm can support only such.

Table 5. Baseline SAC hyper parameters.

Hyperparameter	Value
Gradient descent batch size	64
Learning rate	0.0003
SAC alpha (α) value	0.2
Discount factor	0.999
Loss type	Mean squared error
Number of experience episodes between each policy-updating iteration	1

Judging from the reward graph (Figure 5), the trained model was unstable, and iterations contained a lot of extreme values for both training and evaluation. As a result, the agent was unable to complete the race in the maximum allotted time. The reason may be due to the fact that the SAC algorithm requires much more time to converge as it uses a continuous action space. After this experiment, the PPO algorithm and a discrete action space were used to focus more on the improvements of the reward function, to achieve meaningful results while keeping the training time constant.

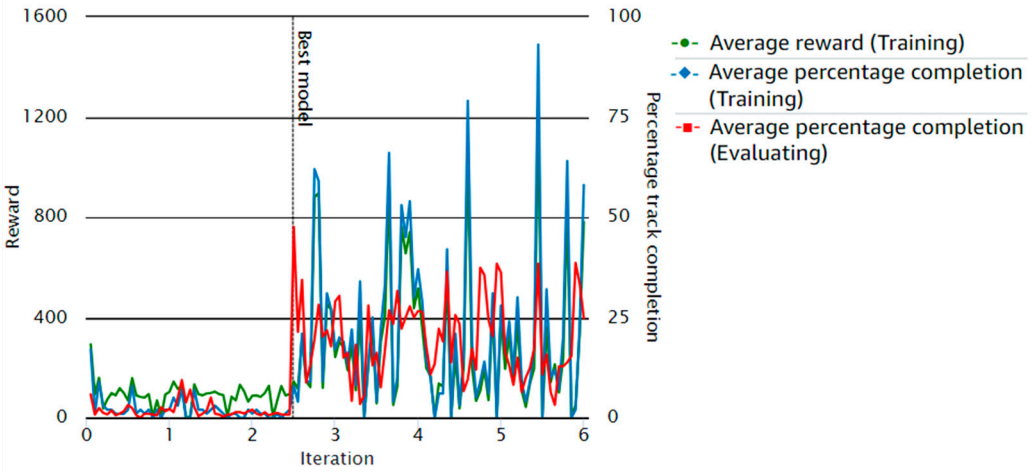

Figure 5. Baseline model with SAC algorithm reward graph.

4.4. Extended Baseline Model

Based on the revealed drawbacks and limitations of the baseline solution, the default reward function was reworked, to address the cases where the agent turned right into an obstacle located in the adjacent track lane. The improved function encourages the agent to avoid the obstacle in the mentioned case by penalising it not just for approaching an obstacle on the same lane, but also for approaching an object in the other lane. The model was trained for three hours with stereo camera sensors, using the same discrete action space (Table 2) and set of hyperparameters (Table 3) as in the baseline experiment.

Judging from the reward graph (Figure 6), the model demonstrated stabilisation after the 10th iteration, characterised by a decrease in extreme values compared to the evaluation of the previous experiment.

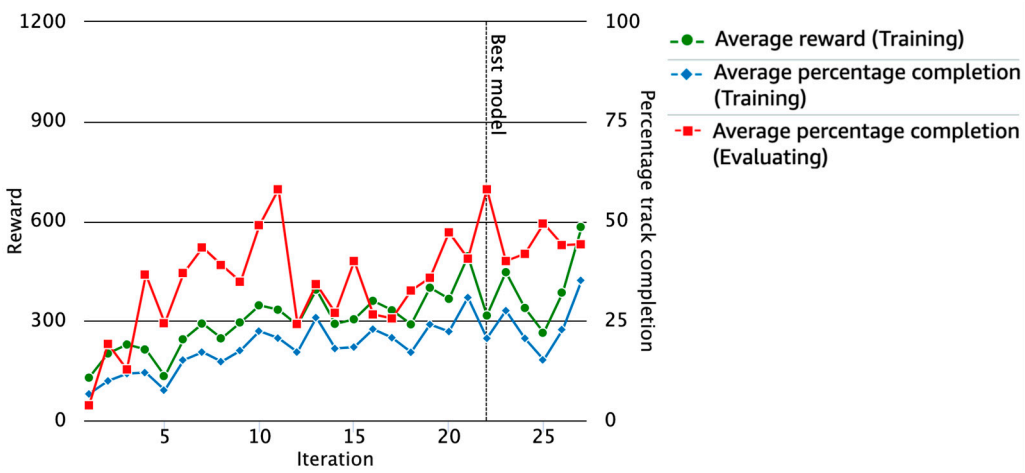

Figure 6. Extended baseline model reward graph.

The extended reward function led to a notable gain in accuracy while running the evaluation, resulting in 15 crashes, one off-track penalty, and 2:40 min of race time (Table 6). However, while watching the simulation stream, some failure cases were still detected, particularly when the agent turned directly into the obstacle and crashed.

Table 6. Evaluation results of the extended baseline model.

Trial	Time (MM:SS.mmm)	Trial Results	Off-Track	Off-Track Penalty	Crashes	Crash Penalty
1	00:46.98	100%	0	--	4	20 s
2	01:02.74	100%	1	2 s	6	30 s
3	00:52.74	100%	0	--	5	25 s

Moreover, most crashes were in scenarios where the obstacle was outside the view area of the camera. On the basis of the conducted experiment, it was concluded that the model might experience a lack of data from sensors to navigate the environment effectively.

4.5. Extended Baseline with Light Detection and Ranging (LiDAR) Model

The experiments carried out previously showed that the trained agent kept colliding with obstacles when they were not captured by the front-facing camera. To address this issue, a LiDAR sensor was added, which should be capable of detecting obstacles even if they are located on the side of the agent, as it uses light in the form of a pulsed laser to measure ranges.

Training the model with LiDAR yielded better results, based on the reward graph (Figure 7). The average percentage completion series contained several outliers, but certain gradual result improvements were observed.

The evaluation analysis also noted a slight improvement compared to the results of the previous experiment; the total number of crashes decreased, and the race was finished in 2:25 min with 10 crashes and one off-track penalty (Table 7).

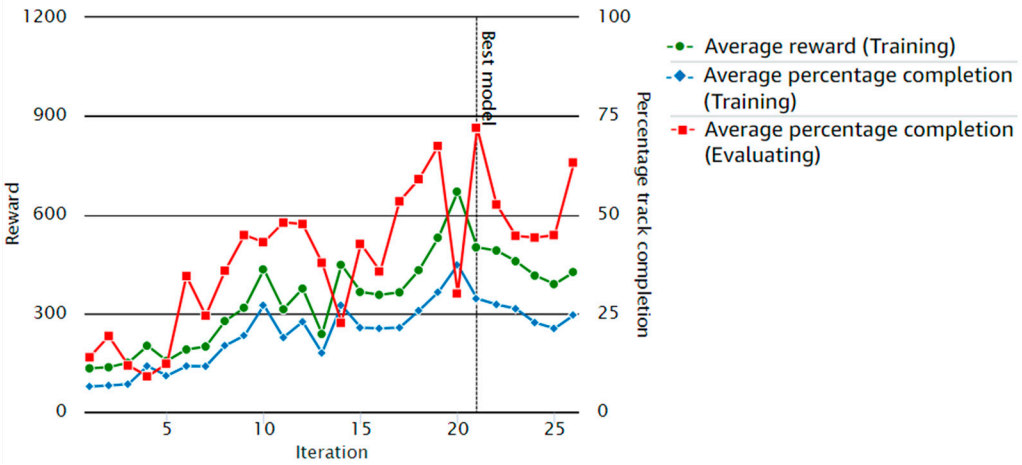

Figure 7. Extended baseline model with LiDAR reward graph.

Table 7. Evaluation results of the extended baseline model with LiDAR.

Trial	Time (MM:SS.mmm)	Trial Results	Off-Track	Off-Track Penalty	Crashes	Crash Penalty
1	00:51.988	100%	0	--	4	20 s
2	00:29.824	100%	0	--	1	5 s
3	01:04.653	100%	1	2 s	5	25 s

Despite these improvements, the model still underperformed and resulted in a significant number of crashes. This could be due to several factors: suboptimal hyperparameter settings, limitations in the PPO algorithm and reward function, or the increased number of sensors, which may lead to slower convergence.

4.6. Continuous Reward Function

All previous experiments had one common drawback in the reward function. They were designed to output discrete reward values. The function assigns the rewards based on the agent's proximity to obstacles, with the distance divided into three discrete ranges. Therefore, the agent will receive the same reward, regardless of its precise location within the region. Such a designed reward function can confuse the agent and lead to suboptimal learning outcomes, as the agent cannot distinguish subtle changes in the environment and adjust its behaviour accordingly.

In the following experiment, it was decided to handle the problem described above by designing a continuous reward function that outputs a smoothly varying reward value, directly proportional to the exact distance from the obstacle and the edge of the track. This approach allows for a more fine-grained feedback mechanism that allows the agent to make more precise adjustments to its behaviour in response to the continuously changing environment. In this way, the agent can better learn the intricacies of navigating around obstacles and the track, leading to more efficient and effective learning.

The reward graph during the training phase looked similar to the extended baseline experiment but did not seem optimal; it contained a lot of values far from the mean that were fluctuating from minimum to maximum extremes after almost each iteration (Figure 8).

The evaluation results were almost identical to those of the extended baseline experiment. The agent encountered the same problems that caused crashes in scenarios where the obstacle was outside the view area of the camera. The results included 16 crashes, one off-track penalty, and a race time of 2:45 min (Table 8).

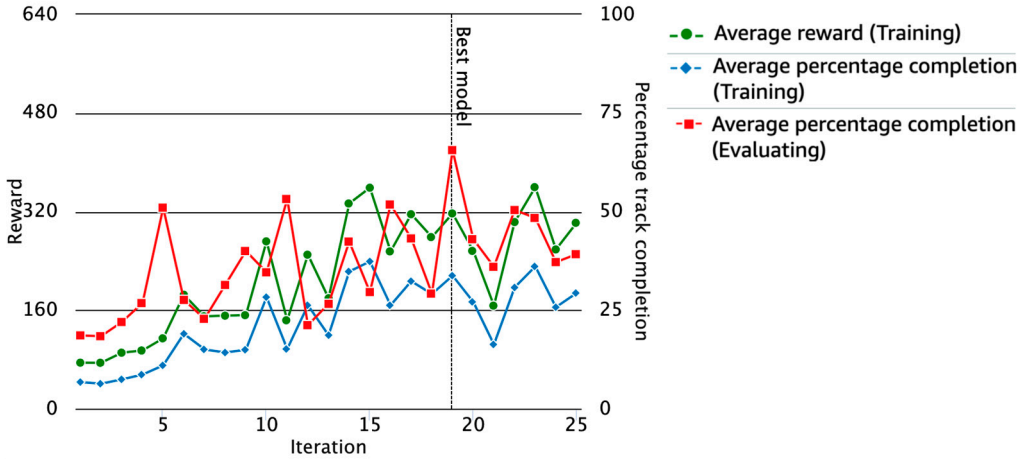

Figure 8. Continuous reward function model reward graph.

Table 8. Evaluation results of the continuous reward function model.

Trial	Time (MM:SS.mmm)	Trial Results	Off-Track	Off-Track Penalty	Crashes	Crash Penalty
1	01:12.190	100%	0	--	8	40 s
2	00:46.606	100%	0	--	4	20 s
3	00:47.126	100%	1	2 s	4	20 s

The experiment carried out did not give the desired result, suggesting that the agent likely faced a lack of data from sensors to define its behaviour.

4.7. Continuous Reward Function with Light Detection and Ranging (LiDAR)

Previously carried out experiments indicated that the data from the front-facing camera may not have been enough for the agent to navigate the environment properly. Therefore, a LiDAR sensor was added to the previously designed setup of the continuous reward function model.

The training reward graph appeared stable and depicted the gradual improvement of the model (Figure 9). The average percentage of track completion metric reached a decent level during the 16th to 20th iterations.

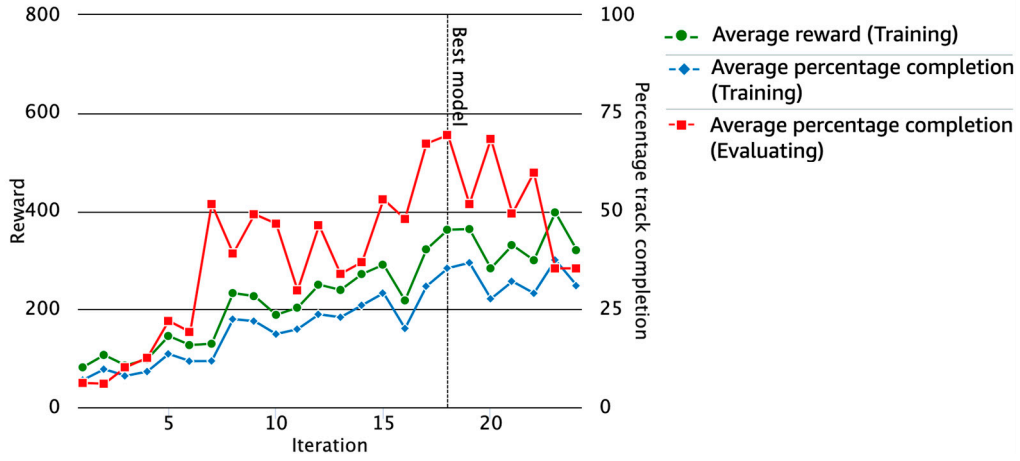

Figure 9. Continuous reward function with LiDAR model reward graph.

The evaluation process yielded a significant gain in accuracy, outperforming all previously trained models. The evaluation was carried out on the standard track (Figure 3), and the agent managed to finish the first lap without hitting any obstacles, resulting in two crashes, no off-track penalties, and a race time of 1:22 min (Table 9).

Table 9. Evaluation results of the continuous reward function with LiDAR model.

Trial	Time (MM:SS.mmm)	Trial Results	Off-Track	Off-Track Penalty	Crashes	Crash Penalty
1	00:22.857	100%	0	--	0	--
2	00:29.673	100%	0	--	1	5 s
3	00:30.285	100%	0	--	1	5 s

In addition, the agent was deployed on a previously unseen track (Figure 10) to evaluate the generalisation of the model. The environment contained four obstacles on the circuit, placed at 20%, 40%, 60%, and 80% of the track length. The agent had to finish three laps.

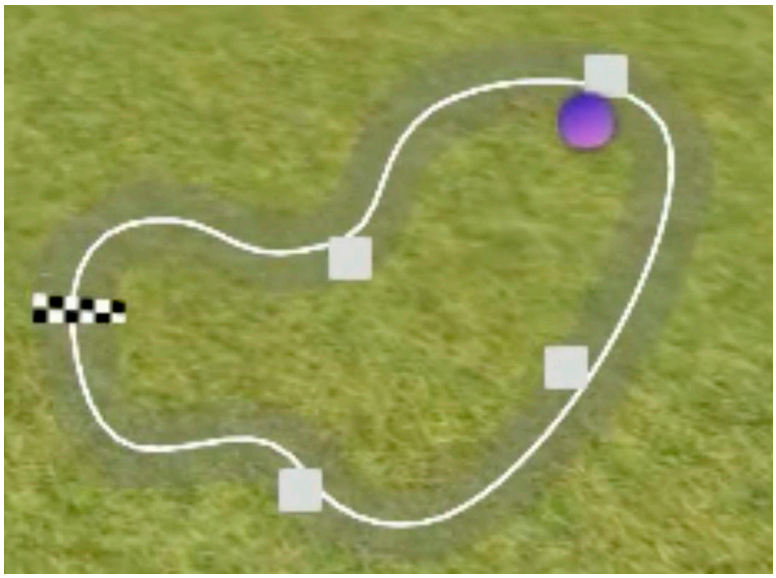

Figure 10. Unknown environment simulation track map.

The results were reasonably good for an unknown environment. The agent successfully completed two out of three laps without hitting any obstacles; however, it gained a few off-track penalties. The final results included three crashes, five penalties for not being on the track, and a total time of 1:59 min (Table 10).

Table 10. Evaluation results of the continuous reward function with LiDAR model in an unknown environment.

Trial	Time (MM:SS.mmm)	Trial Results	Off-Track	Off-Track Penalty	Crashes	Crash Penalty
1	00:53.292	100%	1	2 s	3	15 s
2	00:27.494	100%	2	4 s	0	--
3	00:39.144	100%	2	4 s	0	--

4.8. Continuous Reward Function with Reduced Action Space

The results obtained in the previously conducted experiment were satisfactory. As a next step, it was decided to make the model more lightweight to achieve a faster convergence while keeping the accuracy at the desired level. An approach was to reduce the action space of the PPO algorithm (Table 11).

Table 11. Reduced action space.

Action No.	Steering Angle (°)	Speed (m/s)
1	−30.0	0.75
2	−15.0	0.75
3	0.0	0.75
4	15.0	0.75
5	30.0	0.75

The initial steering angles were retained, and a constant speed of 0.75 m/s was used. Such a speed value was used as it is a mean of the maximum (1.00 m/s) and minimum (0.50 m/s) speed values from the default action space (Table 2). The resulting model (Figure 11) had a low average percentage of track completion across all iterations.

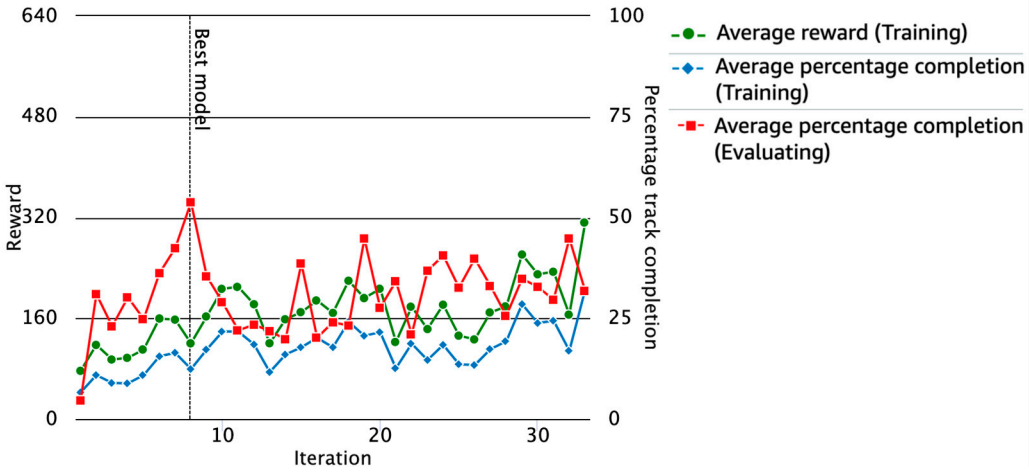

Figure 11. Continuous reward function with reduced action space model reward graph.

The model failed to converge, and as a result, did not finish the evaluation in the allotted time. One of the potential failure reasons may be the smaller value of the maximum speed of the agent compared to the previous experiments (0.75 m/s vs. 1.00 m/s), resulting in a shorter distance covered and a slower convergence during training.

Another hypothesis is that various speed values are important for proper model training. The model may benefit from lower speed values when bending the track and higher speed values in straight sections. However, the current agent had only one relatively high speed value, which it was trying to maintain, leading to a significant number of crashes and a failed evaluation as a result.

The resulting metrics from all the experiments carried out are summarised in Table 12.

Table 12. Comparative results of the experiments.

Experiment Name	Total Crashes	Total Off-Track	Total Race Time	Laps without Crashes
Baseline	48	0	06:31.129	0
SAC baseline	N/A *	N/A *	N/A *	N/A *
Extended baseline	15	1	02:40.256	0
Extended baseline with LiDAR	10	1	02:26.465	0
Continuous reward	16	1	02:45.922	0
Continuous reward with LiDAR	2	0	01:22.815	1
Continuous reward with LiDAR, unknown environment	3	5	01:59.930	2
Continuous reward, reduced space	N/A *	N/A *	N/A *	N/A *

(*): Experiment failed.

5. Discussion and Future Research

5.1. Evaluation of Findings

The results of the experiments confirm the effectiveness of end-to-end reinforcement learning methods for training autonomous driving agents, particularly for obstacle avoidance tasks in simulated environments. In the context of this paper, it involves training the AWS DeepRacer car by using a three-layer convolutional neural network architecture coupled with a multi-sensor setup, comprising a camera and LiDAR mounted on top of the vehicle. The addition of LiDAR enhances the environmental awareness of the agent, enabling the model not only to extract both relevant visual and depth information about its environment, but also to accurately correlate the input from the environment with the reward it receives.

In addition to the sensor suite, this research also established that several other factors need to be considered when designing these self-driving systems, as follows:

1. The reward function: Using a continuous reward function that outputs smoothly varying reward values that are directly proportional to the agent's distance from the obstacle and the edge of the track, rather than one that rewards discrete values based on the agent's proximity to obstacles, provided a more effective guidance to the model. As a result, the agent could make much more precise adjustments to its driving behaviour and react quickly to the environment.
2. The RL algorithm: SAC and PPO are both model-free reinforcement learning algorithms commonly used in DeepRacer to train agents to complete tasks. SAC is a model-free algorithm that can learn from past experiences generated by any policy, while PPO is also a model-free algorithm but is more sample-efficient, meaning that it can learn a good policy with fewer training examples. Judging from the reward graph (Figure 5), the trained SAC model was unstable, exhibiting extreme values in both training and evaluation. This instability led to the agent not completing the race in the allotted time. This could be explained by the fact that SAC uses a continuous action space, which requires more data to accurately represent the agent's actions. In contrast, PPO uses a discrete action space, making it less susceptible to overfitting and allowing for faster convergence. It is possible that training SAC for a longer time would have yielded better results in this particular case, but it is not guaranteed. The reason for the instability of the SAC model is likely due to the large number of parameters and the complex nature of the DeepRacer environment. Additionally, the continuous action space requires a large amount of data to learn a good policy. Increasing the training time would give SAC more opportunities to explore the environment and try different actions. This could lead to the discovery of better policies and a reduction in instability. However, it is also possible that longer training would simply reinforce the existing instability.

5.2. Possible Applications

The use of AWS DeepRacer to train intelligent agents to navigate simulated tracks using reinforcement learning algorithms holds great potential for real-world applications. One promising area of application lies in the development of autonomous vehicles, where RL algorithms can be used to train agents to navigate complex environments and avoid obstacles. The use of a simulated environment enables the safe and efficient testing of autonomous vehicle systems prior to their deployment on actual roads.

Another potential application is in the development of robotics, where RL algorithms can be utilised to train robots to perform complex tasks in diverse environments. Robots could be trained to traverse cluttered settings, such as warehouses or factories, to perform tasks such as picking up and packing items.

The use of RL algorithms to train intelligent self-driving vehicles can also extend to the gaming industry. Game developers can use these algorithms to create more intelligent and lifelike non-playable characters (NPCs) [67], which can interact with players in more engaging ways.

5.3. Future Research

The research presented in this article has opened several promising avenues for further investigation. One such avenue of exploration is to modify the simulation environment to reproduce more complex driving scenarios that were not investigated in this work. It is important to note that designing these scenarios should only be carried out in controlled and predefined environments where the agent can learn from repeated trials and adapt to specific patterns and obstacles. Introducing dynamism and unpredictability to the environment, such as varying traffic patterns, pedestrian movements, traffic signals, and diverse environmental conditions, pose significant challenges that extend well beyond what RL models like AWS DeepRacer are designed for. For applications requiring robust performances in highly dynamic environments, other specialised reinforcement learning approaches or platforms, such as CARLA or AirSim, may be more suitable.

Additionally, while the chosen best-performing reward function effectively incentivises the agent to move to the opposite lane in case it encounters an obstacle in its path, by implementing a circular 'Danger Zone' around the obstacle that is proportional to the agent's distance from the obstacle, this could lead, in some edge cases where the object is placed behind a blind corner, for example, to unpredictable behaviour. Future research could implement a more intricate zone geometry around the obstacle to penalise such behaviour. Furthermore, in some cases, the trained model exhibited a tendency to fall off track while avoiding obstacles. A possible point of investigation could look at the trade-off between encouraging the agent to stay on track and avoiding obstacles. This could involve adjusting the weights in the reward function to prioritize track adherence. Generalisation is also an important property that should be evaluated, specifically how well the trained models can adapt when faced with completely new and unknown environments. Models that fail to perform well on unseen data are often the result of overfitting [68].

6. Conclusions

This research explores the use of AWS DeepRacer as a viable solution to train end-to-end RL autonomous driving models to instruct car agents to navigate a simulated environment. Through experimentation, the performance of various RL algorithms and configurations is assessed, specifically focussing on training agents to manoeuvre a track efficiently while avoiding obstacles. Different models were trained with different configurations and the results were recorded. The best experiment used a combination of a single-lens camera for visual information, and a LiDAR mounted on top of the DeepRacer vehicle to collect depth information and generate accurate 3D maps of the environment. In addition, the model used the PPO algorithm with the default action space and a custom function that rewarded continuous values in the punishment zones. This setup resulted in a significant gain in accuracy, particularly when compared to the baseline approach provided by the AWS documentation, with a total of two crashes over the entire evaluation track distance, compared to 48 for the baseline (a 95.8% improvement), whilst reducing both the individual lap times and the total race time, i.e., a total of 1:22 min compared to 6:30 min (a 78.9% decrease). The reproducibility of these results is ensured through the public availability of the source code and detailed instructions for training the obstacle avoidance algorithm using the AWS DeepRacer web console. For the full implementation details, code, and results obtained, refer to the public GitHub repository at https://github.com/ktu-samurai-team/aws-deepracer-research (accessed on 2 January 2024).

Overall, the presented discoveries are of importance in shaping intelligent agents in autonomous driving simulations. By shedding light on the strengths and limitations of different RL algorithms, sensor configurations, action spaces, and reward functions, the results offer valuable insights. These findings can guide the development of more potent and streamlined approaches for training intelligent agents and applying them to real world environments. The analysis and conclusions are drawn from a diverse range of sources to enrich the depth of exploration.

Author Contributions: Conceptualization, B.P., S.P. and S.B.B.; methodology, B.P., S.P. and S.B.B.; software, B.P., S.P. and S.B.B.; validation, B.P., S.P., S.B.B. and A.O.; formal analysis, B.P., S.P. and S.B.B.; investigation, B.P., S.P. and S.B.B.; resources, B.P., S.P. and S.B.B.; data curation, B.P., S.P. and S.B.B.; writing—original draft preparation, B.P., S.P. and S.B.B.; writing—review and editing, B.P., S.P., S.B.B. and A.O.; visualization, B.P., S.P. and S.B.B.; supervision, A.O.; project administration, A.O.; funding acquisition, A.O. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data are available from the corresponding author upon reasonable request.

Acknowledgments: The authors acknowledge the use of artificial intelligence tools for grammar checking and language improvement.

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

References

1. Singh, S. Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey. 2015. Available online: https://trid.trb.org/view.aspx?id=1346216&source=post_page (accessed on 2 January 2024).
2. Montgomery, W.D.; Mudge, R.; Groshen, E.L.; Helper, S.; MacDuffie, J.P.; Carson, C. America's Workforce and the Self-Driving Future: Realizing Productivity Gains and Spurring Economic Growth. 2018. Available online: https://trid.trb.org/view/1516782 (accessed on 2 January 2024).
3. Yurtsever, E.; Lambert, J.; Carballo, A.; Takeda, K. A Survey of Autonomous Driving: Common Practices and Emerging Technologies. *IEEE Access* **2020**, *8*, 58443–58469. [CrossRef]
4. OpenAI; Andrychowicz, M.; Baker, B.; Chociej, M.; Jozefowicz, R.; McGrew, B.; Pachocki, J.; Petron, A.; Plappert, M.; Powell, G.; et al. Learning dexterous in-hand manipulation. *Int. J. Robot. Res.* **2020**, *39*, 3–20. [CrossRef]
5. Tan, J.; Zhang, T.; Coumans, E.; Iscen, A.; Yunfei Bai, Y.; Danijar Hafner, D.; Steven Bohez, S.; Vincent Vanhoucke, V. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv* **2018**, arXiv:1804.10332.
6. Peng, X.B.; Andrychowicz, M.; Zaremba, W.; Abbeel, P. Sim-to-real transfer of robotic control with dynamics randomization. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia, 21–25 May 2018; pp. 1–8.
7. Muratore, F.; Treede; Gienger, M.; Peters, J. Domain randomization for simulation-based policy optimization with transferability assessment. In Proceedings of the Conference on Robot Learning, Zürich, Switzerland, 29–31 October 2018; pp. 700–713.
8. Mandlekar, A.; Zhu, Y.; Garg, A.; Fei-Fei, L.; Savarese, S. Adversarially robust policy learning: Active construction of physically-plausible perturbations. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 3932–3939.
9. Zhu, Y.; Mottaghi, R.; Kolve, E.; Lim, J.J.; Gupta, A.; Fei-Fei, L.; Farhadi, A. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 29 May–3 June 2017; pp. 3357–3364.
10. Higgins, I.; Pal, A.; Rusu, A.; Matthey, L.; Burgess, C.; Pritzel, A.; Botvinick, M.; Blundell, C.; Lerchner, A. Darla: Improving zeroshot transfer in reinforcement learning. In Proceedings of the 34th International Conference on Machine Learning, Sydney, NSW, Australia, 6–11August 2017; Volume 70, pp. 1480–1490.
11. Kaur, P.; Taghavi, S.; Tian, Z.; Shi, W. A Survey on Simulators for Testing Self-Driving Cars. In Proceedings of the 2021 Fourth International Conference on Connected and Autonomous Driving (MetroCAD), Detroit, MI, USA, 28–29 April 2021. [CrossRef]
12. Li, Y.; Yuan, W.; Yan, W.; Shen, Q.; Wang, C.; Yang, M. Choose Your Simulator Wisely: A Review on Open-source Simulators for Autonomous Driving. *arXiv* **2023**, arXiv:2311.11056.
13. Juliani, A.; Berges, V.P.; Teng, E.; Cohen, A.; Harper, J.; Elion, C.; Goy, C.; Gao, Y.; Henry, H.; Mattar, M.; et al. Unity: A General Platform for Intelligent Agents. *arXiv* **2018**. Available online: https://arxiv.org/abs/1809.02627 (accessed on 6 November 2023).
14. Balaji, B.; Mallya, S.; Genc, S.; Gupta, S.; Dirac, L.; Khare, V.; Roy, G.; Sun, T.; Tao, Y.; Townsend, B.; et al. DeepRacer: Autonomous Racing Platform for Experimentation with Sim2Real Reinforcement Learning. In Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 31 May–31 August 2020; ISBN 9781728173955. [CrossRef]
15. Cheng, Y.; Wang, G.Y. Mobile robot navigation based on lidar. In Proceedings of the 2018 Chinese Control and Decision Conference (CCDC), Shenyang, China, 9–11 June 2018; ISBN 9781538612446. [CrossRef]
16. Khaksar, W.; Vivekananthen, S.; Sahari, K.; Yousefi, M.; Alnaimi, F. A review on mobile robots motion path planning in unknown environments. In Proceedings of the 2015 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS), Langkawi, Malaysia, 18–20 October 2015; ISBN 9781467371247. [CrossRef]
17. Lei, X.; Zhang, Z.; Dong, P. Dynamic Path Planning of Unknown Environment Based on Deep Reinforcement Learning. *J. Robot.* **2018**, *2018*, 5781591. [CrossRef]

18. Hui, H.; Yuge, W.; Wenjie, T.; Jiao, Z.; Yulei, G. Path Planning for Autonomous Vehicles in Unknown Dynamic Environment Based on Deep Reinforcement Learning. *Appl. Sci.* **2023**, *13*, 10056. [CrossRef]
19. Hausknecht, M.; Stone, P. Deep Recurrent Q-Learning for Partially Observable MDPs. *arXiv* **2015**, arXiv:1507.06527.
20. Güçkiran, K.; Bolat, B. Autonomous Car Racing in Simulation Environment Using Deep Reinforcement Learning. In Proceedings of the 2019 Innovations in Intelligent Systems and Applications Conference (ASYU), Izmir, Turkey, 31 October 2019–2 November 2019; ISBN 9781728128689. [CrossRef]
21. Cai, P.; Wang, H.; Huang, H.; Liu, Y.; Liu, M. Vision-Based Autonomous Car Racing Using Deep Imitative Reinforcement Learning. *IEEE Robot. Autom. Lett.* **2021**, *6*, 7262–7269. [CrossRef]
22. Rezaee, K.; Yadmellat, P.; Chamorro, S. Motion Planning for Autonomous Vehicles in the Presence of Uncertainty Using Reinforcement Learning. In Proceedings of the 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Prague, Czech Republic, 27 September–1 October 2021. [CrossRef]
23. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; ISBN 9781467388511. [CrossRef]
24. Gao, M.; Chang, D.E. Autonomous Driving Based on Modified SAC Algorithm through Imitation Learning Pretraining. In Proceedings of the 2021 21st International Conference on Control, Automation and Systems (ICCAS), Jeju, Republic of Korea, 12–15 October 2021. [CrossRef]
25. Hien, P.X.; Kim, G.-W. Goal-Oriented Navigation with Avoiding Obstacle based on Deep Reinforcement Learning in Continuous Action Space. In Proceedings of the 2021 21st International Conference on Control, Automation and Systems (ICCAS), Jeju, Republic of Korea, 12–15 October 2021. [CrossRef]
26. Song, Q.; Liu, Y.; Lu, M.; Zhang, J.; Qi, H.; Wang, Z.; Liu, Z. Autonomous Driving Decision Control Based on Improved Proximal Policy Optimization Algorithm. *Appl. Sci.* **2023**, *13*, 6400. [CrossRef]
27. Muzahid, A.J.M.; Kamarulzaman, S.F.; Rahman, M.A. Comparison of PPO and SAC Algorithms towards Decision Making Strategies for Collision Avoidance Among Multiple Autonomous Vehicles. In Proceedings of the 2021 International Conference on Software Engineering & Computer Systems and 4th International Conference on Computational Science and Information Management (ICSECS-ICOCSIM), Pekan, Malaysia, 24–26 August 2021. [CrossRef]
28. Fu, M.; Zhang, T.; Song, W.; Yang, Y.; Wang, M. Trajectory Prediction-Based Local Spatio-Temporal Navigation Map for Autonomous Driving in Dynamic Highway Environments. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 6418–6429. [CrossRef]
29. Mavrogiannis, A.; Chandra, R.; Manocha, D. B-GAP: Behavior-Rich Simulation and Navigation for Autonomous Driving. *IEEE Robot. Autom. Lett.* **2022**, *7*, 4718–4725. [CrossRef]
30. Mohammadhasani, A.; Mehrivash, H.; Lynch, A.F.; Shu, Z. Reinforcement Learning Based Safe Decision Making for Highway Autonomous Driving. *arXiv* **2021**, arXiv:2105.06517.
31. Liu, M.; Zhao, F.; Niu, J.; Liu, Y. ReinforcementDriving: Exploring Trajectories and Navigation for Autonomous Vehicles. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 808–820. [CrossRef]
32. Sallab, A.E.; Abdou, M.; Perot, E.; Yogamani, S. Deep Reinforcement Learning framework for Autonomous Driving. *Electron. Imaging* **2017**, *2017*, 70–76. [CrossRef]
33. Liu, M.; Zhao, F.; Yin, J.; Niu, J.; Liu, Y. Reinforcement-Tracking: An Effective Trajectory Tracking and Navigation Method for Autonomous Urban Driving. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 6991–7007. [CrossRef]
34. Sun, Q.; Zhang, L.; Yu, H.; Zhang, W.; Mei, Y.; Xiong, H. Hierarchical Reinforcement Learning for Dynamic Autonomous Vehicle Navigation at Intelligent Intersections. In *KDD '23M Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Long Beach, CA, USA, 6–10 August 2023*; ACM: New York, NY, USA, 2023. [CrossRef]
35. Tremblay, M.; Halder, S.S.; de Charette, R.; Lalonde, J.-F. Rain Rendering for Evaluating and Improving Robustness to Bad Weather. *Int. J. Comput. Vis.* **2021**, *129*, 341–360. [CrossRef]
36. Chen, G.; Huang, V. Ensemble Reinforcement Learning in Continuous Spaces—A Hierarchical Multi-Step Approach for Policy Training. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence {IJCAI-23}*; International Joint Conferences on Artificial Intelligence Organization: San Francisco, CA, USA, 2023. [CrossRef]
37. Zhai, P.; Luo, J.; Dong, Z.; Zhang, L.; Wang, S.; Yang, D. Robust Adversarial Reinforcement Learning with Dissipation Inequation Constraint. *Proc. AAAI Conf. Artif. Intell.* **2022**, *36*, 5431–5439. [CrossRef]
38. Tiboni, G.; Arndt, K.; Kyrki, V. DROPO: Sim-to-real transfer with offline domain randomization. *Robot. Auton. Syst.* **2023**, *166*, 104432. [CrossRef]
39. Jaekyeom, K.; Seohong, P.; Gunhee, K. Constrained GPI for Zero-Shot Transfer in Reinforcement Learning. NeurIPS Proceedings. Available online: https://proceedings.neurips.cc/paper_files/paper/2022/hash/1d8dc55c1f6cf124af840ce1d92d1896-Abstract-Conference.html (accessed on 8 November 2023).
40. Truong, J.; Chernova, S.; Batra, D. Bi-Directional Domain Adaptation for Sim2Real Transfer of Embodied Navigation Agents. *IEEE Robot. Autom. Lett.* **2021**, *6*, 2634–2641. [CrossRef]
41. Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; Koltun, V. CARLA: An Open Urban Driving Simulator. In Proceedings of the 1st Annual Conference on Robot Learning. Conference on Robot Learning, Mountain View, CA, USA, 13–15 November 2017.
42. Shah, S.; Dey, D.; Lovett, C.; Kapoor, A. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *Field and Service Robotics*; Springer International Publishing: Cham, Switzerland, 2017; pp. 621–635, ISBN 9783319673608. [CrossRef]

43. Espié, E.; Guionneau, C.; Wymann, B.; Dimitrakakis, C.; Coulom, R.; Sumner, A. *TORCS, The Open Racing Car Simulator*; EPFL: Lausanne, Switzerland, 2015.
44. Krajzewicz, D.; Hertkorn, G.; Feld, C.; Wagner, P. SUMO (Simulation of Urban MObility); An open-source traffic simulation. In Proceedings of the 4th Middle East Symposium on Simulation and Modelling, Sharjah, United Arab Emirates, 28–30 October 2002; ISBN 90-77039-09-0.
45. Terapaptommakol, W.; Phaoharuhansa, D.; Koowattanasuchat, P.; Rajruangrabin, J. Design of Obstacle Avoidance for Autonomous Vehicle Using Deep Q-Network and CARLA Simulator. *World Electr. Veh. J.* **2022**, *13*, 239. [CrossRef]
46. Zhang, E.; Zhou, H.; Ding, Y.; Zhao, J.; Ye, C. Learning How to Avoiding Obstacles for End-to-End Driving with Conditional Imitation Learning. In *SPML '19, Proceedings of the 2019 2nd International Conference on Signal Processing and Machine Learning, Hangzhou, China, 27–29 November 2019*; ACM: New York, NY, USA, 2019; ISBN 9781450372213. [CrossRef]
47. Zhang, S.; Wang, S.; Yu, S.; Yu, J.J.Q.; Wen, M. Collision Avoidance Predictive Motion Planning Based on Integrated Perception and V2V Communication. *IEEE Trans. Intell. Transp. Syst.* **2022**, *22*, 9640–9653. [CrossRef]
48. Liang, X.; Liu, Y.; Chen, T.; Liu, M.; Yang, Q. Federated Transfer Reinforcement Learning for Autonomous Driving. In *Federated and Transfer Learning*; Springer International Publishing: Cham, Switzerland, 2022; pp. 357–371, ISBN 9783031117473. [CrossRef]
49. Spryn, M.; Sharma, A.; Parkar, D.; Shrima, M. Distributed deep reinforcement learning on the cloud for autonomous driving. In *ICSE '18, Proceedings of the 40th International Conference on Software Engineering, Gothenburg, Sweden, 28 May 2018*; ACM: New York, NY, USA, 2018; ISBN 9781450357395. [CrossRef]
50. Capo, E.; Loiacono, D. Short-Term Trajectory Planning in TORCS using Deep Reinforcement Learning. In Proceedings of the 2020 IEEE Symposium Series on Computational Intelligence (SSCI), Canberra, ACT, Australia, 1–4 December 2020; ISBN 9781728125473. [CrossRef]
51. Wang Sen Jia, D.; Weng, X. Deep Reinforcement Learning for Autonomous Driving. *arXiv* **2019**, arXiv:1811.11329.
52. Spatharis, C.; Blekas, K. Multiagent reinforcement learning for autonomous driving in traffic zones with unsignalized intersections. *J. Intell. Transp. Syst.* **2024**, *28*, 103–119. [CrossRef]
53. Garza-Coello, L.A.; Zubler, M.M.; Montiel, A.N.; Vazquez-Hurtado, C. AWS DeepRacer: A Way to Understand and Apply the Reinforcement Learning Methods. In Proceedings of the 2023 IEEE Global Engineering Education Conference (EDUCON), Salmiya, Kuwait, 1–4 May 2023. [CrossRef]
54. Zhu, W.; Du, H.; Zhu, M.; Liu, Y.; Lin, C.; Wang, S.; Sun, W.; Yan, H. Application of Reinforcement Learning in the Autonomous Driving Platform of the DeepRacer. In Proceedings of the 2022 41st Chinese Control Conference (CCC), Hefei, China, 25–27 July 2022. [CrossRef]
55. Du, H. A Dynamic Collaborative Planning Method for Multi-vehicles in the Autonomous Driving Platform of the DeepRacer. In Proceedings of the 2022 41st Chinese Control Conference (CCC), Hefei, China, 25–27 July 2022. [CrossRef]
56. Tian, A.; John, E.G.; Yang, K. Poster: Unraveling Reward Functions for Head-to-Head Autonomous Racing in AWS DeepRacer. In *MobiHoc '23, Proceedings of the Twenty-Fourth International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing, Washington, DC, USA, 23–26 October 2023*; ACM: New York, NY, USA, 2023. [CrossRef]
57. Koenig, N.; Howard, A. Design and use paradigms for gazebo, an open-source multi-robot simulator. In Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), Sendai, Japan, 28 September–2 October 2004; ISBN 0780384636. [CrossRef]
58. Li, J.; Abusharkh, M.; Xu, Y. DeepRacer Model Training for autonomous vehicles on AWS EC2. In Proceedings of the 2022 International Telecommunications Conference (ITC-Egypt), Alexandria, Egypt, 26–28 July 2022. [CrossRef]
59. Mccalip, J.; Pradhan, M.; Yang, K. Reinforcement Learning Approaches for Racing and Object Avoidance on AWS DeepRacer. In Proceedings of the 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC), Torino, Italy, 26–30 June 2023. [CrossRef]
60. AWS DeepRacer Concepts and Terminology. AWS DeepRacer. Available online: https://docs.aws.amazon.com/deepracer/latest/developerguide/deepracer-basic-concept.html (accessed on 26 December 2023).
61. AWS DeepRacer Evo Is Coming Soon. Amazon Web Services. Available online: https://aws.amazon.com/blogs/machine-learning/aws-deepracer-evo-is-coming-soon-enabling-developers-to-race-their-object-avoidance-and-head-to-head-models-in-exciting-new-racing-formats/ (accessed on 26 December 2023).
62. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347.
63. Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; Moritz, P. Trust Region Policy Optimization. *arXiv* **2015**, arXiv:1502.05477.
64. Haarnoja, T.; Zhou, A.; Hartikainen, K.; Tucker, G.; Ha, S.; Tan, J.; Kumar, V.; Zhu, H.; Gupta, A.; Abbeel, P.; et al. Soft Actor-Critic Algorithms and Applications. *arXiv* **2018**, arXiv:1812.05905.
65. AWS DeepRacer Reward Function Examples. AWS DeepRacer. Available online: https://docs.aws.amazon.com/deepracer/latest/developerguide/deepracer-reward-function-examples.html#deepracer-reward-function-example-3 (accessed on 26 December 2023).
66. Understanding Racing Types and Enabling Sensors Supported by AWS DeepRacer. AWS DeepRacer. Available online: https://docs.aws.amazon.com/deepracer/latest/developerguide/deepracer-choose-race-type.html#deepracer-get-started-training-object-avoidance (accessed on 2 January 2024).

67. Kovalský, K.; Palamas, G. Neuroevolution vs Reinforcement Learning for Training Non Player Characters in Games: The Case of a Self Driving Car. In *Social Informatics and Telecommunications Engineering*; Lecture Notes of the Institute for Computer Sciences; Springer International Publishing: Cham, Switzerland, 2021; pp. 191–206. [CrossRef]
68. Géron, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2022.