Pedestrian Avoidance Simulation by Deep Reinforcement Learning Using Webots

Chalumpol Trararak
Dept. of Computer and Network Eng.
Univ. of Electro-Communications
Tokyo, Japan
chalumpol@vlsilab.ee.uec.ac.jp

Trong-Thuc Hoang
Dept. of Computer and Network Eng.
Univ. of Electro-Communications
Tokyo, Japan
hoangtt@uec.ac.jp

Pham Cong-Kha
Dept. of Computer and Network Eng.
Univ. of Electro-Communications
Tokyo, Japan
phamck@uec.ac.jp

***Abstract*—Road accidents involving pedestrians remain a leading cause of mortality worldwide. As autonomous driving technology advances to address this issue, Deep Reinforcement Learning (DRL) has emerged as a promising approach for teaching vehicles to navigate safely, particularly in avoiding pedestrians. This paper presents a DRL algorithm for pedestrian avoidance in autonomous four-wheeled vehicles, simulated using Webots and Deepbots framework. This study employs stochastic and deterministic algorithms to train the vehicle in specific simulated scenarios. The study also aims to enhance pedestrian avoidance capabilities, advance autonomous vehicle safety, and address a challenge in fully autonomous vehicles capable of drivers' and pedestrians' safety. The study also aims to provide a fundamental approach for further improvement on safer autonomous driving DRL algorithms in Webots simulators in the future.**

***Index Terms*—Deep Reinforcement Learning, Webots, Pedestrian Avoidance, Simulation**

I. Introduction

Road mortality emerges as a hugely familiar health uneasiness, contributing substantially to mortality rates worldwide. According to the 2019, Approximately 1.19 million yearly fatalities, 30% of the deaths are between the ages of 5 and 29 years. Moreover, An analysis of the fatality distribution across various road user categories reveals a concerning pattern: two- and three-wheeled vehicle users (cyclists and motorcyclists) accounted for the highest proportion at 30% of total fatalities, occupants of four-wheeled vehicles accounted for the proportion 25% of total fatalities, and pedestrians, a particularly vulnerable group, comprising 21% of the overall mortality rate [1]. With regard to the statistics, road accidents must emphasize the development of road safety measures. Substantial improvements in vehicle safety technologies have innovated to upgrade road security for vehicle drivers and pedestrians.

With safety technological improvements in modern vehicles, these technologies raise performance through advanced mobility control and accident avoidance algorithms. Currently, The advent of the AI era has shaken up autonomous driving algorithms. This evolution highlights tremendous progress toward fully autonomous vehicles, performing promising improvements and safety that may encounter adverse driving environments [2].

*This work is funded by the Royal Thai Scholarship.

DRL algorithm uses a trial-and-error approach to optimize the agent in numerous environments, such as autonomous driving [3]. It allows modern vehicles to learn and improve dodging navigation by sensor data, which enables immediate sensibility, decision-making, and control to evade hurdles. The robustness and flexibility of DRL are idealized for real-world implementation, cruising as an advanced algorithm in autonomous driving systems to safer maneuvers in sophisticated atmospheres.

This study is inspired by the DRL algorithms on pedestrian avoidance to escape pedestrian accidents, especially in the occluded area. Many DRL works apply this approach, including pedestrian avoidance [4] and UAV obstacle avoidance [5]. Moreover, upgrading Advanced Driving Assitance Systems to boost autonomous driving safety [6]. DRL is an approach to upgrade the vehicles' mobility and safety to escape difficult circumstances. Also, this study employs Webots, an open-source robotics simulator. It simulates rat behavior [7], models indoor robotic waste bins [8], replicates real-world scenarios analyzing agent's path planning [9], and creates complex driving environments for automotive research [10]. With the implementation above, Webots was utilized as a simulation platform for this study due to its versatility and robust capabilities.

From the aforementioned inspirations, This study employs pedestrian avoidance simulation with a four-wheeled car containing sensors using the Webots simulator. It implements DRL in stochastic and deterministic models to train the agent in pedestrian avoidance and safely reach the destination. Indeed, this study encourages the DRL algorithm for pedestrian avoidance, improving the safety-based autonomous driving simulation and the development of simulated autonomous vehicles.

II. Background

DRL builds on the principles of the Markov Decision Process, defined by states, actions, transition probabilities, rewards, and a discount factor. It creates decision-making scenarios through trial-and-error actions in specified environments. It allows the agent to learn policies appropriately and select the best action in the environment to achieve the goal. The DRL approach proceeds in both stochastic and

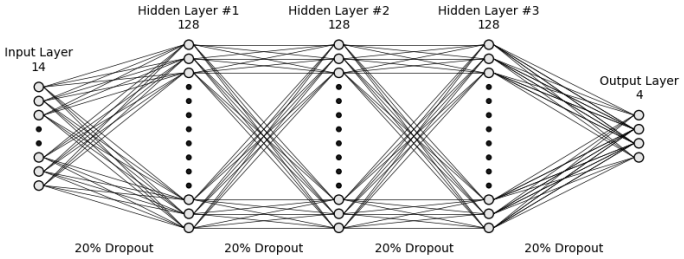

Fig. 1. Neural Network architecture implemented in this study.

deterministic models by exploiting deep neural networks to empower reinforcement learning for optimal policy.

Fig.[1] illustrates the neural network architecture implemented in this study comprises a sophisticated multi-layer structure designed for autonomous navigation tasks. The input layer processes 14 distinct parameters representing critical environmental and vehicular state information. This feeds into three densely connected hidden layers, containing 128 neurons in each layer, providing sufficient computational capacity for complex pattern recognition. To enhance the generalization capabilities of the model and prevent overfitting, dropout regularization with a 20% rate is systematically applied between successive hidden layers. Network optimization employs the Adam algorithm, facilitating stable convergence through adaptive learning rate adjustments for individual parameters. The architecture culminates in an output layer of four neurons, corresponding to the action space probabilities for autonomous navigation decision-making.

A. Deep Q - Learning

Deep Q-Learning (DQN) is a sophisticated model-free deterministic reinforcement learning technique that merges original q-learning with a deep neural network; in principle, this method leverages the Bellman equation as an iterative update mechanism to estimate the action-value function. The key of DQN is the optimal q-value calculation, which is proved from a combination of episodic rewards, estimated action values generated by the neural network, and a discount factor [11].

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}[r + \gamma \max_{a'} Q^*(s',a')|s,a] \tag{1}$$

Where $Q^*(s,a)$ represents the optimal action-value function, r is the accumulated rewards, γ defines the discount factor balancing the episodic reward and future reward, which is between 0 and 1. $Q^*(s',a')$ is the estimated action-value function from next state and action.

Iterative minimizing the loss function $L_i(\theta_i)$ can improve the DQN algorithm by continually updating the neural network weights inside the model. This procedure optimizes the Q-value approximates to refine the model's accuracy and action-value function estimate performance [11].

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim p(\cdot)}[(y_i - Q(s,a;\theta_i))^2] \tag{2}$$

Where y_i illustrates the optimal action-value function, $Q(s,a;\theta_i)$ and θ_i are the estimated action-value function and the network hyperparameters at iteration i, respectively.

Regarding the DQN approach, This study employs a DQN with a target network to imitate the action network with the mean-squared error loss function, shown in Algorithm 1. The policy of the agent is chosen by the maximum of the action value from the output layer.

Algorithm 1 Deep Q-Learning with Target Network and experience replay. [11]

Initialize Replay Memory D to Capacity N
Initialize action-value function Q with random weights θ
Initialize target-value function $\hat{Q}$ with weights $\theta^- = \theta$
for episode $= 1, T$ **do**
  Initialize sequence $s_1 = x_1$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t otherwise select $a_t = argmax_a Q(\phi(s_t), a; \theta)$
    Execute action a_t and observe reward r_t and observation x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocessed $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a, r, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_t, a, r, \phi_{t+1})$ from D
    $y_j = \begin{cases} r_j & \text{(terminate)} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{(otherwise)} \end{cases}$
    Perform a gradient descent step on $(y_i - Q(s,a;\phi))^2$ w.r.t. the network parameters θ
    reset $\hat{Q} = Q$
  end for
end for

B. Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) trains in a policy-based way that observes the sampling actions regarding its latest stochastic policy. The figure of action selection randomness relies on initial conditions and training methods. During training, the policy regularly trains less randomly due to an iterative update approach, which enables the leverage of rewards. However, local minima can emerge from the updated policy [12]. The advantage function is calculated from (3)

$$\hat{A}_t = \mathbb{E}_t[r(s,a) + \gamma \upsilon_{\pi_{\theta_{old}}}(s') - \upsilon_{\pi_{\theta_{old}}}(s)] \tag{3}$$

Where $r(s,a)$ is the episode's reward, γ defines the discount factor, $\upsilon_{\pi_{\theta_{old}}}(s')$ and $\upsilon_{\pi_{\theta_{old}}}(s)$ are the value function from the value's network in the next state and current state.

The surrogate function is calculated from (4), consisting of the clipping function stabilizing the policy updates. The function's key is the clipping parameter (ϵ), which excessively prevents policy updates that could destabilize policy training.

$$g(\epsilon, \hat{A}_t) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{4}$$

Where $\epsilon = 0.2$, $r_t(\theta)$ is the proportional of the policy function from the policy's network in the current and older value. and $\hat{A}_t$ is the advantage function at the timestep t

To update the policy, the PPO objective function must be maximized to proceed with stochastic gradient ascent [12].

$$\theta_{k+1} = arg\max_{\theta} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \min\left(r_t(\theta)\hat{A}_t), g(\epsilon, \hat{A}_t)\right) \tag{5}$$

Algorithm 2 Proximal Policy Optimization - Clip [12]

```
Input: initial policy parameters θ_0 initial value function parameters φ_0
for episode = 1, T do
    Collect set of trajectories D_k = {τ_i} by running policy π_k = π(θ_k)
    Compute rewards R̂_t
    for PPO update Iteration = 1, N do
        Compute advantage estimates, Â_t
        Update the policy by (4)
        Fit value function by (5)
    end for
end for
```

The regression on mean-squared error is applied to fit the value function [12].

$$\phi_{k+1} = arg\min_{\phi} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2 \tag{6}$$

Where $V_\phi(s_t)$ is the predicted value of the value network in the current state and $\hat{R}_t$ is the sum of discounted rewards.

From Algorithm 2, PPO trains in an on-policy way that observes the sampling actions regarding its latest stochastic policy. The figure of action selection randomness relies on initial conditions and training methods. Over the training, the policy regularly trains less randomly due to an iterative update approach, which enables the leverage of rewards. As a result, actions can either continue to be sampled stochastically or be selected deterministically by taking the action with maximum probability from the output layer.

III. Implementation

A. Scenario

The experiment utilizes Webots, an advanced 3D simulator, to model a complex scenario, Fig. 2, with a 4-wheeled agent navigating a dual-lane road (0.6m/lane) around a static obstacle and an occluded pedestrian. The agent moves at 0.6 m/s, starting 1.3m from the obstacle, with the pedestrian crossing at 0.16 m/s on an adjacent lane 1.8m away - the pedestrian starts walking if the agent can drive to 1.5 m; this simulation aims to find the agent's optimal policy for agent navigation to evade the obstacle and pedestrian and safely reach the goal, which is 2.5m far from the robot.

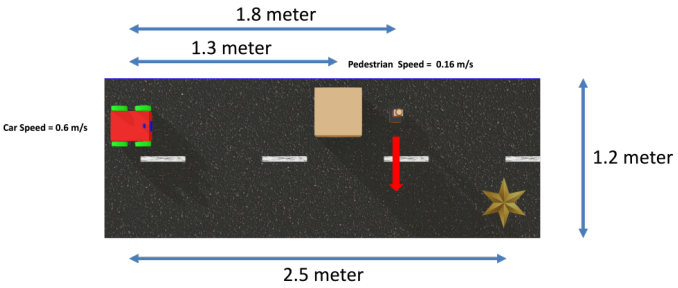

Fig. 2. Pedestrian avoidance scenario in Webots simulator.

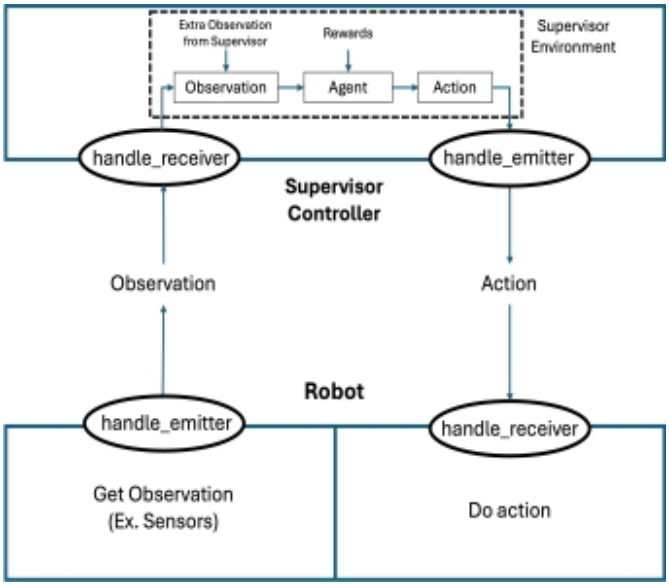

Fig. 3. Robot-Supervisor scheme on Deepbots [13]

B. 4-Wheeled Robot Settings

The robot is configured within the simulated environment using node structures. Its chassis is formed as a square solid node measuring 26cm x 18cm x 5cm in dimension, while the wheels are modeled as cylinders with a 5cm radius and 3cm thickness. The front wheel is shaped as two single hinge-joint nodes to enable driving and steering. In contrast, the rear wheels employ single hinge joints solely for propulsion.

The robot contains an array of front-mounted distance sensors and a camera for obstacle and pedestrian detection. An internal GPS tracks the robot's position, and rotational encoders on the motors to measure wheel speed and steering angle. The data from sensors allows us to learn how to navigate by DRL in occluded pedestrian situations.

C. Deepbots

Deepbots, a python open-source framework integrated with Webots, enables deep reinforcement learning in robotics. It duplicates the famous OpenAI-Gym interface for user-friendly simulation [13]. This study applies the Deepbots framework to launch DQN and PPO algorithms for robot control and goal-oriented navigation using the robot-supervisor scheme

demonstrated in Fig. 3, spectating their benefits in complex scenarios within a simulation environment.

a) States: This study employs a 14-parameter state representation for the agent, including normalized coordinates, sensor readings, steering angle, and distances to environmental elements. It incorporates lane detection status, obstacle and pedestrian detection states, and the robot's roll angle.

b) Actions: The agent's movement defines four discrete action spaces to manipulate the agent at the iteration in the episode. Action 0 accelerates at 0.15 m/s speed, while Action 1 decelerates at 0.075 m/s speed. Action 2 steers the agent to the left at 5 degrees, and Action 3 steers it to the right at 5 degrees. These action spaces simplify the decision-making action for the agent's movements.

c) Rewards: The rewards per episode are computed by movement progress and goal achievement and deducted from the misdirection and collision penalty. In contrast, The movement progress rewards are calculated from the agent's goal-distance change between timesteps and punished by the agent's immobility. Similarly, a 20-point bonus applies when accomplishing the goal. This reward structure improves the agent's movement and eliminates inappropriate policy.

$$rewards_{prog} = \begin{cases} (d_{r,tar_{t+1}} - d_{r,tar_t}) * 0.15 & \Delta d > 0 \\ -0.05 & \Delta d = 0 \end{cases} \quad (7)$$

Where $d_{r,tar_{t+1}}$ and d_{r,tar_t} are the distance between the robot and target in timestep t+1 and t accordingly.

An obstacle collision produces a penalty of 7 points, while a pedestrian collision produces a more severe deduction of 8 points from the rewards. The out-of-lane penalty is calculated from the minimum distance between the agent and the target.

$$penalty_{outlane} = \min\left[abs(2.733 - d_{r,target_{t+1}} * 10, 2)\right] \quad (8)$$

Where $d_{r,tar_{t+1}}$ is the distance between the robot and target in timestep t+1.

d) Hyperparameters: In this study, the simulation hyperparameters of DQN and PPO are presented in Table I.

IV. Usage Notes

The example simulation code is available at: https://github.com/DragonKorn1/dqn_ppo_avoid_webots.
This study uses computing hardware, including an Intel Core i9-9820X processor with 128 GB of RAM. The system employs an NVIDIA Quadro K620 GPU with CUDA version 11.8 and CUDNN version 8.9.7 for GPU computation.

V. Evaluation and Comparison

This paper comprehensively evaluates the rewards obtained and the robot's maneuverability capabilities through an extensive experimental period spanning 30,000 episodes.

TABLE I
Simulation Hyperparameters [14]

Hyperparameters	DQN	PPO
Discount factor (γ)	0.99	0.99
Starting epsilon (ε_{start})	1	1
Decaying epsilon (ε_{decay})	0.99	0.99
End epsilon (ε_{end})	0.01	0.01
Batch size	256	256
Learning rate	5E-05	5E-05
Weight decay (Adam)	1E-04	1E-04
Target update interval	100	-
Clipped parameter (ϵ)	-	0.2
Critic update iteration	-	7

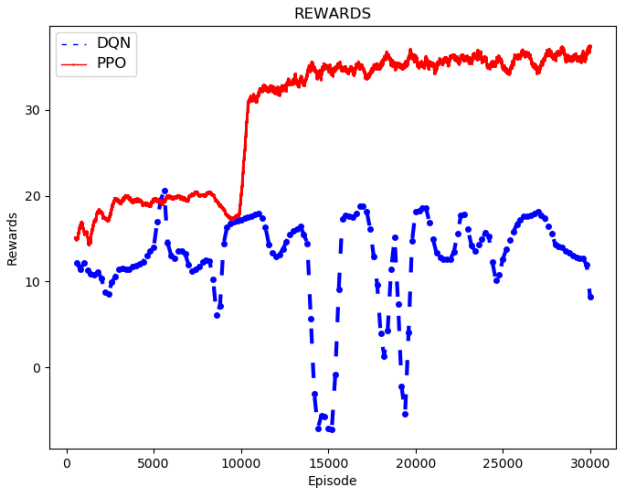

Fig. 4. 500th Moving average accumulated rewards of DQN (red line) vs. PPO (blue line)

A. Rewards

Fig. 4 demonstrates the 500^{th} moving average accumulated rewards between DQN and PPO; the DQN shows high volatility with frequent drops, some reaching near-zero rewards. This oscillating trend demonstrates deficient learning stability with poor rewards. In contrast, PPO demonstrates more stable learning, rapidly improving around the $10,000^{th}$ episode and then slightly increasing the rewards, implying solid learning performance and maintaining a consistently higher reward.

From the rewards, the DQN algorithm, in this paper, consists of a policy network with a target network and performs unstable learning while struggling with the policy's network updates due to overestimation. Unlike the PPO, which comprises the policy and value networks, it undergoes better rewards. This algorithm stabilizes the policy's update by minimizing the value's function from the value network. Furthermore, the clipping function is crucial to handling the overestimation from the policy's updates. As a result, This function improves the overall algorithm's performance, reflecting higher accumulated rewards and the agent's mobility in the scenario.

B. Agent's Maneuverability

Fig. 5 portrays the navigation behavior by DQN and PPO algorithms, revealing significant differences in their effective-

ness for autonomous agent mobility due to policy overestimation during training iterations. The DQN-trained agent performs with limited capability, hesitant forward movement, and inefficient navigation. This result explicitly indicates its struggle to accomplish the scenario. In contrast, the PPO-trained agent demonstrates superior performance by maintaining full speed while executing precise rightward adjustments to simultaneously avoid static obstacles and moving pedestrians while efficiently progressing toward the goal. The stable learning with overestimation handling by the clipping function and actor-critic neural networks enables the agent to receive better results in training. Thus, These results demonstrate that PPO provides more reliable and efficient autonomous navigation capabilities than DQN's more limited approach.

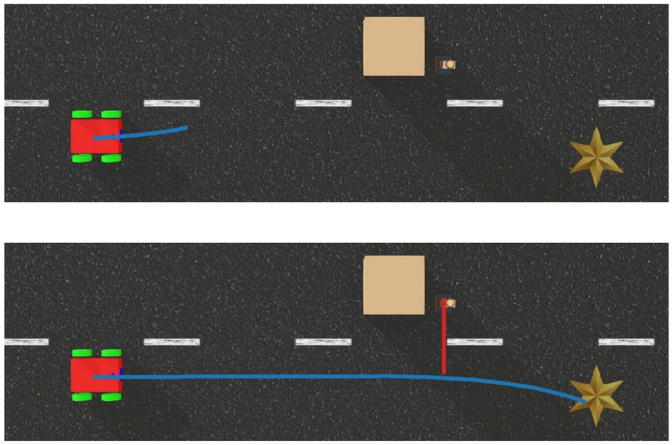

Fig. 5. Robot navigation paths of DQN (top left) and PPO (bottom left) and performance metrics at 30000^{th} episode.

C. Comparing of DRL Across Simulators and Tasks

Following Table II, various simulators can use the DRL simulation applications. This study, concentrating on pedestrian avoidance using Webots with Deepbots as a DRL framework, provides valuable insights compared to other simulation environments. Analysis of related research reveals different performance patterns across platforms and algorithms.

A study using Gazebo with ROS [15] for view planning tasks offers high-fidelity physics simulation and sensor modeling, though with higher computational resources than Webots. Research utilizing Pygame for UAV object avoidance [16] highlighted how environmental complexity impacts algorithm performance. The training iterations may train faster because of the simpler physics; however, they may not comprehensively capture real-world dynamics.

OpenAI Gym implementations for mobile robot pathfinding [17] demonstrated notably different performance characteristics. This reversal of the typical performance pattern underscores how task-specific requirements can fundamentally affect algorithm suitability. The Duckietown simulator [18], which is focused on autonomous driving scenarios, also reflects its specialized reward structure for driving tasks.

Our implementation using Webots simulates complex navigation tasks within a realistic range. The Webots environment balances simulation fidelity and computational efficiency, making it particularly suitable for autonomous vehicle research. Each simulator demonstrates distinct advantages: Gazebo excels in sensor simulation, Pygame offers rapid prototyping capabilities, OpenAI Gym provides standardized environments, and Duckietown specializes in autonomous driving scenarios.

TABLE II
SIMULATION COMPARISON WITH OTHER RESEARCH.

Research	Simulator	DRL Framework	Rewards	
			DQN	PPO
[15]	Gazebo + ROS	Stable Baselines3	0.5	0.8
[16]	Pygame	Stable Baselines3	-5.666	31.117
[17]	OpenAI Gym	Stable Baselines3	100%	56.1%
[18]	Duckietowns	Stable Baselines3	∼5,700	∼2,000
This study	Webots	Deepbots	12.64	29.61

This comparative analysis reveals that both simulator characteristics and task specifications heavily influence algorithm performance. While PPO generally outperforms DQN across platforms for navigation tasks, exceptions exist based on specific environmental constraints and objectives. These findings emphasize the importance of carefully matching simulation platforms and algorithms to specific research requirements in autonomous driving applications.

VI. Conclusion and Future Work

Following the simulation result, As the DQN algorithm, the robot struggles to learn the scenario, appearing as the reward, and it slightly navigates towards the goals but fails to avoid pedestrians and reach the goal. This algorithm demonstrates lower performance for this pedestrian avoidance scenario. In contrast, the PPO algorithm performs more efficiently, archiving to reach the goal safely. The PPO's rewards reflect efficient, stable learning over the episode. However, the PPO takes over 10000 episodes to stabilize the learning.

However, although the PPO's algorithm completes the task, the accumulated reward does not reach the maximum possible reward after the 30000^{th} iteration. The extended training duration required for the policy global minima convergence indicates potential inefficiencies in the learning process that could be addressed through algorithmic refinements. The suboptimal accumulated reward suggests that the agent may adopt satisfactory but not optimal navigation strategies, possibly due to premature convergence to local maxima in the policy space.

In term of the simulation transferring to the real-world scenario perspective, although this platform furnishes precious insights for algorithm development and testing, it is important to acknowledge that entire validation through real-world testing remains essential for ensuring reliable autonomous vehicle performance. This validation necessitates further research to evaluate and validate the reliability of simulation-based optimizations to practical applications. This assures that algorithms developed in the controller environment maintain their performance when deployed in real-world scenarios.

Furthermore, training for stable performance in a real-world scenario requires significant iterations and presents practical challenges for deployment and actual adaptation. This limitation becomes important when considering the desire to retrain or adapt the system for different environmental conditions or vehicle configurations.

In the future, other techniques will be applied, such as Prioritized Experience Replay (PER), which adjusts prioritized experiences' probabilities to perform more frequently in agent training, and Curriculum Learning, which makes an agent learn from simple to complex traffic scenarios in a similar environment, for example, the agent learns from driving on the straight road to safely maneuvering in a crowded complex intersection or roundabout. Likewise, more advanced algorithms will also be introduced, for example, soft actor-critic (SAC), Deep Deterministic Policy Gradient (DDPG), and Double Deep Q-Learning (DDQN) to improve the agent's learning performance and computational time.

Future research also explores more complex and diverse scenarios within the Webots simulation environment to address these limitations and advance the capabilities of autonomous navigation systems, including implementing multi-agent interactions with increased vehicle and pedestrian density and introducing challenging environmental conditions such as nighttime operations and varied terrain characteristics. Moreover, near real-world simulation will emerge for real-world benchmarking. This approach would make it much easier to imitate the environment and vehicle configuration to retrain in real-world scenarios, which reduces the implementation time and cost to train the agent.

REFERENCES

[1] World Health Organization (WHO), *Global Status Report on Road Safety 2023*. Geneva: World Health Organization, 2023. [Online]. Available: https://iris.who.int/bitstream/handle/10665/375016/9789240086517-eng.pdf

[2] M. M.-Díaz and F. Soriguera, "Autonomous Vehicles: Theoretical and Practical Challenges," *Transportation Research Procedia*, vol. 33, pp. 275–282, 2018.

[3] W. Terapaptommakol, D. Phaoharuhansa, P. Koowattanasuchat, and J. Rajruangrabin, "Design of Obstacle Avoidance for Autonomous Vehicle Using Deep Q-Network and CARLA Simulator," *World Electric Vehicle Journal*, vol. 13, no. 12, pp. 1–13, Dec. 2022.

[4] H. Chen, X. Cao, L. Guvenc, and B. A.-Guvenc, "Deep-Reinforcement-Learning-Based Collision Avoidance of Autonomous Driving System for Vulnerable Road User Safety," *Electronics*, vol. 13, no. 10, May 2024.

[5] B. Joshi, D. Kapur, and H. Kandath, "Sim-to-Real Deep Reinforcement Learning based Obstacle Avoidance for UAVs under Measurement Uncertainty," pp. 1–9, Mar. 2023. [Online]. Available: https://arxiv.org/abs/2303.07243

[6] D. Lee and M. Kwon, "ADAS-RL: Safety Learning Approach for Stable Autonomous Driving," *ICT Express*, vol. 8, no. 3, pp. 479–483, Sep. 2022.

[7] X. Zou, E. Scott, A. Johnson, K. Chen, D. Nitz, K. D. Jong, and J. Krichmar, "Neuroevolution of a Recurrent Neural Network for Spatial and Working Memory in a Simulated Robotic Environment," in *Genetic and Evolutionary Comp. Conf. Companion*, Jul. 2021, p. 289–290.

[8] M. Mohan, R. M. K. Chetty, K. M. Azeem, P. Vishal, B. Poornasai, and V. Sriram, "Modelling and Simulation of Autonomous Indoor Robotic Wastebin in Webots for Waste Management in Smart Buildings," in *IOP Conf. Series: Materials Science and Engi.*, vol. 1012, no. 1, Jan. 2021, p. 012022.

[9] R. Wang, "The Comparison and Analysis of Autonomous Food Delivery Robot Based on Artificial Potential Field and Breadth-First Search Methods," in *IEEE Int. Conf. on Elec. Engi., Big Data and Algorithms (EEBDA)*, Feb. 2023, pp. 1943–1947.

[10] S. De, Y. Huang, S. Mohamed, D. Goswami, and H. Corporaal, "Hardware- and Situation-Aware Sensing for Robust Closed-Loop Control Systems," in *Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, Feb. 2021, pp. 1751–1756.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," Dec. 2013. [Online]. Available: https://arxiv.org/abs/1312.5602

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," Aug. 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[13] M. Kirtas, K. Tsampazis, N. Passalis, and A. Tefas, "Deepbots: A Webots-Based Deep Reinforcement Learning Framework for Robotics," in *Artificial Intelligence Appl. and Innovations (AIAI)*, Jun. 2020, pp. 64–75.

[14] L. Schäffer, Z. Kincses, and S. Pletl, "Hyperparameter Optimisation of Reinforcement Learning Algorithms in Webots Simulation Environment," in *IEEE Int. Symp. on Comp. Intelligence and Info. (CINTI)*, Nov. 2023, pp. 000 065–000 070.

[15] C. Landgraf, B. Meese, M. Pabst, G. Martius, and M. F. Huber, "A Reinforcement Learning Approach to View Planning for Automated Inspection Tasks," *Sensors*, vol. 21, no. 6, pp. 1–17, Mar. 2021.

[16] Z. Rybchak and M. Kopylets, "Comparative Analysis of DQN and PPO Algorithms in UAV Obstacle Avoidance 2D Simulation," in *Int. Conf. on Comp. Linguistics and Intelligent Syst. (CEUR)*, vol. 3688, Apr. 2024, pp. 391–403.

[17] M. Sugimoto, R. Uchida, K. Kurashige, and S. Tsuzuki, "Evaluation of a Path Finding Algorithm for Mobile Robots using Deep Reinforcement Learning," *The Japanese J. of the Inst. of Industrial Appl. Engi.*, vol. 9, no. 2, pp. 118–124, Sep. 2021.

[18] P. B. Almási, "Real-world Deep Reinforcement Learning via Simulation for Autonomous Driving," Master's thesis, Budapest Univ. of Tech. and Economics, Budapest, Hungary, May 2021. [Online]. Available: https://www.hte.hu/documents/10180/4724350/Almasi_Peter_Bela-MSc_diplomaterv.pdf