Reinforcement Learning-Based Guidance of Autonomous Vehicles

Abstract

Reinforcement learning (RL) has attracted significant research efforts to guide an autonomous vehicle (AV) for a collision-free path due to its advantage of investigating interactions among multiple vehicles and the dynamic environment. This study deploys a Deep Q-Network (DQN)-based RL algorithm with reward shaping to control an ego AV in an environment with multiple vehicles. Specifically, the state space of the RL algorithm depends on the desired destination, the ego vehicle's location and orientation, and the location of other vehicles in the system. The training time of the proposed RL algorithm is much shorter compared with most current image-based algorithms. The RL algorithm also provides an extendable framework to include different numbers of vehicles in the environment and can be easily adapted to different maps without changing the setup of the RL algorithm. Three scenarios were simulated to validate the effects of the proposed RL algorithm while guiding the ego AV interacting with multiple vehicles on straight and curvy roads. Our results showed that the ego AV could learn to reach its destination within 5000 episodes for all scenarios tested.

Keywords

Autonomous Vehicle, Reinforcement Learning, Cars Learn to Act (CARLA)

1. Introduction

Over the past several years, control of autonomous vehicles has been a rapidly growing field of research. The promise of safer roads and more smooth traffic flow has transformed AVs into a billion-dollar market. Progresses have been made to guide AVs with model-based approaches.[1-3] However, a fully autonomous vehicle is not right around the corner, as the complex physical components involved in traffic systems are difficult to model. To tackle this challenge different model-free machine learning algorithms and guidance strategies have been developed. Of these, Markov Decision Process (MDP) has been one of the promising direction for autonomous driving as a probability-based decision-making strategy.[4-8] These results reported path planner for an AV in different environment including driving of an AV in a single lane, lane change of an AV in multiple lanes, and intention of other human-drivers in the system. These MDP based approaches still highly depend on the model of the driving environment.

Recently, RL has shown great potential to derive the appropriate decision-making strategy by AVs with little or no knowledge of a mathematical model for the dynamical environment.[9] Instead, an AV can observe its environment in a given state, takes actions, and receives rewards for those actions. Through "trial and error" iterations, these rewards guide the AV towards its goal, allowing the AVs to perform complex tasks.

Due to the wide adoption of cameras as an onboard sensor for vehicles, one current trend in utilizing RL algorithms for the decision-making of AVs is to represent the ego vehicle's observations of the environment with images captured by the AV. These images were referred to as states for RL algorithms. Significant strides in collision avoidance and path guidance for an ego vehicle have been reported with the image-based decision-making strategy.[10, 11] The image-based state space has versatility in varying environments because the ego vehicle's observations are not dependent on global variables, such as its absolute position in the environment or the orientation of the road it is on. This method can be effectively deployed in environments with different factors, like road geometry, number of obstacles, and map size. However, these RL algorithms often suffer from sampling inefficiency, in which the ego vehicle learns very little during each iteration it experiences. Image-based state spaces compound this issue by requiring the process of images, increasing the complexity of the problem to be solved by the ego vehicle. This also results in a large number of iterations being needed to develop a sufficient decision-making policy. Commonly, images are processed with convolutional neural networks (CNN) to highlight the features in the images that are relevant to the ego vehicle's goal, and sophisticated RL algorithms such as Deep Deterministic Policy Gradient (DDPG) and Proximal Policy Optimization (PPO).[12, 13] Other proposed solutions include imitation learning, in which an "expert vehicle" with more information about the environment assists the ego vehicle during training.[14]

Besides the image-based approaches, many other Q-learning-based algorithms have been proposed with different states, rewards, and action pairs.[15] These studies have been focusing on different specific tasks such as collision-free path planning, merging on to a highway, and lane changing for an AV.[16-18]

In a Q-Learning algorithm for autonomous driving of an ego vehicle, a finite set of all possible states, S, and actions, A, can be defined and referred to as the state space and action space, respectively. A Q-Value is given for each available action, $a \in A$, in a state, $s \in S$. These values are determined by a Q-function, $Q(s, a)$, which is often randomly initialized. At each time step, t, a reward, r_t, is given for the ego vehicle using a particular action, a_t, to transit from the current state, s_t, to the next state, s_{t+1}. To determine a_t, a user defined hyperparameter ε in the range of [0, 1] is used to determine the probability of taking a random action or the action with the highest Q-value for s_t. When ε is closer to 1, random actions are more likely taken, representing the ego vehicle exploring the environment to learn new paths. When ε is closer to 0, the best action is taken, representing the vehicle exploiting what it already knows about the environment. Typically, this value starts close to 1 and decays to near 0 over the learning process. The Q-value for the action-state pair used in each time step, $Q(s_t, a_t)$, is then updated as the following equation:

$$Q(s_t, a_t) = Q(s_t, a_t) + \lambda[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)], \quad (1)$$

where the hyperparameters λ and γ are also user-defined values in the range [0, 1], called the learning rate and discount factor, respectively. The learning rate, λ, represents how much an update changes the current Q-value, while the discount factor, γ, determines the importance of future actions in the update. During the ego vehicle's training, this update process is performed iteratively. With enough training, $Q(s, a)$ begins to accurately reflect the actual value of the actions taken in each state, concerning the goal of the ego vehicle. Thus, the optimal policy for the ego vehicle becomes the sequence of actions with the best Q-values.

However, a major drawback to Q-Learning is that there must be a finite set of states in the state space, resulting in inaccuracies due to the quantization errors of states in a discrete system. The inaccuracy can begin to have an impact when the ego vehicle tries to learn more complex policies.

To address this issue, the DQN algorithm approximates $Q(s, a)$ with two fully connected neural networks: an online neural network with a set of parameters, θ, and a target neural network with a set of parameters, θ^-.[15] The online and target neural networks have identical structure. At a given time step, the online neural network takes s_t as its input and outputs the predicted Q-values for all actions available to that state, $Q_\theta(s_t, a)$. Either the action with the maximum Q-value, or a random action, will be the selected as a_t with the following conditions:

$$a_t = \begin{cases} best\ Q - value\ action, & n_r > \varepsilon \\ random\ action, & n_r \leq \varepsilon \end{cases} \quad (2)$$

where n_r is a random number between [0, 1] generated at each time step.

Parameter θ in the online neural network will be updated using back propagation with the difference between $Q_\theta(s_t, a_t)$ and $Q_\theta^*(s_t, a_t)$. The target neural network takes s_{t+1} as its input and outputs, $Q_{\theta^-}(s_{t+1}, a)$, the predicted Q-values for all available actions in the next state. The target Q-value, $Q^*(s_t, a_t)$, is obtained using (2) as follows:

$$Q^*(s_t, a_t) \approx r_t + \gamma \max_a Q_{\theta^-}(s_{t+1}, a). \quad (3)$$

The discount factor in (2) plays a similar role as in Q-Learning. Parameters of the target network, θ^-, will be replaced with the parameters, θ, from the online neural network for every user-defined number of steps, C.

In a RL learning algorithm with DQN setup, observation of the ego vehicle at a time point is defined as a tuple, $e_t=(s_t, s_{t+1}, a_t, r_t)$ and stored as part of the ego vehicle's history. Input to the online network also includes past observations, s_i. The parameters, θ, are instead updated using a batch of uniformly sampled past observations. The procedure is defined as experience replay, which improves the learning efficiency using prior experiences.

The ego vehicle will continue to take steps in the environment until it reaches a terminal state, signifying the end of an episode. When reaching a terminal state, the simulation environment resets to its initial conditions and training continues with a new episode. This allows for the ego vehicle to experience similar states again and explore different actions for those states. Since there is no state after the terminal state, the target Q-value, $Q^*(s_t, a_t)$, reduces to just the reward gained for transitioning to the terminal state, r_t, for all actions.

This project seeks to develop a DQN algorithm to guide an ego vehicle in a dynamic environment with multiple other vehicles. The driving strategies of the vehicles are not shared with each other while the ego vehicle can observe the position of other nearby vehicles. The DQN algorithm is developed with an image-free state space definition, whose state variables are referenced locally. The state space in the DQN can be easily deployed in different environments with minimal adjustments. To further improve the training speed of the vehicle, a reward function was developed using reward shaping.[19, 20] The motion of the vehicles will be displayed in Cars Learn Act (CARLA) simulator.[21] To test the performance of the RL algorithm, training performances of different scenarios were evaluated, and the accuracy of the vehicle's trajectory was observed. It was found that the vehicle was able to learn a sufficient enough policy to reach its destination in 5000 episodes, roughly 750,000 time step, in all scenarios tested.

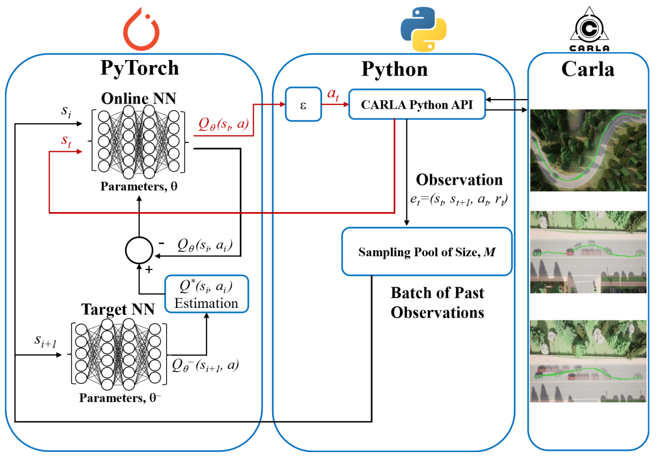

Figure 1: Flow diagram used during training. Each step, the action loop (red) is taken first, then the update loop (black) is done for each sample, i, in the batch.

2. Methodology

Training and testing of the proposed DQN algorithms have been performed in a framework shown in Figure 1. The proposed DQN RL algorithm is developed in PyTorch which interacts with CARLA through a CARLA Python API. The structure of the neural networks used in the DQN algorithm utilized two hidden layers, each with 64 nodes, implemented using PyTorch 1.13.0. The CARLA Python API allows to PyTorch programs to send commands and receive observations about the environment in CARLA.

Training of the ego vehicle was performed episodically. The episode ended if the ego vehicle reached its destination, travelled past the destination, travelled into an oncoming lane, left the road, or collided with an obstacle.

2.1. State and Action Space Definition

A state in the system, $S = (s_e, s_{v1}, \ldots, s_{vn})$, is composed of an ego vehicle's state, s_e, and mobile vehicles' states, $s_{v1} - s_{vn}$, for n number of mobile vehicles existing in the environment. To define the ego vehicle state, $s_e = (\delta_w, \phi_w, \delta_l, \phi_l, l)$, a vector, v_e, can be drawn from the ego vehicle to the destination, as shown in Figure 2(a). The magnitude of v_e determines the variable, δ_w, which represents the distance between the ego vehicle and the destination. The angle difference between v_e and the heading of the lane at the destination is represented as ϕ_w. Since the destination position and lane heading do not change during the episode, the two variables, δ_w and ϕ_w, encode the localized position of the ego vehicle. The variable, δ_l, is the distance between the ego vehicle and the center of its current lane. The angle difference between the heading of the ego vehicle's current lane and the ego vehicle's orientation defines the variable, ϕ_l. Finally, the variable, $l \in [0, 4]$, is an integer described in (4), representing whether the ego vehicle is on a road and if there is a lane present to the left or right of its current lane. These three variables encode the relationship of the ego vehicle with its current lane.

$$l = \begin{cases} 0 & if\ not\ on\ road, \\ 1 & both\ lanes\ present, \\ 2 & only\ right\ lane\ , \\ 3 & only\ left\ lane, \\ 4 & no\ \ lane\ present. \end{cases} \tag{4}$$

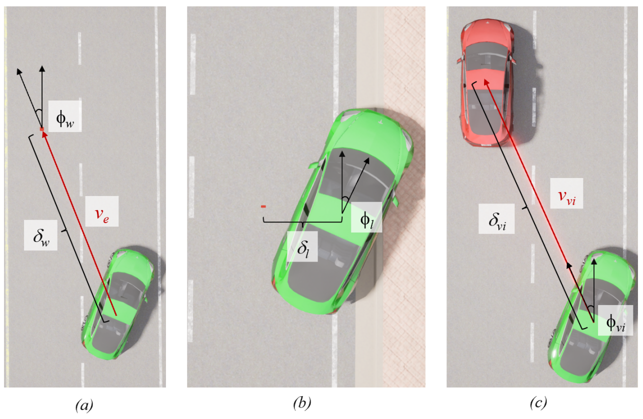

Figure 2: Visualization of state space definition for a vehicle with respect to destination (a, b), and other vehicles (c).

For each mobile vehicle, i, in the environment, a mobile vehicle state, $s_{vi} = (\delta_{vi}, \phi_{vi})$, is defined. A vector, v_i, is created such that the vector points directly from the ego vehicle to the ith mobile vehicle. The magnitude of v_i determines the variable, δ_{vi}, representing the distance from the ego vehicle to the ith vehicle. The variable ϕ_{vi}, is defined as the angle difference between v_i and the heading of the lane at the ego vehicle's position as shown in Figure 2(c). These two variables encode the ego vehicle's position relative to the mobile vehicle, with respect to the lane the ego vehicle is in.

Five actions were defined to determine the motion of the ego vehicle with the following quantifications: the left, slight left, straight, slight right, and right actions were set to -0.5, -0.25, 0, 0.25, and 0.5, respectively. Additionally, the throttle of the ego vehicle was set at a constant 0.5 for all actions in this study. These values are all relative to the range allowed in CARLA's API, which is [-1, 1] for the steering action and [0, 1] for the throttle action.

2.2. Reward Function

To guide the ego vehicle towards learning a collision-free path to reach its desired destination in a dynamic environment, the reward function considers rewards related to the ego vehicle's deviation from its current lane as a lane reward, distance to the destination as a destination reward, and avoidance of mobile vehicles as a collision-free reward.

The lane reward, $r_l = cos(\phi_l)$, assigns a reward close to 1 if the ego vehicle's orientation is similar to the heading of the current lane, but the reward quickly reduces to 0 as the ego vehicle's orientation deviates. The angle ϕ_l represents the angle difference between the heading of the ego vehicle's current lane and the ego vehicle's orientation. As an even function, the lane reward assigns the same value no matter the sign of the variable ϕ_l. Scenarios in which ϕ_l is greater than 90 degrees end the episode, as discussed later.

A destination reward is calculated as, $r_w = (D - \delta_w)/D$. The destination reward is given to encourage the ego vehicle to move toward the destination. The constant, D, is defined as the maximum distance between the ego vehicle and its destination. This constant is fixed at the beginning of the episode.

For each vehicle, i, in the environment, a collision-free reward, r_{vi}, was defined as:

$$r_{vi} = \begin{cases} 1 - \left[\frac{R-\delta_{vi}}{R} * \cos(\phi_{vi})\right] & \delta_{vi} < R\ and\ |\phi_{vi}| < 90^o \\ 1 & otherwise \end{cases} \tag{5}$$

The safety region, R, is a radius around the ego vehicle that determines whether a mobile vehicle is close enough for a potential collision. To guide the ego vehicle away from this potential, the collision-free reward reduces with the distance between the two vehicles but increases again as the ego vehicle positions itself alongside the mobile vehicle in a neighboring lane.

It was observed that the vehicle preferred staying in its current lane over changing lanes to avoid vehicles or turning at an intersection to reach the destination. To resolve this, the destination reward and the collision-free rewards were weighted more heavily by multiplying both by a factor of 2.

The reward obtained for a given time step before the ending of the episode is then given by, $r_t = (r_l + r_w + r_{vi}) / r_m$, where r_m is the maximum reward that could be achieved. This is done so that the reward is normalized to a value between 0 and 1. A reward is given if the following reward conditions are met. Otherwise, the reward given is 0.

The reward conditions are: 1) The distance to the destination must be less than the distance in the previous time step. This condition guarantees that the vehicle is always moving toward the destination without the need for a negative reward. 2) The vehicle must be in a driving lane. This condition is used, rather than ending the episode, to encourage the ego vehicle to learn to return to the road when entering a shoulder lane.

An episode ends if the ego vehicle reaches the destination, collides with an obstacle, drives past the destination, or is oriented opposing the current lane heading. The ego vehicle is oriented opposing the current lane heading if the absolute value of the relative angle between the ego vehicle's orientation and the current lane heading, ϕ_l, is greater than 90 degrees.

If the ego vehicle reaches the destination, an additional reward of 1 is added to the reward already gained. For the other episodes ending scenarios, the reward is set to -2. Thus, the reward gained in each time step has a range of [-2, 2].

2.3. Hyperparameters

The hyperparameters used were determined experimentally. A list of the hyper parameters used is shown in Table 1. The value of ε was used to determine the action to be taken and was initialized as 1 and decreased linearly to 0.01 by episode 100. At each step, a random value between 0 and 1 was generated. If the random value is greater than ε, the best action based on the Q-value will be taken, otherwise, a random action will be taken. If ε=1, a random action is always taken. The parameters of the online NN, θ, were copied over to the parameters of the target NN, θ^-, every $C=5,000$ steps. To store the observations obtained in each time step, a buffer of size 250,000 was used. Every time step, a batch of 32 observations was used to update the parameters of the online neural network.

Table 1: Hyperparameters used when conducting training and testing.

Loss Function	Mean Square Error (MSE)
Optimizer	Adam
Learning rate, λ	0.0001
Final ε Value	0.01
Target Update, *C*	5000
Observation History Size, *M*	250000

2.4. Simulation

A CARLA 9.13 was installed and run on a Lambda workstation with an AMD Ryzen threadripper pro 3995wx and two Nvidia RTX A5000s. Three different scenarios were established. Scenario 1 was set up in "Town07" environment in CARLA on a curving single lane road. The RL algorithm will guide the ego vehicle to reach the destination 140 (D = 140) meters away while staying on the road. Scenarios 2 and 3 were set up in "Town 05" in Carla, where the ego vehicle had a collision-free path to reach the destination 70 (D=70) meters away with 1 and 2 other mobile vehicles on a straight 2-lane road. In Scenario 2, one mobile vehicle was placed in front of the ego vehicle in the same lane. Compared to Scenario 2, Scenario 3 has a 3rd vehicle which was placed ahead of the 2nd vehicle in the other lane. All mobile vehicles stayed in their lane during each episode and moved in a straight line at 0.2 throttle. The ego vehicle only knew the distance between itself and other vehicles and did not know the decision-making strategy of other vehicles.

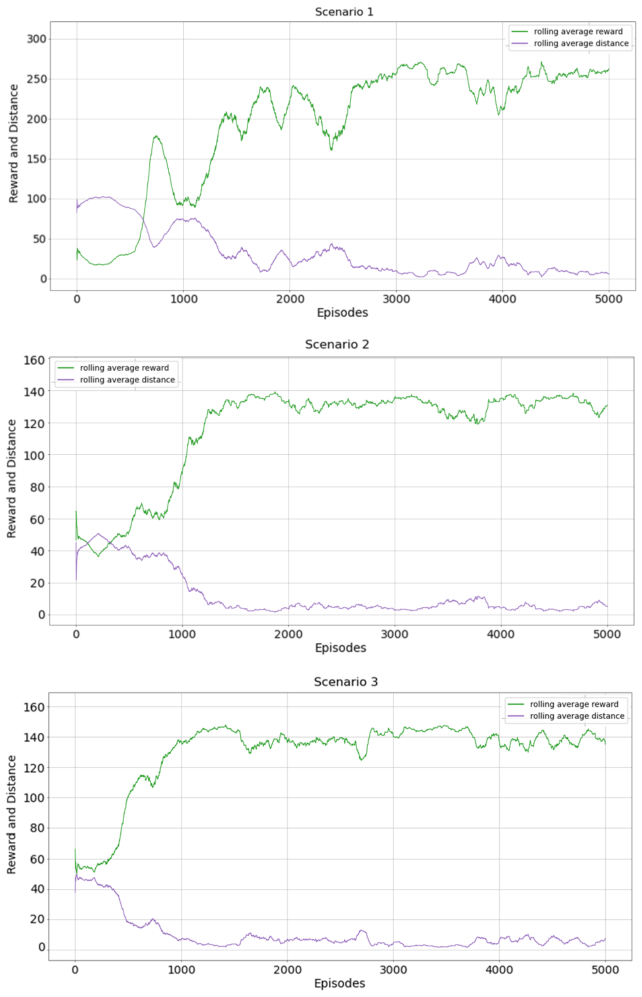

Figure 3: Rolling average reward and distance of ego vehicle to a destination during training for 5000 episodes in each scenario. The green curve represents the rolling average reward of the past 100 episodes while the blue curve represents the average distance.

3. Results

For all three scenarios, the vehicle was trained for 10,000 episodes. For Scenarios 2 and 3, it took approximately 150 steps for the agent to reach the destination. For Scenario 1, it took twice the number of steps, as the distance to the destination was further. Due to the inherent randomness of the initial exploration actions, 3 trials were performed to confirm consistency of the results. During training, the

CARLA simulator was set in synchronous mode, in which the physics of the world paused each to time step to allow the algorithm to train and provide the next action to take. When performing testing, this was set to asynchronous mode, where the physics of the world would continue to run, and actions were taken by the ego vehicle as quickly as the algorithm could provide them.

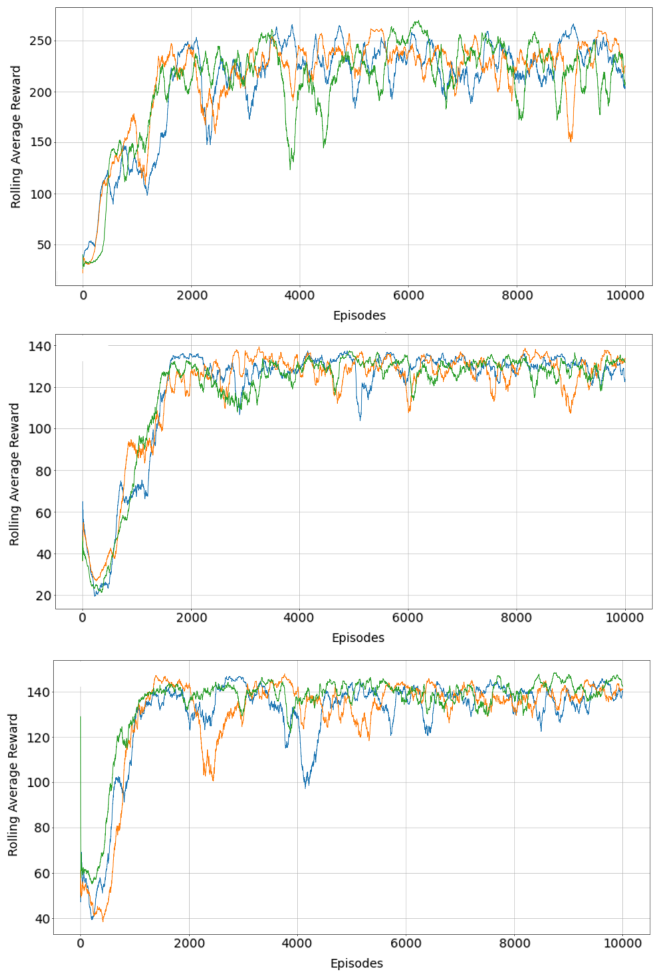

Figure 4: Rolling average rewards of three sets of training for scenario 1 (top), scenario 2 (middle), and scenario 3 (bottom). The RL algorithms all converge with similar best rewards.

3.1. Training Evaluation

The rolling average over the last 100 episodes for the reward gained and the distance from the center of the vehicle to the destination is displayed in Figure 3. It was determined, due to the of the size of the vehicle and a radius around the waypoint ending the episode, that a distance of about 10 meters indicates that the vehicle reached the waypoint. Figure 3 shows the downward trend of the distance to the waypoint for all scenarios, on average reaching the destination by episode 5000. It also shows that a rolling average reward to reach the destination is about 250 for Scenario 1, 130 for Scenario 2, and 140 for Scenario 3.

The training results for the 3 trials of each scenario are shown in Figure 4. It can be seen that the ego vehicle begins to learn successful paths to the destination by episode 4000. This is supported by the general plateau seen at that point in training. Beyond that, the ego vehicle attempts to optimize its pathing according to the reward function. However, the ego vehicle still experiences instability while training, as seen by dips in the average reward. One potential cause for this is a phenomenon known as catastrophic forgetting, where a neural network forgets what it has learned from training when it experiences new information. For Scenario 3, it seems that it can recover from this, as indicated by the reduction in the size of the dips as training progressed. Scenario 1 is particularly affected by these stability issues, as compared to the other scenarios, oscillating between 150 and 250 rolling average rewards. This is likely due to the destination being set further away.

3.2 Test Results

After the training evaluation was completed, the trial with the best performance in each environment was chosen. These were then tested on their respective environments by having the vehicle's path traced out over five episodes. Only the best action, determined by the Q-values produced from the trained online NN, was taken by setting ε to 0. Figure 5 shows an overlay of the motion of the vehicles in each test at various time points in an episode. For clarity, the points in time are labelled.

Since the reward function incentivizes the agent to stay in the episode, the optimal path becomes the longest one. This does not seem to pose too much of an issue for Scenarios 2 and 3, but it is believed to be a source of the instability seen in training for Scenario 1. It can be observed in Figure 5 that the ego vehicle's optimal policy for Scenario 1 involves hugging the edge of the lane to maximize the time it is in the episode. It is believed that this is the main source of the instability seen in the training evaluation of Scenario 1. Because the ego vehicle is so close to the edge of the road, minor steering actions taken can result in the ego vehicle leaving the lane and ending the episode early. This then negatively affects the q-values for the sequence of actions that lead to that result, despite many of them being good. Scenarios 2 and 3 are less affected by this because they have additional guidance in the form of a reward from mobile vehicles and because the distance to the destination is closer.

4. Discussion

In this paper, a DQN-based reinforcement learning algorithm was proposed to guide an ego vehicle to a destination with a collision-free trajectory. This algorithm was then implemented, trained, and tested with a framework integrating the PyTorch package and CARLA simulator. Autonomous driving for three scenarios were trained and compared to assess the viability of this RL algorithm. Our results showed that the RL algorithm did guide the vehicle in a dynamic environment to its destination. Though the simulations only show the collision-free trajectory within a limited distance from the ego vehicle to the destination, multiple destinations can be assigned continuously to guide the vehicle to any desired destination. One advantage of the proposed RL algorithm lies in the fact that the decision-making strategies of vehicles are not shared. The state of the system is defined based on the relative distance between the ego vehicles and the destination, the orientation of the ego vehicle and the roads on the map, and the positions of other vehicles near the ego vehicle. Though we only showed the scenario with two mobile vehicles in the training, more mobile vehicles could be included in the system and the training process is prolonged with the increased number of mobile vehicles.

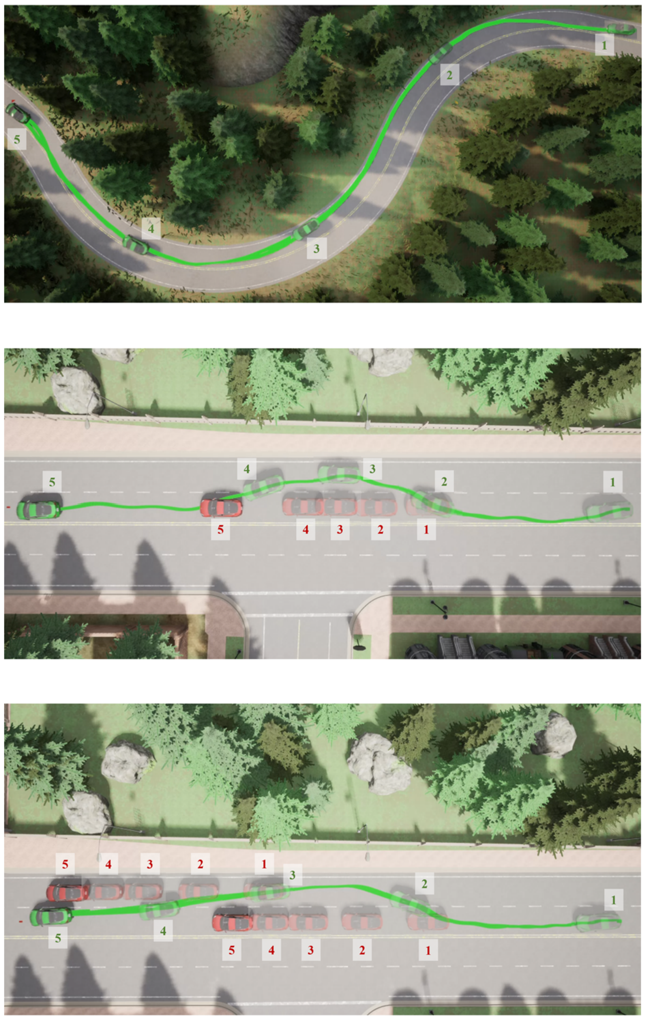

Figure 5: Testing Results of Scenario 1 (top), Scenario 2 (middle) and Scenario 3 (bottom). The motion of the ego vehicle (green) and mobile vehicles (red) are numbered 1-5 as time progressed with 1 as initial time and 5 as the latest time point.

Though the method learned quickly, it struggled with stability when the destination was placed further from the ego agent. To address the observed instability seen when the destination is placed further away from the agent, the reward function will be revisited and the use of more complex reinforcement learning algorithms will be explored.

We are aware that the ego vehicle was trained in the same environmental conditions in each episode and that the choice of actions was relatively simple. Future works will focus on varying the initial conditions of the environment, such as the starting position of the ego and mobile vehicles so that the ego vehicle learns a more generalized policy.

Additionally, more subtle control of the ego vehicle's motion could be reached by introducing more actions, which can be further optimized in future studies concerning constraints on jerk and fuel efficiency.

References

[1] M. Maurer, J. C. Gerdes, B. Lenz, and H. Winner, *Autonomous driving: technical, legal and social aspects*: Springer Nature, 2016.

[2] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, and V. Pratt, "Towards fully autonomous driving: Systems and algorithms." pp. 163-168.

[3] Y. Wang, P. Chen, and Y. Jin, "Trajectory planning for an unmanned ground vehicle group using augmented particle swarm optimization in a dynamic environment." pp. 4341-4346.

[4] S. Brechtel, T. Gindele, and R. Dillmann, "Probabilistic decision-making under uncertainty for autonomous driving using continuous POMDPs." pp. 392-399.

[5] C. Hubmann, M. Becker, D. Althoff, D. Lenz, and C. Stiller, "Decision making for autonomous driving considering interaction and uncertain prediction of surrounding vehicles." pp. 1671-1678.

[6] N. Li, A. Girard, and I. Kolmanovsky, "Stochastic predictive control for partially observable Markov decision processes with time-joint chance constraints and application to autonomous vehicle control," *Journal of Dynamic Systems, Measurement, and Control,* vol. 141, no. 7, 2019.

[7] X. Lin, J. Zhang, J. Shang, Y. Wang, H. Yu, and X. Zhang, "Decision making through occluded intersections for autonomous driving." pp. 2449-2455.

[8] S. Surya, and N. Rakesh, "Flow based traffic congestion prediction and intelligent signalling using Markov decision process." pp. 1-6.

[9] S. Aradi, "Survey of deep reinforcement learning for motion planning of autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[10] H.-T. Tseng, C.-C. Hsieh, W.-T. Lin, and J.-T. Lin, "Deep reinforcement learning for collision avoidance of autonomous vehicle." pp. 1-2.

[11] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning." pp. 3052-3059.

[12] Ó. Pérez-Gil, R. Barea, E. López-Guillén, L. M. Bergasa, C. Gómez-Huélamo, R. Gutiérrez, and A. Díaz-Díaz, "Deep reinforcement learning based control for autonomous vehicles in carla," *Multimedia Tools and Applications,* vol. 81, no. 3, pp. 3553-3576, 2022.

[13] P. Zieliński, and U. Markowska-Kaczmar, "3D robotic navigation using a vision-based deep reinforcement

learning model," *Applied Soft Computing,* vol. 110, pp. 107602, 2021.

[14] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems,* vol. 57, no. 5, pp. 469-483, 2009.

[15] C. J. Watkins, and P. Dayan, "Q-learning," *Machine learning,* vol. 8, no. 3, pp. 279-292, 1992.

[16] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, "Navigating occluded intersections with autonomous vehicles using deep reinforcement learning." pp. 2034-2039.

[17] P. Wang, C.-Y. Chan, and A. de La Fortelle, "A reinforcement learning based approach for automated lane change maneuvers." pp. 1379-1384.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski, "Human-level control through deep reinforcement learning," *nature,* vol. 518, no. 7540, pp. 529-533, 2015.

[19] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping." pp. 278-287.

[20] T. Okudo, and S. Yamada, "Subgoal-based reward shaping to improve efficiency in reinforcement learning," *IEEE Access,* vol. 9, pp. 97557-97568, 2021.

[21] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator." pp. 1-16.