Deep Q-Learning for Car Control

1st Sohan Gadiraju
Robotics
Johns Hopkins University
Baltimore, USA
sgadira2@jhu.edu

***Abstract*—In recent years, there has been a rapid increase in the development of autonomous and driving-assisted cars. One of the most challenging problems in autonomous driving is making car control decisions based on complex and uncertain environments. Deep Q-learning is a promising technique of model-free reinforcement learning that provides a method to learn complex environments for optimal decisions. In this paper, we propose a Deep Q-learning based approach for autonomous car control that evaluates the velocity and steering angle of the car to determine the optimal trajectory in multiple scenarios. We evaluate the proposed approach using a simulated car environment in MATLAB and analyzing the collision rate and time to reach goal. Although the approach was not successful for all scenarios, the results indicate deep q-learning may be applied for obstacle avoidance and car control in more real-word scenarios.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Driving is an essential part of most people's lives, with American drivers spending over 84 billion hours driving during 2015 [1] and the average driver driving over 13,000 miles a year [2]. The development of fully autonomous cars can provide drivers with an opportunity to save hours daily and put their time during more meaningful tasks. Furthermore, the implementation of assisted-driving and self-driving features into vehicles can greatly decrease the number of car accidents. In 2022, Tesla reported one crash for every 4.85 million miles driven while drivers were using their Autopilot technology, whereas people who were not using it reported one crash for every 1.40 million miles driven [3]. However, the process of creating autonomous vehicles is quite difficult as the task of driving contains a lot of uncertainties that can be difficult to model or simulate. In recent years, the emergence of deep neural networks has allowed us to model complex relationships which contain many states and uncertainties, making it quite effective for driving scenarios.

Previous studies in this field have shown positive results in utilizing Deep Q-Learning to generate trajectories for cars. One study involves a simulation study of a robocar learning to drive in an environment with static obstacles and lane markings on a circular track [4]. Another study involved comparing the Deep Q-Network (DQN) and Deep Deterministic Policy Gradient (DDPG) algorithms for control of an agent in a CARLA simulation [5]. Although both these studies utilized deep reinforcement learning, the scenarios presented for the problems were not realistic to real-world scenarios.

This study focuses on implementing Deep Q-Learning for car control in 2D real-world scenarios created using DrivingScenario in MATLAB. DrivingScenario allows creating simple 2D driving environments with vehicles and obstacles that reflect the real world. We utilize a simple bicycle dynamics model to represent the car in the environment and update the position of the car. This research shows the potential Deep Q-Learning has for obstacle avoidance and determining car trajectories in realistic situations.

II. Bicycle Dynamics

The bicycle dynamics model used for this study consists of a simplified car model based on the front and rear axles. The equations utilized assume the reference point utilized is at the center of gravity for the vehicle [6]. The model consists of a three-element vector to determine the state of a vehicle: the global positions x and y, and the direction the vehicle is heading θ. The dynamics are calculated utilizing inputs of the current velocity v and the steering angle δ to calculate $\dot{x}$, $\dot{y}$, and $\dot{\theta}$.

The following equations describe the bicycle dynamics model:

$$\begin{aligned} \beta &= \arctan(l_r * \tan(\delta)/L) \\ \dot{x} &= v \cdot \cos(\theta + \beta) \\ \dot{y} &= v \cdot \sin(\theta) \\ \dot{\theta} &= \frac{v}{L} \cdot \tan(\delta) \cdot \cos(\beta) \end{aligned} \tag{1}$$

where L represents the distance between the rear wheel and the center of gravity, which is 1.3 m for this study, and L represents the length of the vehicle, which is 4.7 m. However, as this is a simplified bicycle dynamics model, uncertainties arise. One example of uncertainty within these dynamics is the side-slip angle β, as the equation assumes tire slip can be modeled simply by lateral and longitudinal slip. The outputs are calculated using a discrete time model as shown below:

$$\begin{aligned} x_{k+1} &= x_k + \Delta t \cdot \dot{x} \\ y_{k+1} &= y_k + \Delta t \cdot \dot{y} \\ \theta_{k+1} &= \theta_k + \Delta t \cdot \dot{\theta} \end{aligned} \tag{2}$$

where Δt represents the time step.

III. Deep Q-Learning

Deep Q-Learning is an expansion on the reinforcement learning method of Q-learning using neural networks. Q-learning is a reinforcement learning algorithm which allows you to reward an agent for taking certain actions. The agent learns by calculating a Q-value for every state and action and choosing the action that minimizes it. Although Q-Learning is good for teaching agents with small action spaces, in a task as complicated as driving, the action space is too large for a Q-table to store Q-values. Deep Q-learning fixes this issue by utilizing multiple networks to approximate the Q-values. A main network which is trained every step and a target network which is a copy of the main network and is trained every 50-100 steps which helps stabilize our Q-value estimations. The networks consist on an initial input layer of 3 neurons, the intermediate two layers range between 60-256 neurons depending on the scenario, and the ouput layer is either 3 or 10 depending on the scenario. For this study, the network takes the current state [x,y,θ], where theta is the yaw angle ranging from -180 to 180 degrees, and the action set which contains the speed, v, and steering angle, delta, as inputs for the dynamic. We want our network to choose v and delta such that collisions are avoided, and the car gets to the goal state as quickly as possible. The Q-value is calculated as such:

$$Q(s,a) = Q(s,a) + \alpha \cdot (cost(s,a,r) + \gamma \cdot Q(s',a') - Q(s,a)) \tag{3}$$

The alpha represents our learning rate, which was set to 0.8, and gamma is the discount factor, which was 0.7. The current state cost function is a mix of the quadratic cost as well as the reward, r, for the states.

$$\begin{aligned} cost &= Q(X - Xd)^2 + R * u^2 + reward \\ reward &= roadPenalty + distPenalty + goalReward \end{aligned} \tag{4}$$

The Q and R represent weight matrixes for determining what state inputs we want to emphasize the most in cost. For this study, the x and y states have high weights as we want to emphasize reaching the goal position, regardless of orientation. X represents a matrix of all our observational states and Xd is our goal state while u represents our control. The rewards are calculated by taking the sum of the roadPenalty, which penalizes the state for not being on the road, distPenalty, which penalizes the car based on its distance to its closest obstacle, and lastly, a goalReward, which rewards the agent for reaching the goal state. By rewarding and penalizing the agent in this method, we are able to make the agent stay away from negative actions, such as collisions, while further pushing it to get to the goal state.

$$\begin{aligned} roadPenalty(s,a) &= \begin{cases} 0 & \text{if state is on road} \\ 1 & \text{if state is NOT on road} \end{cases} \\ distPenalty(s,a) &= \begin{cases} 5 & \text{if distance to obstacle} < 0.5 \\ \frac{1}{\text{distObstacle}} & \text{otherwise} \end{cases} \\ goalReward(s,a) &= \begin{cases} -10 & \text{if distance to goal} < 0.5 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{5}$$

We experimented with many different values for each of these reward factors and testing showed that making the penalties or rewards too high reduced convergence. To get the predicted action, an epsilon greedy algorithm was used with epsilon decreasing each simulations step.

IV. Experimental Study

A. Setup

In order to test the training algorithm, we first need an environment that can simulate driving. Utilizing DrivingScenario in MATLAB, three unique scenes were created to represent different real-world scenarios. This package allowed me to construct a 2D representation which reduced the uncertainty within the environment and made it easier for the training to converge. For this study we utilized three different scenarios to test the agent and see if it can produce correct trajectories for each scenario. The first scenario depicts a car that needs to make a left turn with two other cars on the road.

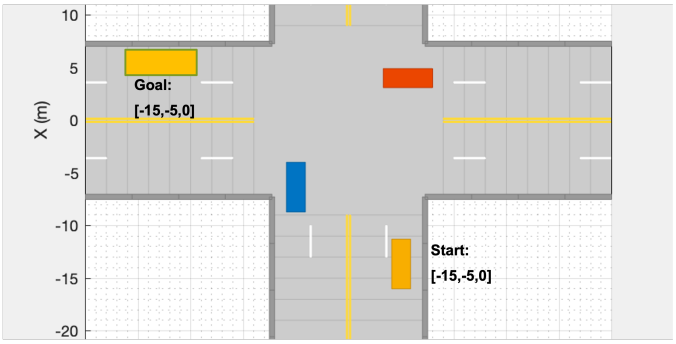

Fig. 1. Scenario 1

The second, Figure 2, depicts a car with obstacles in its linear path.

Lastly, the third depicts a highway scenario with higher speeds and larger turns. All scenarios have barriers on the road edges added as obstacles to further prevent the actor from leaving the road.

B. Training

Each scenario was trained differently to better reflect the circumstances/scenarios. The first scenario has a map that ranges from -25 to 25 in both the x and y directions. As this map is smaller, 10 bins were used to represent the meshes for the cost. The simpler version of Scenario 1, which is shown

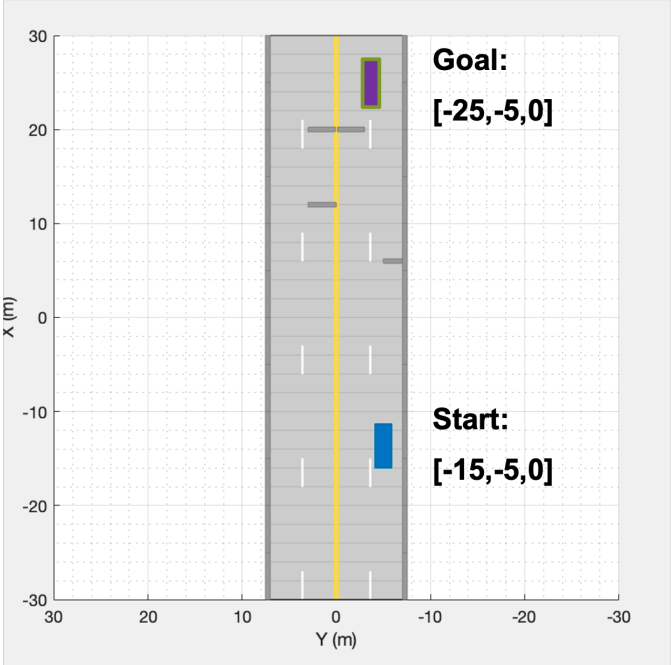

Fig. 2. Scenario 2

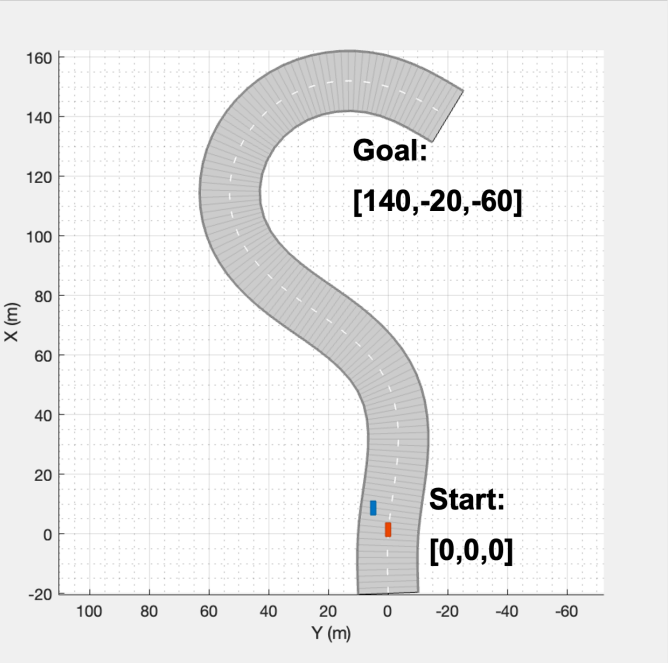

Fig. 3. Scenario 3

in the results section, was trained using only an action space of 3 $(-\frac{\pi}{4},0,\frac{\pi}{4})$ testing with a constant velocity of 1 m/s for initial testing of the network. The more complex version of Scenario 1, with the extra car, was trained using 10 bins for the action space from $[-\frac{\pi}{4},\frac{\pi}{4}]$ with a constant velocity of 1 m/s. The extra action spaces were utilized in hopes of the agent being able to make more precise turns. The second scenario was trained using an action space of 3 $(-\frac{\pi}{4},0,\frac{\pi}{4})$ and a map of $[-30,30]$ for x and $[-10,10]$ for y with 10 bins each. Both of the first two scenarios were trained using 60 neuron intermediate layers. Scenario 3 was a much larger map, requiring x to range from 0 to 160 while y was from -50 to 50. Therefore, more bins were used to represent this space, which ranged from 25 to 50 bins. Furthermore, the action space consisted of 10 bins from $[-\frac{\pi}{4},\frac{\pi}{4}]$ and constant velocity was increased to 5m/s. This scenario was trained using 256 neurons in the intermediate layer. All scenarios were trained using a dt of 1, as smaller discrete times seemed to inhibit the learning of the agent. All scenarios were also trained for 100 episodes, however, the smaller maps, such as Scenario 2, had a lower number of maximum simulation steps.

C. Results

The figure below shows the trajectories outputted by the network on a simpler version of Scenario 1 with only one true obstacle, the car. For a simple task, such as just turning left, the convergence of the training was quite good and the optimal trajectory could be executed in under 5 seconds at a speed of 1 m/s.

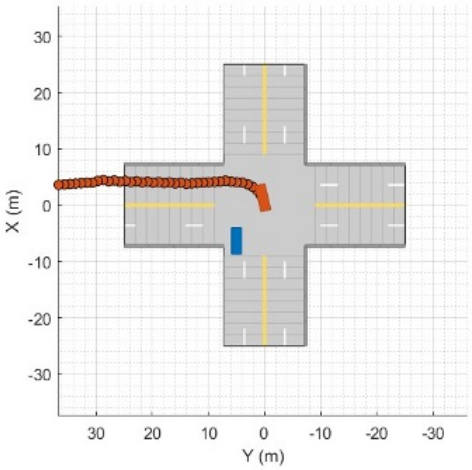

Fig. 4. Simpler Scenario 1 with trajectory output

However, convergence became a lot harder with the addition of another obstacle.

As shown by the resulting trajectory in Figure 6, the actor was not correctly avoiding obstacles, therefore we attempted to increase the distPenalty to increase the penalty of collisions. This resulted in the actor attempting to avoid getting near obstacles all together and did not reach the goal state, as shown in Figure 7.

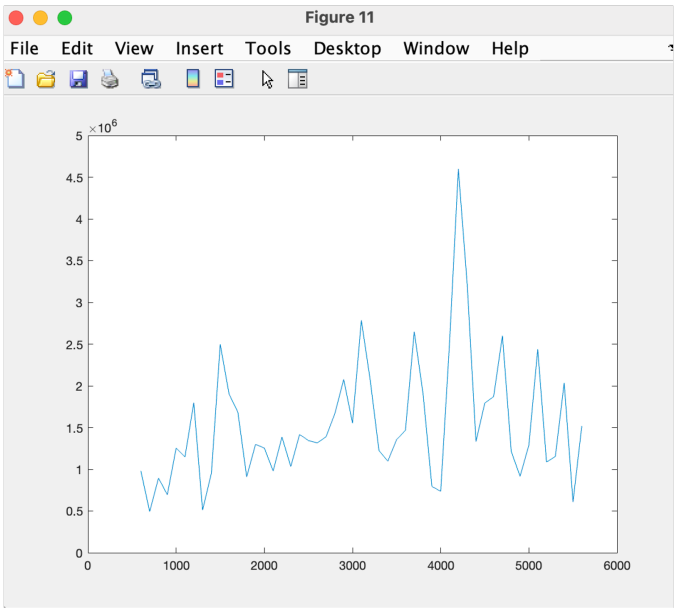

Fig. 5. Scenario 1 Cost per Iteration

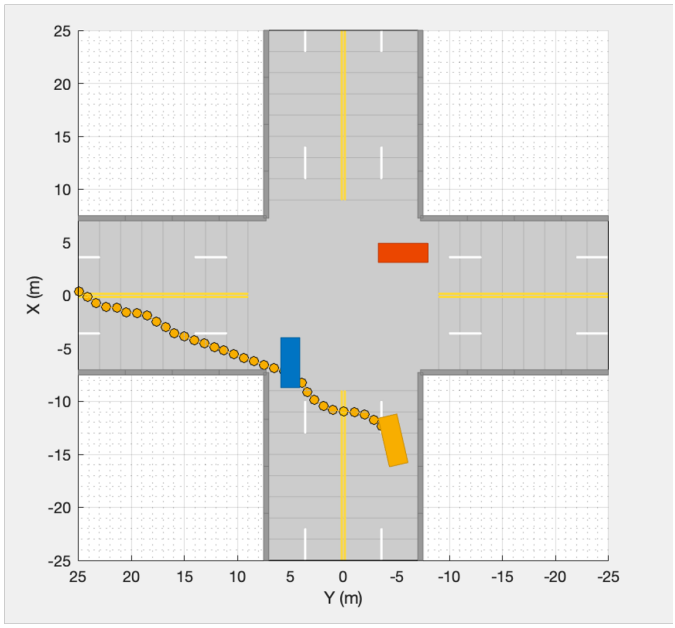

Fig. 6. Scenario 1 Trajectory with low distPenalty

For Scenario 2, the agent was able to mostly find the optimal trajectory (Figure 8), as it took the shortest path to the goal and avoided the first two obstacles; however, it collided with the last barrier before reaching the goal state. The agent was able to execute this trajectory in around 12 seconds and did it successfully without collisions once.

Scenario 3 proved the most difficult for the agent to converge due to the size of the map it was attempting to explore, as well as issues with the roadPenalty, which are further addressed in the future work section.

As shown in Figure 10, the car does not stay on the road

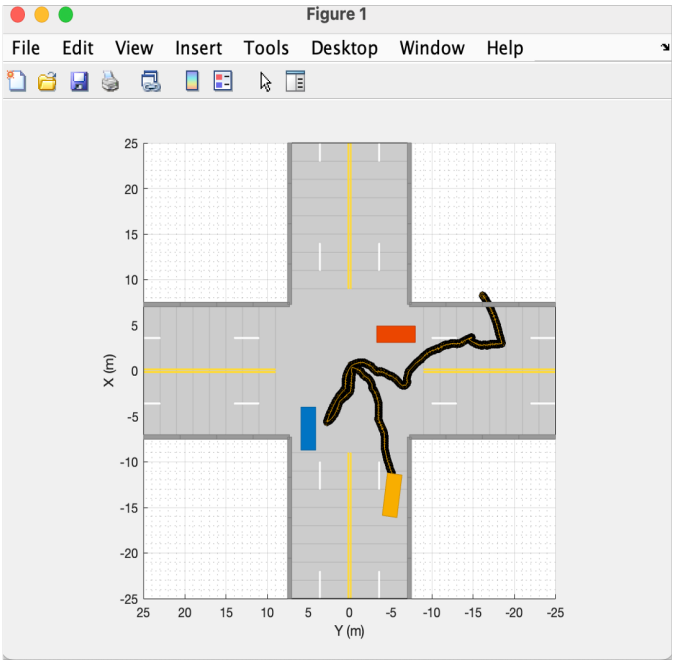

Fig. 7. Scenario 1 Trajectory with high distPenalty

in this scenario and always seems to find a way to get to the lower right corner of the map. This issue is thought to be caused by the distPenalty. As the penalty is higher the closer the agent gets to obstacles, the agent attempts to avoid this penalty by finding a spot that is farthest away from all obstacles. Additionally, once it has gone off the road, the agent will need to collide with the barriers once again to get back on the road, which would increase costs. Therefore, the agent chooses to remain off the road rather than re-enter it.

V. Future Work

To improve this work in the future, we would primarily try to fix the rewards functionality of the code. Currently, the Deep Q-learning model seems to converge best when only utilizing the distPenalty as a factor for rewards. This is largely due to the methodology used for defining the road boundries not being able to accurately detect which states are within the road. To detect road boundaries, we created a polygon shape from all the barriers around the road, however the polygon created form the barriers was not always able to accurately capture the road. Additonally, problems also arise with the distance penalty as it utilizes the distance from the centers of the object to detect collisions rather than bounding boxes. The combination of these issues results in some states being wrongly classified as high cost. In order to address these issues, a sensors could be implemented on the car to help with road and collision detection. This would help improve convergence as there would be less error in the costs calculated and the rewards will more accurately reflect the agent's chosen actions for each state. Furthermore, we would attempt to implement live learning into simulations after finishing the training to see if a better policy can be found. The live

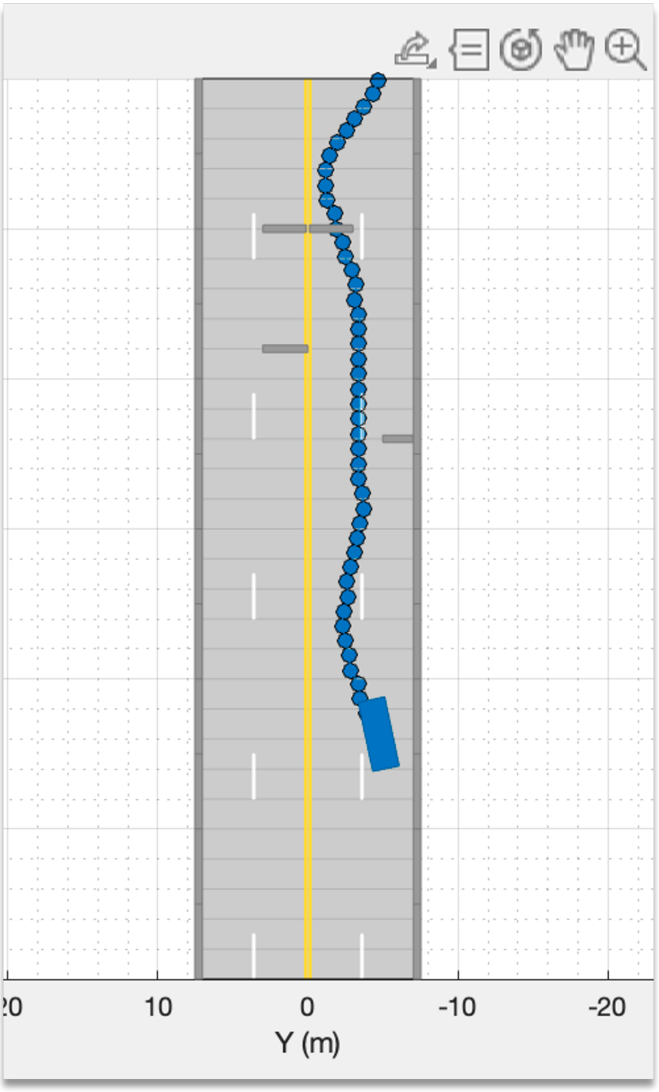

Fig. 8. Scenario 2 Trajectory with only distPenalty

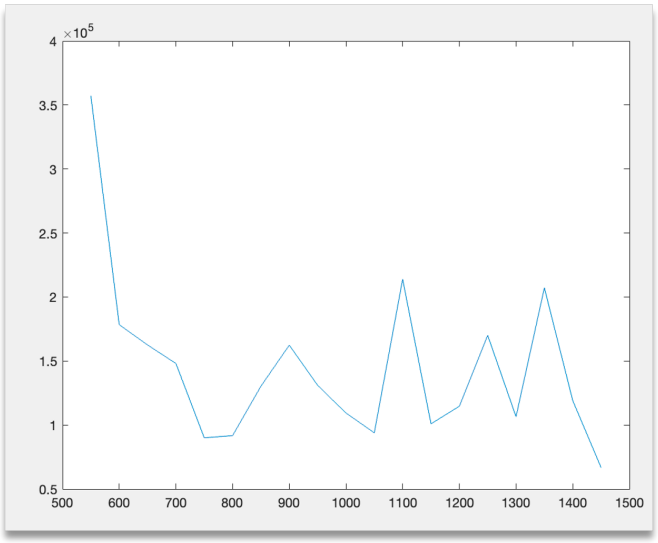

Fig. 9. Scenario 2 Cost Per Iteration

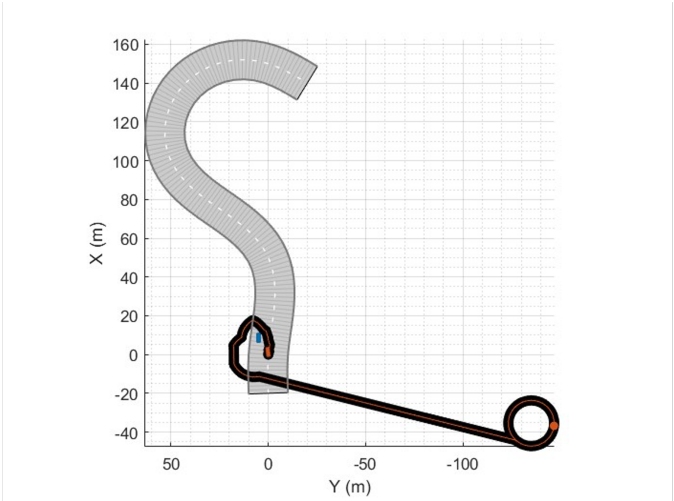

Fig. 10. Scenario 2 Trajectory with only distPenalty

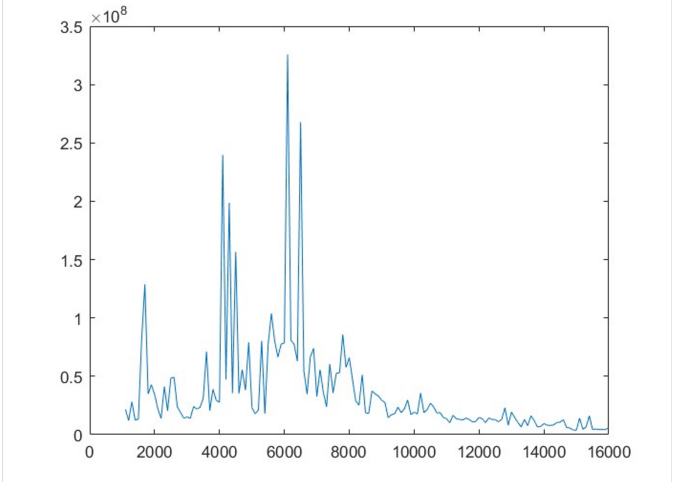

Fig. 11. Scenario 2 Cost Per Iteration

simulations would carry greater penalties for mistakes which could aid in refining the Q-values and choosing appropriate actions. The implementation of both of these features would greatly improve the learning of the agent in more complex situations and make it more more robust for expansion to other simulations such as CARLA.

VI. Conclusion

In this study, we show how Deep Q-Learning can be utilized for obstacle avoidance and car control. While the approach was successful in simpler scenarios, it faced challenges in more complex scenarios with staying on the road and avoiding collisions. By addressing the current issues within this research, a much stronger Q-Learning agent can be built that is more robust and accurate. Tackling the problem of autnomous driving can greatly help increase safety during driving while also automizing a mundane task to save time.

REFERENCES

[1] "How Much Time Do Americans Spend Behind the Wheel? Volpe National Transportation Systems Center." [Online]. Available: https://www.volpe.dot.gov/news/how-much-time-do-americans-spend-behind-wheel#:~:text=96%20hours%20per%20day%20average,right%20ballpark%2C%E2%80%9D%20Pickrell%20said. [Accessed: May 10, 2023]

[2] "Average Annual Miles per Driver by Age Group," Office of Highway Policy Information, May 31, 2022. [Online]. Available: https://www.fhwa.dot.gov/ohim/onh00/bar8.htm

[3] "Tesla Vehicle Safety Report," Tesla. [Online]. Available: https://www.tesla.com/VehicleSafetyReport. [Accessed: May 10, 2023]

[4] T. Okuyama, T. Gonsalves, and J. Upadhay, "Autonomous Driving System based on Deep Q Learnig," in 2018 International Conference on Intelligent Autonomous Systems (ICoIAS), Mar. 2018, pp. 201-205, doi: 10.1109/ICoIAS.2018.8494053.

[5] 0. Perez-Gil et al., "Deep reinforcement learning based control for Autonomous Vehicles in CARLA," Multimed Tools Appl, vol. 81, no. 3, pp. 3553-3576, Jan. 2022, doi: 10.1007/s11042-021-11437-3. [Online]. Available: https://doi.org/10.1007/s11042-021-11437-3. [Accessed: May 10, 2023]

[6] "Simple Understanding of Kinematic Bicycle Model," Medium, Nov. 22, 2021. [Online]. Available: https://dingyan89.medium.com/simple-understanding-of-kinematic-bicycle-model-81cac6420357. [Accessed: May 10, 2023]