Autonomous Navigation in CARLA Using Reinforcement Learning

Group: Srinivas Peri, Srimathnath Thejasvi Vondivillu

Abstract

The primary goal of this project is to design, implement, and train a reinforcement learning (RL) agent capable of autonomously navigating a vehicle through a variety of urban driving scenarios within the CARLA simulator. The agent will be required to make decisions in real-time that allow it to avoid obstacles, adhere to traffic regulations, and successfully reach predetermined destinations without human intervention. This project aims to demonstrate the feasibility and effectiveness of using RL techniques in solving complex navigation problems in simulated environments that closely mimic real-world conditions.

Introduction

Autonomous vehicles (AVs) represent a significant advancement in automotive technology, with the potential to improve safety, efficiency, and accessibility in transportation. However, developing an AV that can navigate complex urban environments reliably is a challenging problem, requiring the vehicle to make intelligent decisions in real-time based on sensory input. Reinforcement learning, a type of machine learning where an agent learns to make decisions by performing actions and receiving feedback from its environment, presents a promising approach to developing intelligent decision-making capabilities.

In this project, the inputs to our autonomous navigation system comprise observations from various sensors installed on the simulated vehicle within the CARLA environment. These sensors include a camera, and collision sensor, which collectively provide essential information about the surrounding urban environment, such as road layout, vehicle positions, traffic signals, and pedestrian movements. These observations are preprocessed and integrated to form a comprehensive state representation, capturing key features necessary for decision-making, including vehicle speed, distance to obstacles, traffic light status, and lane markings. Subsequently, the reinforcement learning (RL) agent processes this state representation and selects actions from a discrete action space, including accelerating, decelerating, turning left, turning right, and maintaining the current speed. The outputs of our system consist of the actions chosen by the RL agent, which are then executed by the simulated vehicle in the CARLA environment. Additionally, the RL agent receives a reward signal based on its actions and resulting state transitions, with positive rewards incentivizing adherence to traffic regulations, successful navigation to predetermined destinations, and avoidance of collisions, while negative rewards penalize violations and accidents. Through this iterative process, our system aims to

develop an autonomous navigation agent capable of safely and efficiently maneuvering through complex urban driving scenarios.

Figure 1: Image of Carla environment- simulates traffic, pedestrians, traffic lights, and more.

Environment Setup and Interaction

The simulator we chose to implement the reinforcement learning algorithm is CARLA. It is an open-source simulator for autonomous driving research, CARLA provides a rich, realistic set of urban scenarios for training and testing AVs. It offers diverse weather conditions, day and night cycles, and a variety of pedestrian and vehicle traffic scenarios.

The software stack for this project includes Python 3.9 as the primary programming language, CARLA 0.9.15 for simulation, TensorFlow 2.15 and Keras for implementing the Deep Q-Network (DQN) model, OpenCV for image processing, and additional libraries for data handling and manipulation. The agent configuration utilizes an Xception-based neural network architecture modified to output Q-values, with a replay memory size of 5,000 to store experience tuples and a batch size of 16 for network training. Epsilon decay is set to linearly decrease from 1 to 0.001 during training to facilitate the transition from exploration to exploitation. Data collection occurs over episodes lasting 10 seconds or terminating prematurely upon collision. Speed, collision, and reward data are logged for analysis, with images captured from the vehicle's camera serving as inputs to the neural network. This setup forms the foundation for training and evaluating an autonomous navigation agent within the CARLA simulator environment.

CarEnv class handles the connection to the CARLA environment, manages the simulation, and processes the sensor data. It initializes the vehicle and attaches cameras and collision sensors to it. The vehicle is spawned at a random starting position at the beginning of each episode.

A camera sensor (sensor.camera.rgb) is used to capture RGB images from the vehicle's perspective. These images serve as the state inputs to the neural network. A collision sensor (sensor.other.collision) detects collisions, which helps determine negative rewards and episode termination conditions.

Figure 2: Simulating a car in Carla's town - throttle, Steer and brake are controllable parameters.

Policy and Actions

The model we developed uses a deep Q-network (DQN) approach, where an action-value function Q is approximated using a deep neural network. The possible actions for the vehicle are:

0: Turn left with full steering.

1: Move straight with no steering.

2: Turn right with full steering.

The agent selects actions using an epsilon-greedy strategy, which balances exploration (selecting random actions) and exploitation (selecting actions based on the highest predicted Q-values from the model). The value of epsilon decays over time, reducing the frequency of random actions as the agent learns more about the environment.

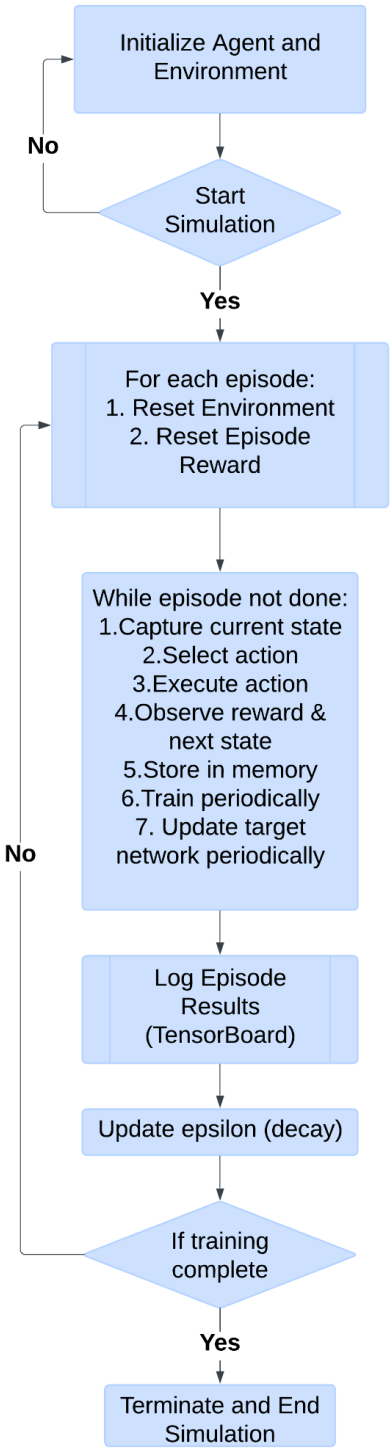

Figure 3: Flow chart of deep Q-learning algorithm

Rewards and Rules

A collision event incurs a substantial penalty of -200, discouraging the agent from actions leading to collisions and emphasizing safety. Additionally, maintaining a speed below 50 kmph

(31 mph) results in a minor penalty of -1, incentivizing the agent to sustain a higher velocity conducive to efficient navigation. Conversely, achieving and maintaining a speed of 50 kmph or higher yields a positive reward of +1, reinforcing the behavior of maintaining an appropriate speed for effective traversal of the urban environment. Through these reward signals, the agent learns to prioritize safe and efficient navigation while adhering to predefined speed thresholds, contributing to the development of a robust autonomous driving system. An episode termination occurs if there is a collision. Also, each episode has a fixed duration (10 seconds as per SECONDS_PER_EPISODE), after which it resets regardless of other conditions.

Model Training and Neural Network Architecture

Neural Network: The model is built using the Xception architecture, modified for regression to output Q-values for each possible action. The network takes an image (state) as input and outputs three values representing the estimated rewards for each action.

Training:

The agent stores experiences (state, action, reward, new state, done) in a replay memory. During training, batches of experiences are randomly sampled from the replay memory to train the model. This process helps in stabilizing the learning and avoiding strong correlations between consecutive learning samples. The network weights are occasionally copied from the prediction model to a target model, which helps in stabilizing training. Training occurs continuously in a separate thread to improve efficiency. OpenCV is to display the camera images if SHOW_PREVIEW is set to True. Images captured from the camera are reshaped and converted to exclude the alpha channel (only RGB is used). These processed images are then normalized and fed into the neural network as input states.

Hypotheses/Expected Results:

Learning Efficiency: The agent will learn to navigate the environment without collisions over time. The hypothesis is that as the number of episodes increases, collisions will decrease, indicating that the agent is learning to avoid obstacles and make safer driving decisions.

Speed Optimization: The agent will learn to maintain a speed close to or above 50 km/h as training progresses. This is hypothesized because the reward structure penalizes low speeds, thus the agent should learn to accelerate appropriately to maximize rewards.

Generalization: The agent trained in diverse conditions (different maps and weather scenarios) will perform better in unseen environments compared to one trained in a uniform setting. This would test the agent's ability to generalize from its training and adapt to new situations.

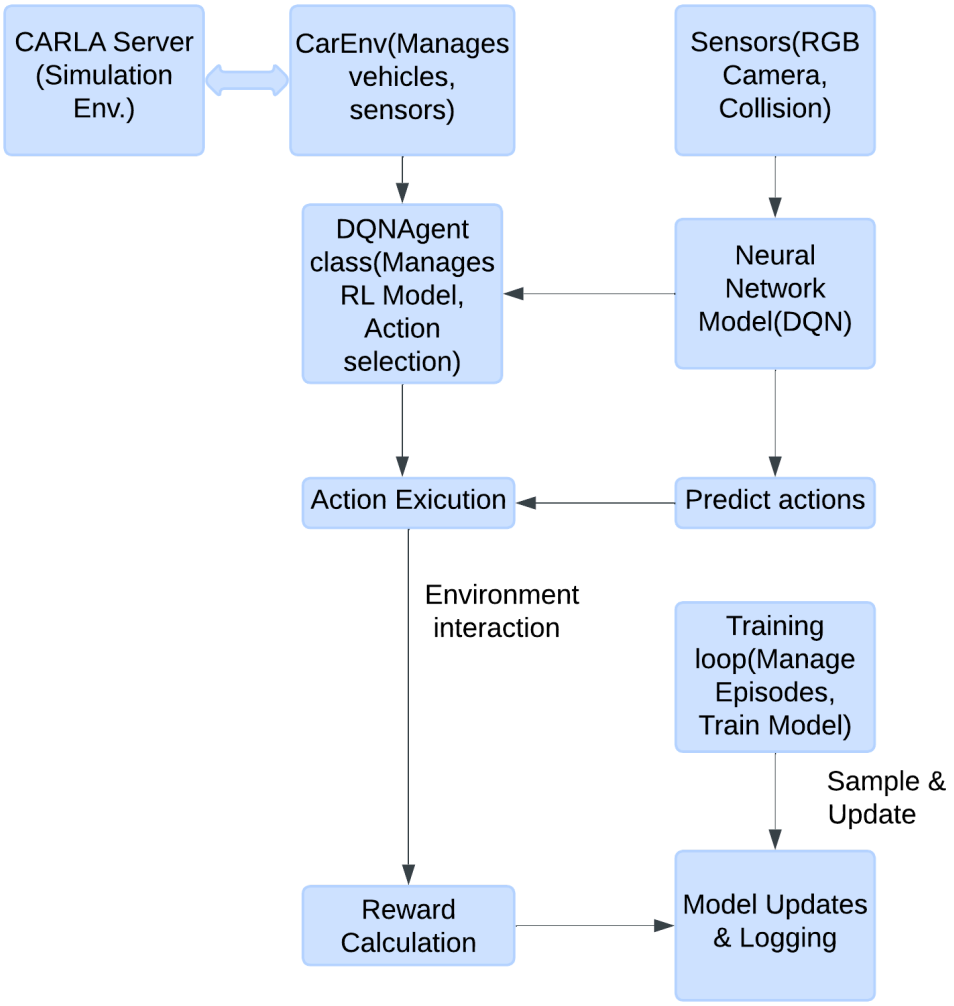

Figure 4: Flow chart of all components of the project

Analysis of empirical results:

The results are calculated over 8000 episodes. Plots for the specified parameters are shown below. The car is still going in circles without exploring the surroundings. This is the result of a computational difficulty. The number of frames received by the camera sensor is quite low; when the intended number of frames is 10-20, we are receiving just 1 or 2 frames per second due to system incompatibility.

- At 1-2 fps, the agent's observations of the environment are sparse and delayed. This can lead to decisions based on outdated information, causing the agent to take undesirable actions, like turning continuously around a point.
- A low frame rate means less frequent updates on the environment state, leading to fewer data points and experiences from which the agent can learn.

- Typically, the agent's policy is updated at a frequency aligned with the frame rate as the frame rate is too low, the policy updates are less frequent, slowing down the learning process.
- With a high frame rate, the agent can closely correlate actions with their immediate outcomes.
- A lower frame rate makes assigning rewards (or penalties) to specific actions hard since many possible actions and outcomes are missed between frames.

All of these analysis assumptions are supported by the acquired results, which are detailed below.

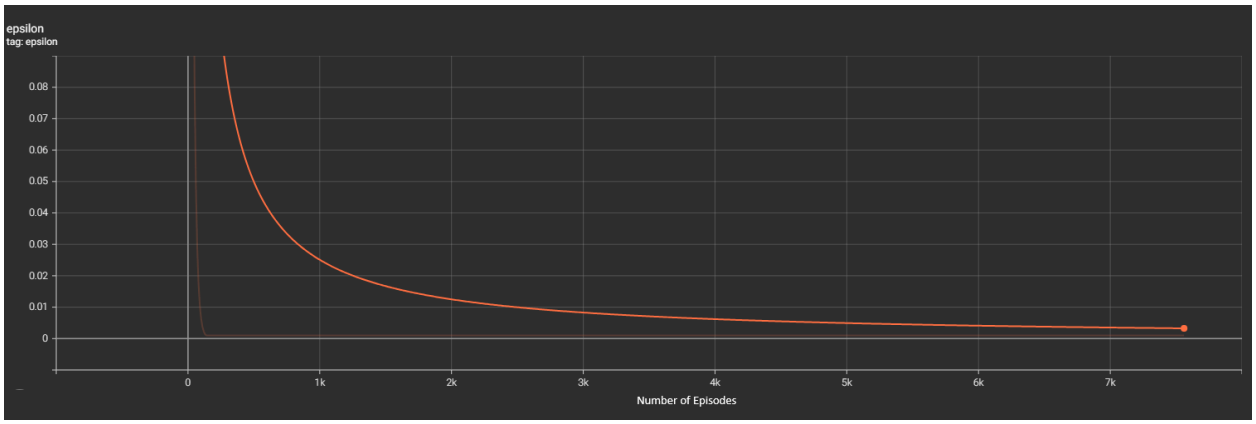

Figure 5: Graph of Epsilon vs Number of Episodes

Figure 5 shows a steep decline initially, which flattens out as episodes increase. This indicates that the agent starts with a high level of exploration and gradually shifts towards exploiting its learned experiences. The flattening of the curve suggests that the decay rate has slowed down significantly, reaching near the MIN_EPSILON value, where it will continue to exploit its learned policy with minimal exploration.

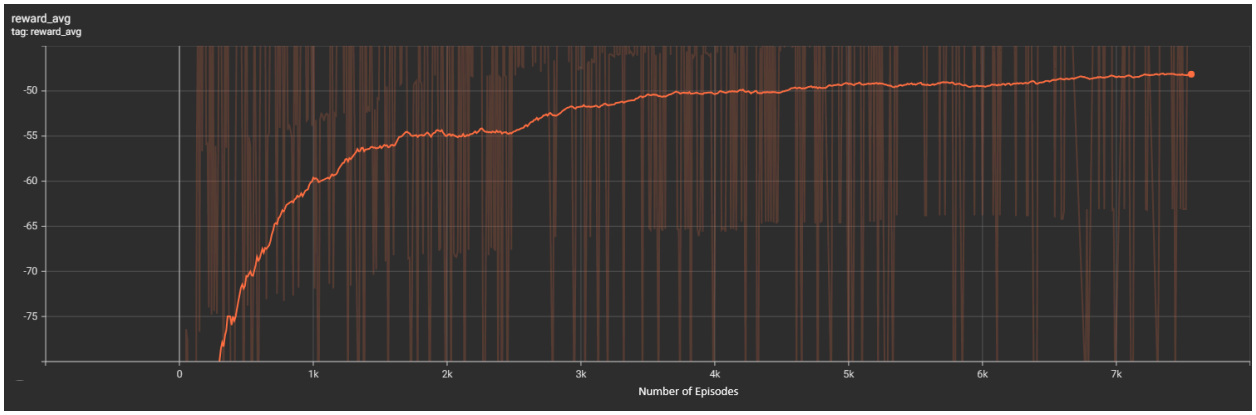

Figure 6: Graph of Average Reward vs Number of Episodes

The trend in the average reward is upward over time, which is a positive sign. It indicates that as the agent learns, its strategy leads to higher rewards, suggesting an improvement in its driving policy. The presence of sharp peaks and valleys, especially in the early stages, suggests significant variability in the agent's performance, which is expected as it explores different strategies. The smoothing of the line over time indicates the agent's policy is becoming more stable and consistently yielding higher rewards.

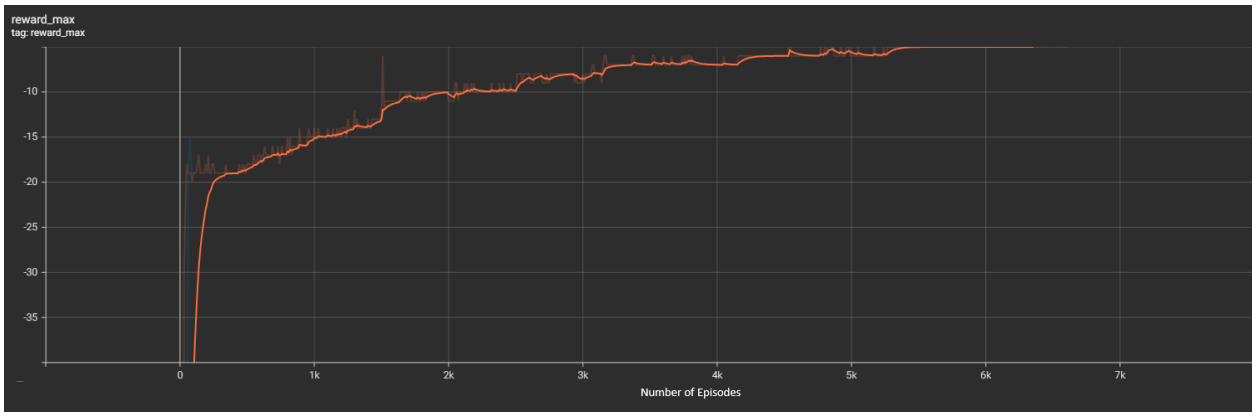

Figure 7: Graph of Maximum Reward Accumulated vs Number of Episodes

The increasing trend in the maximum reward plot suggests that the agent is achieving better performance in at least some episodes as training progresses. This could mean that the agent occasionally executes near-optimal policies, leading to higher rewards. The plateau toward the later episodes may indicate that the agent has reached a performance ceiling with its current setup or that the maximum potential reward for the given environment and task has been achieved. However, as previously stated, the agent adopted a suboptimal policy of going in circles to avoid collisions and eventually ceased exploring.

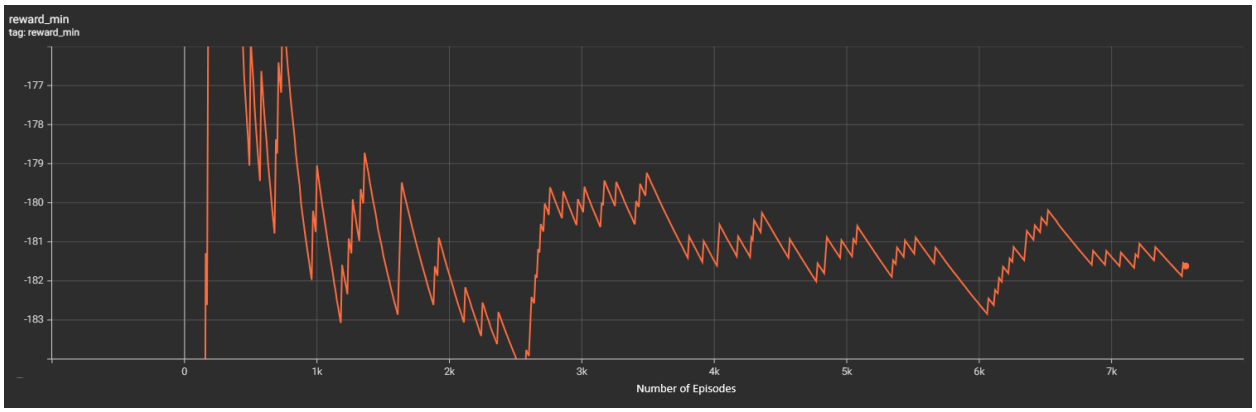

Figure 8: Graph of Minimum Reward over each episode vs Number of Episodes

The minimum reward is important because it shows the worst-case performance in any episode. The upward trend in the minimum reward indicates that the agent is making fewer catastrophic mistakes (like collisions) as time goes on. However, the overall values are quite low, suggesting that there are still episodes where the agent performs poorly. The goal would be to see this value increase, minimizing the negative outcomes.

Difficulties Encountered:

One issue we faced early into testing is that the agent will collide as soon as the episode starts because it spawns in an unsuitable area. So we had to make sure it spawns on the road and in a place/time without any vehicles nearby.

Another difficulty was fine-tuning the hyperparameters of the DQN model and agent configuration to achieve optimal performance. Additionally, ensuring the stability of the training process, especially in complex urban driving scenarios, required careful attention to detail and experimentation with different training strategies.

Given more time to spend on the project, further enhancements could be made in several areas. Firstly, expanding the complexity and diversity of urban driving scenarios to encompass a broader range of challenges, such as navigating crowded intersections and handling diverse traffic patterns, would provide a more comprehensive evaluation of the agent's capabilities. Additionally, exploring advanced RL techniques, such as hierarchical reinforcement learning or incorporating curriculum learning strategies, could potentially enhance the agent's learning efficiency and performance in complex environments. Training for longer episode length and larger episodes might lead to more favorable results.

Future Work:

New Reward parameter:

- A reward function that proportionately rewards the distance traveled in each episode. This would stop the agent from continuously moving in circles and thereby stop exploiting the rewards, i.e., by not colliding.

Optimize Simulation Performance:

- Adjust simulation settings to increase the frame rate, such as reducing the visual fidelity or simulation complexity.

Frame Skipping and Action Repeating:

- Implement frame skipping where the agent repeats the same action for multiple frames.
- This is common in environments where the state does not change significantly between each frame.

Conclusion

In summary, this project demonstrates the feasibility and potential of using reinforcement learning to tackle complex navigation tasks in urban environments. By leveraging the tools and techniques outlined here, future work can build upon this foundation to create even more sophisticated and capable autonomous driving systems by experimenting with algorithms.

References

1. Hossain, J. (2023). Autonomous Driving with Deep Reinforcement Learning in CARLA Simulation. *ArXiv*. /abs/2306.11217
2. Perez-Gill, Oscar & Barea, Rafael & López-Guillén, Elena & Bergasa, Luis & Gómez-Huélamo, Carlos & Gutierrez, Rodrigo & Díaz, Alejandro. (2021). Deep Reinforcement Learning based control algorithms: Training and validation using the ROS Framework in CARLA Simulator for Self-Driving applications. 1268-1273. 10.1109/IV48863.2021.9575616.
3. Rodrigo Gutiérrez-Moreno, Rafael Barea, Elena López-Guillén , Javier Araluce and Luis M. Bergasa. (2022). Reinforcement Learning-Based Autonomous Driving at Intersections in CARLA Simulator. doi.org/10.3390/s22218373

Github Repository Link:

https://github.com/MAHASHANA/RL_Final_Project_Crala_SeflDriving