2023 8th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS) | 978-1-6654-5530-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/MT-ITS56129.2023.10241376

CarAware: A Deep Reinforcement Learning Platform for Multiple Autonomous Vehicles Based on CARLA Simulation Framework

Túlio Oliveira Araújo, Marcio Lobo Netto, and João Francisco Justo

***Abstract*—To facilitate studies in Deep Reinforcement Learning (DRL) and autonomous vehicles, we present the CarAware framework[1] for detailed multi-agent vehicle simulations, which works together with the open-source traffic simulator CARLA. This framework aims to fill the gap identified in currently available CARLA DRL frameworks, often focused on the perception and control of a single vehicle. The new framework provides baselines for training DRL agents in scenarios with multiple connected autonomous vehicles (CAVs), focusing on their sensors' data fusion for objects' localization and identification. These features and tools allow studying many different DRL strategies and algorithms, applied for multi-vehicle sensors' data fusion and interpretation.**

***Index Terms*—Connected Autonomous Vehicles, Deep Reinforcement Learning, Simulation Framework**

I. Introduction

Despite recent developments in sensors and processing hardware, a completely reliable autonomous vehicle is still elusive, as indicated by recent events and accidents involving semi-autonomous vehicles [1], [2]. Therefore, manufacturers still have concerns to increase the automation levels of their vehicles, such that additional research and development are required. The amount of data generated by sensors is enormous, and a fast and meaningful analysis in real-time is still unattainable.

Studies have indicated that vehicle connectivity could be used to enhance the vehicle environment perception, by data shared by the vehicles with any other meaningful source of traffic information (V2X: Vehicle-to-everything). Those studies have been carried by both companies [3] and academic institutions [4]. With recent advances in internet of things (IoT) technologies, 5G mobile networks, edge/cloud computing, and other connectivity technologies, this solution became feasible and closer to reality.

Machine learning techniques are commonly used for perception tasks, requiring huge datasets from sensors covering most of the situations that the vehicle faces regularly (for supervised training) or many hours of trials-and-errors, which cannot be performed in real vehicles due to the risks of accidents (for reinforcement learning). Traffic simulators represent a viable alternative for the development of autonomous driving, as they allow realistic simulation of vehicles' dynamics, traffic conditions, urban or rural environments, vehicles' sensors, and most of the elements needed to train and test machine learning techniques used in those automotive systems.

This work presents a new framework for highly-detailed multi-agent vehicle simulation, labeled CarAware - CARLA Awareness Framework for Deep Reinforcement Learning. It has been implemented on top of the open-source traffic simulator CARLA, due to its realistic simulation, configurable environments, and open-source nature. This framework allows customizing the number and models of simulated vehicles easily, with sensors and their configurations, pedestrians and their behavior, obstacles, maps, weather, and many other important driving characteristics. The vehicles are automatically driven over randomly predefined routes (without real autonomous controls implemented) throughout the map, generating data from their sensors. A top-view visualization of the map was also created, with high-level information on the simulated objects and main metrics used for training and evaluation of DRL techniques. An episodic simulation was set up, with the possibility of randomized simulation resets for an enhanced DRL training generalization. A DRL algorithm using Proximal Policy Optimization (PPO) [5] was implemented, for online training (without saving sensors' data to the hard drive), focusing on predicting the vehicles/pedestrians' positions based on sensors' inputs on a cloud-based collective perception scenario, which could be used as an example for other DRL implementations.

II. Background

A. Overview of Traffic Simulators

Research on autonomous driving, performed in urban areas with test vehicles, has been limited by the risks and costs involved. Moreover, the only way to evolve the vehicle's algorithms, finding the missing gaps in the equipment or testing field, is to keep designing, testing, and studying the results intensively. In addition, to gather a meaningful amount of data, a wide range of tests should be performed, with many different vehicles. Performing those experiments in real systems is impractical.

Those shortcomings motivated the development of several simulation tools, focused on different aspects of autonomous driving. Those tools allow running a large number of test simulations that would be inaccessible in real systems. On the other hand, the main concern is that simulations must mimic real systems in realistic environments and situations so that

[1] https://github.com/tulioaraujoMG/CarAware

978-1-6654-5530-5/23/$31.00 ©2023 IEEE

Simulators	Model Scope	Category	Visual	Virtual Vehicle Sensors	Pedestrians	Graphics	Flexibility	Multi-Agent
AIMSUN	Micro / Meso	Comercial	3D	None	Individual	Low	High	Yes
AirSim	Nano / Micro	Open-Source	3D	GPS / IMU / LIDAR	Individual	High	High	Yes
ARCHISIM	Micro / Meso	Comercial	2D	None	None	Low	Low	Yes
Carcraft (Waymo)	Nano / Micro	Comercial	3D	GPS / Camera / IMU / LIDAR / Radar / Ultrassonic	Individual	High	Medium	Yes
CARLA	Nano / Micro	Open-Source	3D	GPS / Camera / IMU / LIDAR / Radar / Ultrassonic	Individual	High	High	Yes
CORSIM	Micro / Meso	Comercial	2D	None	Crowd	Low	Low	Yes
DeepDrive	Nano / Micro	Open-Source	3D	GPS / Camera / LIDAR / Radar	None	High	Medium	No
HELIOS++	Nano	Open-Source	3D	LIDAR	None	Medium	High	No
MATSim	Micro / Meso	Open-Source	2D	None	Crowd	Low	High	Yes
NVIDIA Drive	Nano / Micro	Open-Source	3D	GPS / Camera / IMU / LIDAR / Radar / Ultrassonic	Individual	High	Low	No
Paramics	Micro / Meso	Comercial	3D	None	Impacts only	Low	Medium	Yes
SimTraffic	Micro	Comercial	3D	None	Individual	Low	Low	Yes
SUMO	Micro / Meso	Open-Source	2D	None	Individual	Low	High	Yes
TRANSIMS	Micro / Meso	Open-Source	2D	None	Impacts only	Low	High	Yes
TransModeler	Micro / Meso / Macro	Comercial	3D	None	Impacts only	Medium	High	Yes
Udacity Car Sim	Nano / Micro	Open-Source	3D	GPS / Camera / IMU / LIDAR / Infrared	None	Medium	Low	No
VISSIM/VISUM	Micro / Meso / Macro	Comercial	2D / 3D	None	Individual / Crowd	Medium	Medium	Yes
SimSonic	Nano	Open-Source	2D	Ultrassonic	None	Low	Low	No

Table I
COMPARISON OF FEATURES INCORPORATED IN AUTOMOTIVE SIMULATORS.

the results are meaningful. There are several traffic simulators available for testing AVs, each one with its pros and cons. Some are private tools developed by car companies, focused on autonomous driving, such as CarCraft by Waymo and DataViz by Uber. Others are commercial packages, such as AIMSUN, VISSIM, and VISUM. Finally, there are the open-source ones, such as AirSim, SUMO, and CARLA.

As there are several situations in daily operations, and the limited processing computer power used in those investigations, each simulator generally focuses on a specific scope. It is also interesting to notice that some of these simulators have interfaces for co-simulation with others, as shown in [6], [7], [8]. Table I shows a detailed comparison of some available simulators, classified according to their scope and main features [9], [10], [11]. They are divided according to their scopes:

• Nanoscopic: The focus is on each vehicle's control (accelerating, braking, and steering), and interactions between them with other obstacles (static, pedestrians, among others), including car control algorithms, specifications, sensor reading, programming of perception modules.

• Microscopic: Although they simulate individual vehicles, they focus on the interactions of a group of vehicles, including traffic-flow dynamics, while pedestrian simulations are also possible. Normally, they are used to study driver behaviors and their impacts on the traffic flow.

• Mesoscopic: They simulate a large group of vehicles in a city-level analysis and present low detailed interactions between them and other objects. The focus is on general traffic evaluation, traffic lights' cycle periods, and stop-and-go waves, among others.

• Macroscopic: They simulate collective vehicle dynamics and general impacts from public transportation. Used for traffic-flow studies in large areas.

B. CARLA Simulator

CARLA is an open-source simulator for urban driving [12]. It could be classified within the nanoscopic category, but it could be also used for some microscopic applications. One of its strengths is the realistic simulation, associated with vehicles' dynamics, pedestrian representation, highly detailed graphics, and a large number of miscellaneous objects represented in the simulated cities. The simulator can also use scripts and scenario specifications, for specific test cases, and supports a wide range of automotive sensors, providing realistic outputs to be used as inputs in an autonomous vehicle's control logic.

All user-created algorithms are implemented in a client module, via an API in Python. Unlike most of its competitors, it doesn't present a user interface to facilitate the process of simulation setup, visualization, and control. This is one of the main goals of the framework presented here.

C. Simulation Frameworks Available for Carla

Due to the open and highly-customizable nature of CARLA Simulator, as well as the absence of manufacturer user interfaces with quick-to-setup tools, there are many proposals of simulation frameworks for it, even involving other traffic simulators.

In [8], a framework has been proposed to integrate CARLA with another popular open-source traffic simulator called SUMO. The idea was to combine their strengths, CARLA, with its highly detailed sensors and visuals (nanoscopic), and SUMO, with the natural traffic management and trajectory planning algorithms (microscopic and mesoscopic). The project Gym-Carla [13] is a framework that provides the tools to train a single agent to drive a vehicle using DRL in an end-to-end manner (from sensors directly to control). The framework provides all the codes required for DRL in the OpenAI's Gym structure [14].

Despite the functionalities of these frameworks, they still have drawbacks that require different implementations for collective perception DRL research. SUMO [8] focuses only on one simulated autonomous vehicle, and requires high processing power to run two different simulators simultaneously. GymCarla [13] has very good DRL preparations, but focuses only on controlling a single vehicle. To address those limitations, this work proposes a framework with a high level of customization and accessibility, simulating many sensory vehicles, pedestrians, and objects within the same environment, with the possibility of having multiple autonomous and regular vehicles running randomly throughout the map, and

prepared for DRL, focused on multi-vehicle sensors' data analyses.

III. Simulation Framework

A. Modules Description and Their Relationship

The proposed framework provides a simulation environment for DRL implementations related to the perception aspects of connected and autonomous vehicles (CAVs). Training and playing modes were implemented to create the real-time simulation used during the neural network training, as well as to create the simulation environment used to check if the training was effective. The simulation mode does not have machine learning implementations and can be used for many other needs, such as producing datasets for supervised training. The complete framework comprises several python scripts, each one with its predefined role in the simulation system. Figure 1 presents a block diagram, showing the scripts and how they relate.

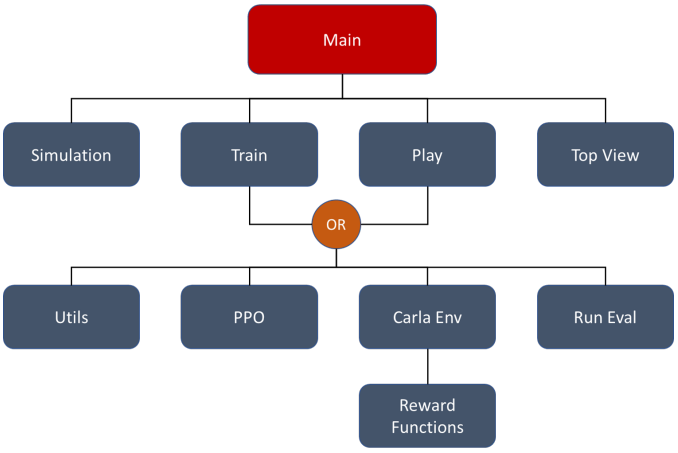

Figure 1. Framework modules and their relationships.

B. Framework's State-Machine Architecture

The framework state machine diagram is shown in figure 2, showing the three execution modes available and their internal states. The "Training Mode" is used to setup/run the simulation and the DRL training episodes, conducting both activities in synchronized parallel threads. The "Play Mode" is used to setup/run indefinitely a simulation in parallel with a neural network providing predictions using a previously trained model. The "Simulation Mode" is used to setup a simulation-only session, without DRL implementations.

C. Implemented Sensors and Related Data Interpretation

CARLA Simulator is a powerful tool when it comes to the sensors available in its simulated environment, including global navigation satellite system (GNSS), camera, LIDAR, and radar, among others. A significant feature offered in this framework is the ability to gather sensor observations from multiple vehicles in the environment, providing a sensor-based collective perception, useful for studies on connected vehicles. The main sensors implemented in this framework include their

Figure 2. State machine diagram of the framework.

data processing, for easy interpretation by the neural networks implemented in the DRL algorithms.

1) GNSS: CARLA Server provides a blueprint for creating GNSS sensors and applying them to simulated vehicles or pedestrians (for smartphone simulation). It has many customizable configurations, such as sampling frequency, bias, and induced error simulations (injection of a standard deviation noise), available as configuration variables at the beginning of the "Main" script. The sensor outputs the data in a geographic coordinates system [15], with latitude and longitude values in degrees. This is obtained by adding its metric position to an initial geographic reference defined in the simulated map, following the OpenDRIVE map definition [16]. This reference simulates a real location as if the simulated map was actually within the globe. To convert these coordinates into the standard needed to position the objects in the top-view window, a few steps of calculation are required, as follows:

• Equation 1: Converts "latitude/longitude" sensor output data, from geographic coordinates, to "x/y" ECEF (Earth-Centered-Earth-Fixed) system.

$$\begin{aligned} x &= R_{earth} \cdot \cos(lat) \cdot \cos(long) \\ y &= R_{earth} \cdot \cos(lat) \cdot \sin(long) \end{aligned} \tag{1}$$

• Equation 2: Converts the results of the previous equation, from ECEF coordinates to East-North-Up (ENU) system. The earth center is chosen as a reference point for this conversion (with $\lambda = lat_o$ / $\phi = long_o$ being its center location in geographic coordinates and x_o / y_o / z_o its center location in ECEF).

$$\begin{aligned} x_{east} &= -\sin\phi \cdot (x - x_o) + \cos\phi \cdot (y - y_o) \\ y_{north} &= -\cos\phi \cdot sin\lambda \cdot (x - x_o) - \sin\lambda \cdot \\ &\sin\phi \cdot (y - y_o) + \cos\lambda \cdot (z - z_o) \end{aligned} \tag{2}$$

• Equation 3: Converts the position in ENU coordinates to the Pygame's coordinates system, including scale, resolution (pixels per meter), and position translations applied to the top-view visualization. The resulting position is then drawn in the shape of blue dots in the top-view window.

$$\begin{aligned} x_{pygame} &= scale \cdot resolution \cdot (x - x_{offset}) \\ y_{pygame} &= scale \cdot resolution \cdot (y - y_{offset}) \end{aligned} \tag{3}$$

2) IMU: CARLA also provides a blueprint for the Inertial Measurement Unit (IMU) sensor. This sensor is composed of accelerometers and gyroscopes, providing the linear acceleration measurements in m/s^2, and the angular velocity in rad/s, in all axes (six degrees of freedom). It also contains a compass, which indicates the direction the vehicle is heading, in radians.

3) Camera: Out-of-the-box, CARLA provides four different types of camera sensors: regular RGB (Red, Green, and Blue), semantic, dynamic vision sensor (DVS), and optical flow. In order to increase the features provided by the cameras, two other types of image sensors were implemented in this framework. The binary camera sensor was created, to improve even more the object identification capabilities of the neural networks, by highlighting in white only the interesting objects: vehicles, pedestrians, and specific static objects. Moreover, the object identifying pre-trained neural network model YOLO was also made available to be used in this framework [17], providing object tracking capabilities. The frame stacking technique was implemented, following an example described by Takeshi [18], where a group of four sequential images is used to transmit the sense of motion to the neural network.

4) LIDAR: This sensor simulates a rotating LIDAR implemented using ray-casting. It creates a point cloud, with each point representing where the laser has hit an object in the simulated environment. In the framework, it is possible to configure the number of generated points per step, number of sensor channels, rotation frequency, maximum range, visualization type, and more.

Regarding the LIDAR sensor visualization in the top-view window, there are two modes available: "All" and "Interest". The "All" mode plots all generated points around each represented vehicle. As it demands high processing power and can be difficult to analyze, the "Interest" mode was also implemented. In this mode, only the interest points are shown in the top-view: The ones that hit vehicles, pedestrians, or specific static objects, each type with a different color. When an object is detected, a circle is also drawn showing the sensor's maximum range.

The coordinates generated by the LIDAR sensor are relative to the sensor's orientation, being the "Y" axis parallel to the vehicle's heading direction and pointing backward, "X" axis perpendicular and pointing to the vehicle's left side, and "Z" starting in the sensor's origin and pointing to the ground. This representation is shown in figure 3.

To convert this relative orientation to the static orientation used in the top-view screen, a function was created to perform this conversion using the Cartesian rotation of axis equation, shown in Equation 4. In the equation, (x, y) represents the position in the top-view absolute plane, $(\bar{x}, \bar{y})$ the position in the LIDAR relative plane, and θ the rotation angle between both planes.

$$\begin{aligned} x &= \cos\theta \cdot \bar{x} - \sin\theta \cdot \bar{y} \\ y &= \sin\theta \cdot \bar{x} + \cos\theta \cdot \bar{y} \end{aligned} \tag{4}$$

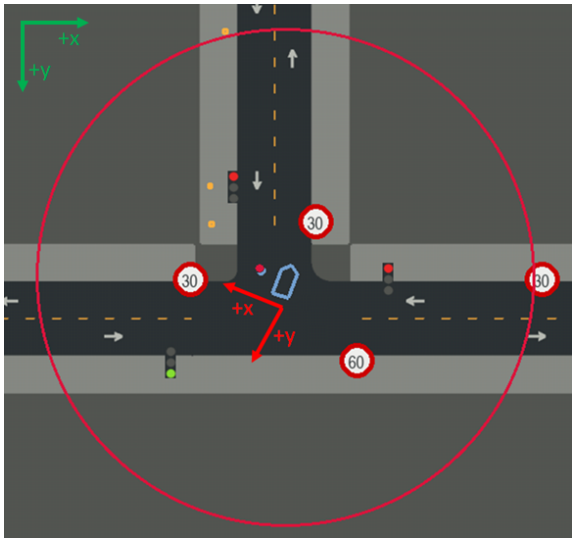

Figure 3. LIDAR coordinates system (red) vs. Pygame coordinates system (green).

5) Speed and Steering Angle Sensor: A new sensor was created on the client-side (in the framework's "Simulation" script), including regular sensors' structures like customizable characteristics, required for many analyses. The steering wheel angle value is acquired from CARLA in the range from -1 to 1, converted to steering rotation values in degrees.

IV. Case Study Analysis: V2X Sensors to Vehicle Localization With DRL PPO

In order to show the framework capabilities, a case study DRL training session was performed. The goal was to collect sensor data from the GNSS and IMU sensors in multiple simulated vehicles and infer correctly the position of all vehicles in the top-view window. These sensors were chosen since they are commonly used in connected vehicles with embedded V2X communication systems, transmitting their data to others via Basic Safety Messages (BSM) [19]. The concept diagram for this proposed DRL implementation is shown in figure 4.

For the proposed task in this case study, the applied algorithm was the policy-based Proximal Policy Optimization (PPO). Each existing DRL algorithm is best fit for certain groups of applications (figure 5) [20]. The PPO algorithm was released in 2017 [5], which is still one of the most efficient DRL algorithms being applied to complex environments with continuous observation and action spaces. Its central idea is to avoid having too large policy updates, by implementing a clipped surrogate objective function (GAE estimator is commonly used), which will constrain the policy changes within a small predefined range of change ratio (normally from 0.8 to 1.2).

For the case study application, the agent was structured to get observations, and provide actions and rewards, on one vehicle per training step, cycling through all of them continuously. The observation space was defined as a one-dimensional vector

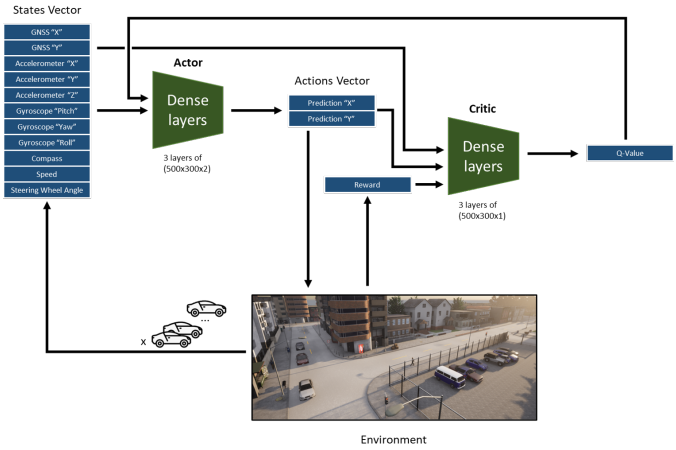

Figure 4. Case study DRL concept diagram.

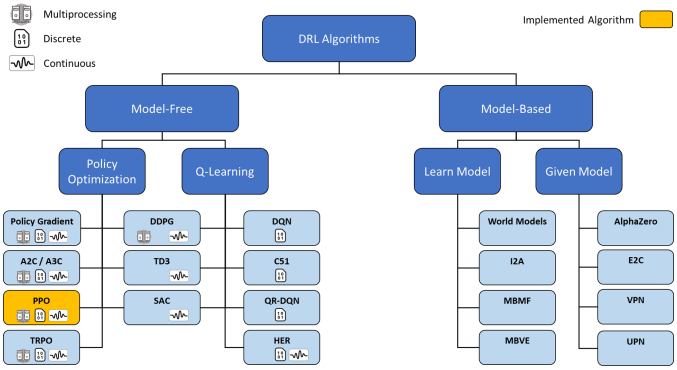

Figure 5. Main DRL algorithms, their features and classification.

with nine continuous elements, representing each simulated vehicle's inputs in the scenario. The action space was defined as a one-dimensional vector of two continuous elements, representing each simulated vehicle's prediction. A reward function was defined as the Euclidean Distance between the prediction position and the real simulated vehicle position (passed to the function as a ground-truth data from the simulator), as defined in Equation 5. The reward value is calculated as the inverse of the distance value, as the reward should be higher the closer the prediction is to the vehicle. Before passing the observations and rewards as inputs to the neural network, their values were normalized to be within the 0 to 1 range, to avoid early weight overfitting during training. As a consequence, the outputs had to be restored to their original ranges continually.

$$d_{euclid} = \sqrt{(veh_x - pred_x)^2 + (veh_y - pred_y)^2} \quad (5)$$

For each application, the DRL algorithms require specific tuning of their hyperparameters. This is because each environment presents different levels of complexity, sizes and patterns of states, agent control behavior, and other properties. The most important hyperparameters used by the PPO algorithm are (more details in [5]):

• Episode Number: Number of training episodes performed.

• Learning-Rate: Defines the rate at which the neural network weights are updated upon training. Higher values haste the training process but can cause wrong learning and unwanted behaviors.

• GAE Lambda: Smoothing factor applied to the Generalized Advantage Estimator, the objective function used to clip each policy update ratio.

• Initial Deviation: Defines the initial standard deviation applied in the neural network output. Its value should be high enough to cause important reward value changes, otherwise, the model won't learn the desired behavior.

• Entropy Scale: This factor is responsible for encouraging some exploration by the agent in later training phases.

• PPO Epsilon: Defines the positive and negative policy update clip ratio values.

• Horizon Number: The number of steps of observation-action-reward performed before the resulting training data is fed to the neural network. For complex environments, this value should be higher, so the agent has more data to learn.

• Batch Size: Defines the size of each batch used in the gradient descent training operations. It should be an exact fragment of the horizon number.

• Epoch Number: Defines the number of times the training process is executed with the same data, generated by a complete horizon execution.

Using hyperparameters optimization techniques for complex environments with continuous action spaces, and curriculum learning techniques [21] to slowly increase environments' complexity during training, the agent was able to fulfill the proposed objective. For the case study application, three steps of learning were established:

• Step 1 - Single Moving Agent w/o Restart: It started with just one simulated vehicle equipped with GNSS and IMU sensors, running around the map using randomized routes. The agent was trained under this condition until the standard deviation reached the value of 0.5, in episode 162.

• Step 2 - Single Moving Agent w/ Restart: The agent continued its training, restarting the vehicle position to a randomized spawn point after each episode. With this simulation strategy, the agent could increase its generalization ability due to the sudden position changes that happened with every restart. The agent was trained until the standard deviation reached the value of 0.35, in episode 235.

• Step 3 - Multiple Moving Agents: At last, the agent continued its training with 8 vehicles (randomized spawn points). As the agent analyzes one vehicle at a time, located far from each other, the previous training steps were necessary for the agent to learn to predict all vehicle positions on the map. The training stopped when the predictions converged, and before there was a network overfitting (standard deviation of 0.27, episode 283).

The resulting predictions provided by the neural network after the training are shown in figure 6 (red dots representing the predicted vehicles' positions and blue arrow-like rectangles representing their actual position), and the report showing the training progress of the two main metrics in figure 7.

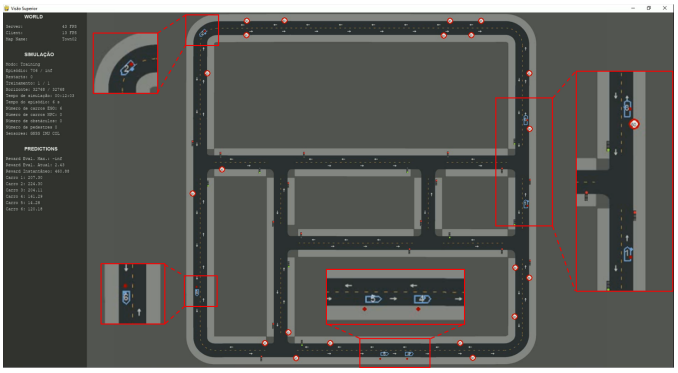

Figure 6. Case study predictions after curriculum-training - Red-dots representing predictions, and blue arrow-like rectangles representing the vehicles' positions.

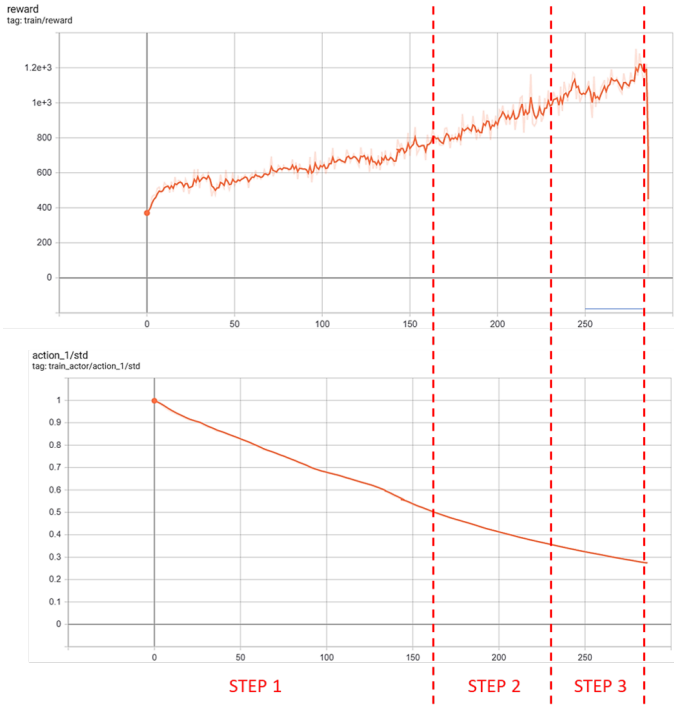

Figure 7. Reward and standard deviation curves, divided by the steps of curriculum learning - Step 1: Single agent without restart; Step 2: Single agent with restart; Step 3: Multiple Agents.

V. Results and Discussions

This framework project was performed to enable the DRL study focused on creating a collective perception methodology. It shows a brief explanation and the first successful results of the case study scenario of V2X data sharing. More difficult training scenarios could be tested in the future. During the development, a more comprehensive toolset for CAVs' simulation was created, so it could also provide the needed sensors' data and complex environment simulations required for the CAVs' complete sensor fusion methodology. In the end, these methodologies could be assigned to real cloud backend systems, collecting data and providing information to running vehicles, improving their capabilities of recognizing their surroundings and therefore, improving the safety of these technologies in overall traffic situations. The development of the framework was focused on DRL due to the study's focus and development time restrictions, but it could be easily improved, to implement other deep learning techniques and additional features.

References

[1] TMC, "Tesla releases statement on fatal model x crash," 2018. [Online]. Available: https://teslamotorsclub.com/blog/2018/03/28/tesla-releases-statement-on-fatal-model-x-crash/

[2] A. J. Hawkins, "A tesla vehicle using 'smart summon' appears to crash into a $3.5 million private jet," 2022. [Online]. Available: https://www.theverge.com/2022/4/22/23037654/tesla-crash-private-jet-reddit-video-smart-summon

[3] Z. Szalay *et al.*, "5g-enabled autonomous driving demonstration with a v2x scenario-in-the-loop approach," *Sensors*, vol. 20, no. 24, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/24/7344

[4] P. Zhou *et al.*, "Aicp: Augmented informative cooperative perception," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–14, 2022.

[5] J. Schulman *et al.*, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[6] D. Nalic *et al.*, "Software framework for testing of automated driving systems in the traffic environment of vissim," *Energies*, vol. 14, no. 11, 2021. [Online]. Available: https://www.mdpi.com/1996-1073/14/11/3135

[7] L. Pariota *et al.*, "Integrating tools for an effective testing of connected and automated vehicles technologies," *IET Intelligent Transport Systems*, vol. 14, no. 9, pp. 1025–1033, 2020. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-its.2019.0678

[8] P. Li *et al.*, "A novel traffic simulation framework for testing autonomous vehicles using SUMO and CARLA," *CoRR*, vol. abs/2110.07111, 2021. [Online]. Available: https://arxiv.org/abs/2110.07111

[9] M. Saidallah *et al.*, "A comparative study of urban road traffic simulators," *MATEC Web Conf.*, vol. 81, p. 05002, 2016. [Online]. Available: https://doi.org/10.1051/matecconf/20168105002

[10] F. Rosique *et al.*, "A systematic review of perception system and simulators for autonomous vehicles research," *Sensors*, vol. 19, no. 3, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/3/648

[11] B. Paterlini, "Assessment of connected and autonomous vehicles impacts on traffic flow through microsimulation," 2021. [Online]. Available: https://www.teses.usp.br/teses/disponiveis/3/3142/tde-09032021-090209/pt-br.php

[12] A. Dosovitskiy *et al.*, "CARLA: an open urban driving simulator," *CoRR*, vol. abs/1711.03938, 2017. [Online]. Available: http://arxiv.org/abs/1711.03938

[13] J. Chen *et al.*, "gym-carla," 2022. [Online]. Available: https://github.com/cjy1992/gym-carla#readme

[14] OpenAI, "Getting started with gym," 2022. [Online]. Available: https://gym.openai.com/docs/

[15] G. Geography, "Latitude, longitude and coordinate system grids," 2021. [Online]. Available: https://gisgeography.com/latitude-longitude-coordinates/

[16] ASAM, "Asam opendrive®," 2022. [Online]. Available: https://www.asam.net/standards/detail/opendrive/

[17] J. Redmon *et al.*, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. [Online]. Available: http://arxiv.org/abs/1506.02640

[18] D. Takeshi, "Frame skipping and pre-processing for deep q-networks on atari 2600 games," 2016. [Online]. Available: https://danieltakeshi.github.io/2016/11/25/frame-skipping-and-preprocessing-for-deep-q-networks-on-atari-2600-games/

[19] R. Miucic, "Connected vehicles: Intelligent transportation systems," 2019. [Online]. Available: https://link.springer.com/book/10.1007/978-3-319-94785-3

[20] F. AlMahamid *et al.*, "Reinforcement learning algorithms: An overview and classification," in *2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, Sep. 2021, pp. 1–7.

[21] X. Wang *et al.*, "A survey on curriculum learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021.