Received 19 November 2020; revised 15 March 2021 and 18 May 2021; accepted 20 May 2021. Date of publication 24 May 2021; date of current version 1 June 2021.

Digital Object Identifier 10.1109/OJITS.2021.3083201

A Reinforcement Learning Framework for Video Frame-Based Autonomous Car-Following

MEHDI MASMOUDI, HAMDI FRIJI, HAKIM GHAZZAI (Senior Member, IEEE), AND YEHIA MASSOUD (Fellow, IEEE)

School of Systems and Enterprises, Stevens Institute of Technology, Hoboken, NJ 07030, USA

CORRESPONDING AUTHOR: H. GHAZZAI (e-mail: hghazzai@stevens.edu)

A part of this work has been published in IEEE International Conference on Vehicular Electronics and Safety (ICVES 2019), Cairo, Egypt, Sep. 2019 [1].

ABSTRACT Car-following theory has received considerable attention as a core component of Intelligent Transportation Systems. However, its application to the emerging autonomous vehicles (AVs) remains an unexplored research area. AVs are designed to provide convenient and safe driving by avoiding accidents caused by human errors. They require advanced levels of recognition of other drivers' driving-style. With car-following models, AVs can use their built-in technology to understand the environment surrounding them and make real-time decisions to follow other vehicles. In this paper, we design an end-to-end car-following framework for AVs using automated object detection and navigation decision modules. The objective is to allow an AV to follow another vehicle based on Red Green Blue Depth (RGB-D) frames. We propose to employ a joint solution involving the You Look Once version 3 (YOLOv3) object detector to identify the leader vehicle and other obstacles and a reinforcement learning (RL) algorithm to navigate the self-driving vehicle. Two RL algorithms, namely Q-learning and Deep Q-learning have been investigated. Simulation results show the convergence of the developed models and investigate their efficiency in following the leader. It is shown that, with video frames only, promising results are achieved and that AVs can adopt a reasonable car-following behavior.

INDEX TERMS Autonomous vehicle, deep learning, reinforcement learning, video frames processing, car-following.

I. INTRODUCTION

OVER the last few years, car-following models have attracted a lot of attention in both research and industrial domains and have witnessed a perpetual evolution since then [2]. Many innovative projects are being undertaken to improve and design new models and approaches of car-following in order to deliver operational and safe technology that will change the pattern of transportation [3], [4]. With the rise of intelligent transportation and autonomous intelligent control systems, different approaches of car-following have been proposed by traffic engineers and transport professionals to simulate and analyze the stability of traffic flow and deal with dynamic and operational traffic problems and traffic security [5], [6].

To date, car-following behaviors have become one of the main research contents on the autonomous vehicle (AV) decision-making. Many automated vehicles, such as Google car, adopt car-following models to control their movements and allow for smoother manipulation [7]. AVs are empowered with sophisticated technology allowing them sensing their surrounding environment and autonomously navigating according to the collected data. This is mainly enabled through diverse types of sensors having vision/non-vision capabilities such as Inertial Measurement Units (IMU), cameras, LiDAR, and RADAR [8]. Based on these inputs, AVs need to accelerate/decelerate, manoeuvre in the environment, and avoid static and moving obstacles. At present and as it will be discussed in details in the next section,

The review of this article was arranged by Associate Editor Xianfeng Yang.

several algorithms have been developed to investigate the car-following behavior of AVs. Most of these studies are based on typical mathematical and control modeling algorithms to ensure smooth car-following such that an autonomous vehicle, defined as the follower, keeps following another vehicle, defined as the leader, while maintaining safety distances [9]–[11]. Recently, few studies have promoted the use of Artificial Intelligence in designing car-following models [12], [13]. Most of them resorted to use Reinforcement Learning (RL) methods to determine navigation decisions for the follower vehicle and hence, design their car-following models based on numerical inputs of the vehicle dynamics, e.g., the lateral position, the speed, and the yaw angle. However, to the best of the authors' knowledge, there is no previous work that proposed RL based car-following framework while making navigation decisions based on video frames processing only.

In this paper, we propose two artificial intelligence algorithms to address the car-following problem for AVs, specifically, RL-based algorithms. We develop an end-to-end car-following framework based on video frames processing. We promote the use of video frames processing for its possibility to extract real-time information with high accuracy and speed. Therefore, we propose to exploit this continuously available data to allow an AV to follow another vehicle. Throughout the rest of the paper, we identify the AV as "the follower", and the leading vehicle as "the leader".

The proposed framework is essentially based on the two following phases:

- A computer vision phase: the follower automatically detects objects in the environment then uses the characteristics of depth images to recognize the leader vehicle and compute the distance and the deviation angle between the leader and the follower by exploiting the information extracted with the real-time object detector: the YOu Look Once version 3 (YOLOv3) algorithm [14].
- An RL phase: an RL algorithm is trained and employed to exploit the extracted features to make real-time navigation decisions that ensure a safe distance with the leader without losing its detection or crashing with other objects.

Two RL algorithms are investigated and tested in the RL phase: the finite-state Q-Learning (QL) algorithm and an improved version of the Deep Q-Network (DQN). We start first by modeling the QL algorithm and adapt to the car-following problem [15]. The QL is a fast-training RL algorithm that we use to enable real-time driving decision making. Afterwards, we investigate the DQN model that we implement with improvements to extract the most significant information from image sensory inputs and combine them with other environmental features. Indeed, unlike the QL algorithm, DQN is capable in handling high-dimensional scenarios and hence, may improve the driving capabilities of the AV in some special cases where the precision of driving is a crucial feature for a safe driving (e.g., narrow roads, dense roads/highways, high density urban area's roads). Our objective is to analyze the autonomous car-following framework using both RL models to ensure the following of a leader-vehicle using only image-based features collected from the front camera of the follower. This is performed while avoiding collision with the leader vehicle and any other actor in the environment without losing the detection of the leader. We consider a common scenario of one leading vehicle and one follower in a straight road such as highway. Afterwards, we extend our study to the case where another vehicle exists and acts as a moving obstacle. In the rest of the paper, we identify the latter vehicle as "the obstacle".

The proposed framework is simulated on the CARLA simulator and trained to navigate in a way to follow a leader-vehicle in several predefined scenarios using seven different maps and four weather conditions, then tested on three different maps. Our experiments and simulation results investigate the object detection technique performance and the behavior of the AV and the decisions taken to follow the leader. For the detection phase, the YOLOv3 model performs well by reaching higher accuracy with a notable ability to detect objects in real-time. For the RL phase, the results show that the models converge after a certain number of training episodes and effectively performs navigation for low speed driving. The purpose of this paper is to show that only with images, the follower is able to efficiently but not at 100% make decisions to follow the leader. The results of the model are very promising for the autonomous car-following approach. It supports our vision to further investigate the problem by involving other environmental actors and different sensors inputs, such as LIDAR and RADAR.

The contributions of the paper can be briefly summarized as follows:

- We investigate the car-following problem by developing an end-to-end AI-driven framework based on video frames processing for obstacle-free and obstacle-aware scenarios.
- We model the car-following algorithm as a two-phased procedure.
  - In the first phase, a computer vision algorithm is executed to allow the follower to automatically identify its position with respect to the leader using RGB-D features.
  - The second phase implements an RL algorithm to autonomously follow the leader.
- We implement the framework on the CARLA simulator and investigate the performance of the proposed approach for different scenarios and versus other computer vision and RL algorithms.

The remainder of the paper is organized as follows. Section II provides a literature review related to the car-following problem. Section III presents the proposed car-following approach methodology. Section IV delves into the computer vision part. Section V presents the used RL algorithms. Simulation results are discussed in Section VI and finally, we conclude the paper in Section VII.

II. LITERATURE REVIEW

Researchers and scientists tackled the problem of car-following to enhance the efficiency of the autonomous driving. Indeed, In 1953, Pipes first presented a car-following technique which only considered the relative velocity between vehicles [16]. The presented technique in [16] investigate the acceleration behavior of the driver as a function of the inter-vehicle separation and relative speed, and it could be improved by considering other features. In [17], Forbes studied the car-following behavior by considering the reaction time needed for the follower to decelerate and apply the brakes. In [18], Ou and Tang developed a car-following model considering Inter-Vehicle Communication (IVC) to examine each of the vehicle movement and to better identify and hence, adjust the driving behavior in a two-lane traffic system when an incident occurs on a lane. In [11], the authors developed a path-following algorithm for AVs. The model is based on road geometry data as well as vehicle dynamics. These analytical models are usually based on deterministic approaches requiring as inputs mathematical parameters reflecting the vehicle dynamics. The models usually require a perfect knowledge about the activity of the leader vehicle, which is practical for real-time decision making as vehicles need to continuously exchange data and this might be subject to connectivity issues and latency effect [19].

In the last few years, the research fields were invaded by Artificial Intelligence (AI) models, and hence some researchers investigated the car-following problem using AI algorithms to increase the performance of car-following based autonomous driving. In [20], the authors proposed a driving style recognition method based on vehicle trajectory data to avoid rear-end collision crash and hence, design useful driver assistance and vehicle control systems. In [21], Wang *et al.* proposed a deep neural network-based car-following model that takes the relative velocities and positions between the two observed vehicles in the last few time intervals as inputs.

In [22], Zhu *et al.* presented a framework for human-like autonomous car-following planning. This method is based on deep Reinforcement Learning (deep RL). The model maps in a human-like way from speed, relative speed between vehicles, and inter-vehicle spacing to acceleration. In [23], Abbas *et al.* aimed at modeling normal and safety-critical driving behavior in traffic under naturalistic driving data using agent based modeling techniques. In [24], Zhou *et al.* proposed an RL-based car-following model for the Connected and Automated Vehicles (CAV) in order to obtain an appropriate driving behavior to improve travel efficiency, fuel consumption and, safety at signalized intersections in real-time. The result shows that by specifying an effective reward function, the automated car can efficiently navigate under different traffic scenarios as well as traffic light cycles with different duration. This study reveals a potential of emerging RL technologies in transport research and applications. In [25], Gao *et al.* established the reward function of each driver data based on the Inverse RL algorithm and analyzed the driving characteristics and the following strategies. These methods have empowered AVs with extra intelligence in understanding the behavior of the leader vehicle by predicting its activity and making in-situ decision making but still with different degree of data exchange. In this study, we propose to making car-following decision making based on local inputs provided by the follower AV's sensors only. We base our study on the video-frames collected by the front camera of the AVs so as to make real-time decisions based on what is seen only. Ideally, other data streams provided by different data sources can be aggregated and combined together for better decision making. However, in this study, we aim to analyze the ability of a two-phase AI framework involving a computer vision module followed by a reinforcement learning module in performing accurate car-following decisions.

III. PROPOSED CAR-FOLLOWING FRAMEWORK

In this section, we present the adopted methodology to model the RGB-D image-based car-following problem for AVs.

In traffic flow theory, car-following is a method for determining how vehicles follow one another on a roadway. The idea with this time-continuous model is that a vehicle will maintain a minimum space and time gap between itself and the vehicle that precedes it. In this paper, we adopt the one-leader one-follower car-following problem. To do so, we need to detect a vehicle using an object-detector algorithm and to follow it in an autonomous way using an RL approach as shown in Fig. 1.

For the object detection phase, we use the pre-trained object-detector YOLOv3 since it effectively performs in real-time applications [14]. To feed this algorithm with images, we need to get, in a continuous way, video frames collected by the front camera of the follower. Then, YOLOv3 tries to detect cars and objects and returns the bounding box of each detected object. Next, we determine the distance and the angle between the leader and the follower.

On the other hand, the RL phase is responsible for the real-time navigation and decision making processes of the follower, in which the trained proposed algorithms use all the knowledge that they acquired during the training phase to autonomously follow the leader using only features extracted from the RGB-D frames. The first employed algorithm is the QL, which is characterized by its rapidity that is an important key in the autonomous driving field. The other algorithm is a modified version of the DQN model, that combine image sensory inputs with other environmental features to enhance the driving capabilities of the AV.

IV. OBJECT DETECTION AND FEATURES EXTRACTION

In this section, we start by presenting the main object detection learning techniques for autonomous application. Then,

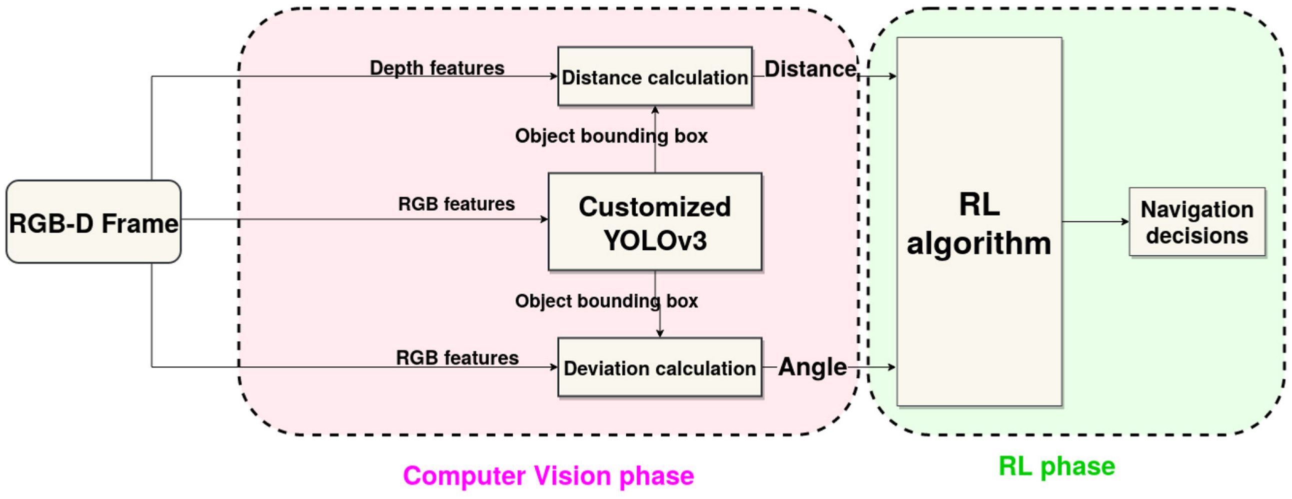


FIGURE 1. Overview of the proposed car-following framework.

we present the architecture of YOLOv3 object detector. Finally, we introduce the YOLOv3 custom object detection and its different steps.

A. OBJECT DETECTION LEARNING TECHNIQUES

In this section, we overview the main object detection learning techniques using image processing for AV applications. The purpose is to briefly investigate the performance of machine learning models such as the Support Vector Machine (SVM) model as well as deep learning algorithms like the You Only Look Once (YOLO) algorithm and the Single Shot Detector (SSD) in terms of accuracy and the timing-process [15]. SVM is a supervised learning model that analyzes data for both regression and classification. YOLO is a fully CNN (FCN). It contains only convolutional layers and is invariant to the size of the input image. YOLO takes a different approach to object detection since it “looks” at each image once as indicates its name. The SSD is one of the first methods using a single convolutional network’s pyramidal feature hierarchy for different size object detection. The pyramidal hierarchy consists of several convolutional layers of decreasing sizes. Simulation results, presented in Section VI-A, show that SVM poorly performs and its speed cannot assure real-time response. Therefore, machine learning solutions, in general, are not suitable for real-time object detection. Deep learning solutions such as YOLO and SSD provide effective vehicle detection results and can be implemented to complement other technologies in detection vehicles and obstacles. YOLO is very fast due to the fact that it looks at the image once. The SSD still suffers from a lower speed in treating frames. Depending on the application’s objectives, YOLO should be employed for extremely real-time processing due to its rapidity in treating the frame while SSD can be employed for its high accuracy in detecting small objects. To this end, we choose to pursue this study with the fastest algorithm, the YOLO model, as the main object detector for the car-following approach.

B. YOLOV3 OBJECT DETECTION TECHNIQUE

In this section, we introduce the multi-object detection algorithm YOLOv3 and the different steps to be followed to segment the environment into different states. With the 3rd version of YOLO [14], YOLOv3 has incremental improvements compared to YOLOv2 letting the algorithm capable of detecting small objects in a much accurate way than the Single Shot Multi-Box Detector (SSD) [27].

Composed of a total of 106 layers with 75 convolutional layers and 31 other layers such as shortcut, upsample, yolo, and route layers, YOLOv3 is a feature-learning based network and can deal with images with any sizes due to the use of no fully-connected layer, which makes the algorithm very suitable for real-time object detection without affecting its precision. YOLOv3 performs detection at layers 82, 94, and 106. During the training phase, YOLOv3 network is fed with input images to predict 3D tensors corresponding to the last feature map which refers to three different scales. These scales are implemented to detect small, medium, and big objects. Hence, it can detect distant and close vehicles. As the network goes deeper, its feature map gets smaller. As in SSD, object detection is carried out on different feature map in order to catch various scales. YOLOv3 is improved by adopting a Feature Pyramid Network (FPN)-like structure [28].

As shown in Fig. 2, the FPN is similar to the pyramidal feature hierarchy but more features are utilized by up-sampling the feature map and merging it with the current feature map. This leads to detect the features in further layers to improve the prediction accuracy and capture both low and high level object’s information.

For class prediction and multi-label classification, YOLOv3 uses binary cross-entropy loss for each label and also replaces the softmax function of YOLOv1, which converts scores into probabilities, with independent logistic classifiers to calculate the likeliness of the input belonging to a specific label.

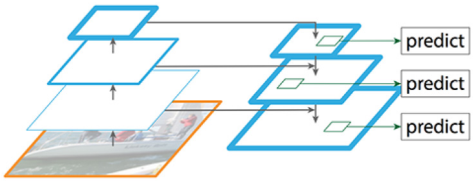


FIGURE 2. Feature Pyramid Network (FPN) structure.

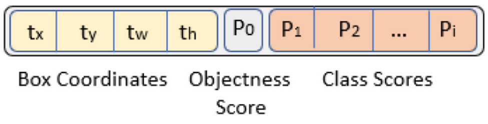


FIGURE 3. Illustration of the attributes of the YOLOv3 bounding box.

For anchor boxes and cost function calculation, YOLOv3 uses in total nine anchor boxes with three for each scale and can predict bounding boxes per image ten times more than YOLOv2. As shown in Fig. 3, YOLOv3 uses refined parameters t_x and t_y related to b_x and b_y: the (x, y) bounding box center coordinates, which are given as follows:

$$b_x = \sigma(t_x) + c_x \text{ and } b_y = \sigma(t_y) + c_y, \tag{1}$$
$$b_w = p_w e^{t_w} \text{ and } b_h = p_h e^{t_w}, \tag{2}$$

where b_w and b_h are the width and the height of the prediction respectively, t_x, t_y, t_w, and t_h are the detector outputs, and σ is the sigmoid function. The parameters c_x and c_y are the top-left coordinates of the grid, p_w and p_h are the anchor box dimensions, p_0 is the confidence of the detected object while p_c denotes the i^{th} class probability.

In Fig. 13, we show the output of YOLOv3 applied to the input image where two objects are detected as shown by the two bounding boxes. For each bounding box, YOLOv3 shows the class of the object detected and the score which represents the level of confidence of the object detected.

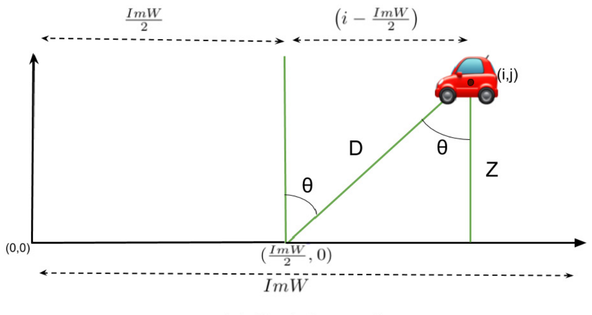
C. YOLOV3 CUSTOM OBJECT DETECTION

In this section, we present the YOLOv3 custom object detection technique to detect obstacles and noises in the road environment, in other words, identify vehicles other than the leader. Custom object detection, is a transfer learning technique where a model developed for a task is reused as the starting point for a model in a second task. It is the situation where what has been learned in one setting, is exploited to improve generalization in another setting.

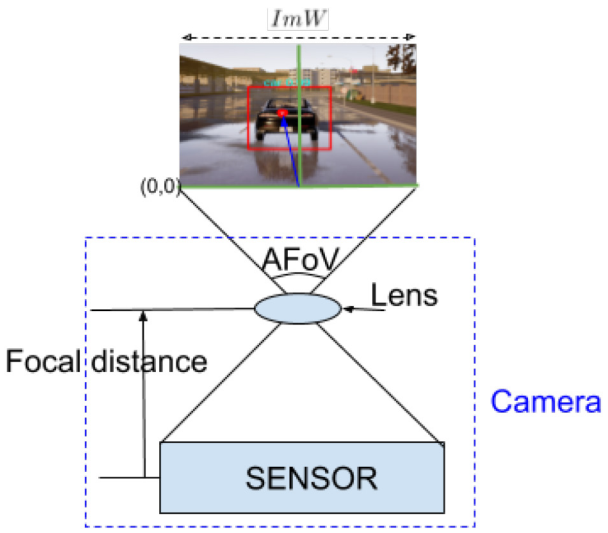
Dataset Collection and Annotation: As with any deep learning task, the first and the most important step is to collect and prepare the dataset. To this end, we collect a set of 2000 images with a size of 608 × 608 with different poses and scales to provide a reasonable accuracy.

The next step is to annotate the dataset using an open-source software written in python to define the location (Bounding box) of the object in each collected image.

Training Process: As with any deep learning training procedure, we need to split the data into training and test sets.



(a) Deviation angle



(b) AFoV

FIGURE 4. Illustration of (a) the deviation angle and (b) the AFoV.

We select randomly, 1800 images of the data for training and 200 images for testing the model.

To train the model, we choose Darknet, an open source deep neural network framework [29]. Darknet comes with multiple configuration for training on different architectures by just using pre-trained models which contain convolutional weights trained on ImageNet.

D. DISTANCE AND DEVIATION ANGLE CALCULATION

The distance and deviation angle are calculated using the RGB-D frames provided by simulated depth sensory inputs (e.g., the Orbbec 3D camera) attached to the follower vehicle. Using the bounding box center point provided by YOLOv3 as a reference of the object location on the image, it is possible to accurately estimate the distance and the deviation angle. This method is much better than using the bounding box coordinates or the 3D object detection [30] due to its complexity or limited accuracy. To this end, we need to express the angular size of each pixel horizontally, denoted by Γ_x, as follows:

$$\Gamma_x = \frac{AFoV}{ImW}, \tag{3}$$

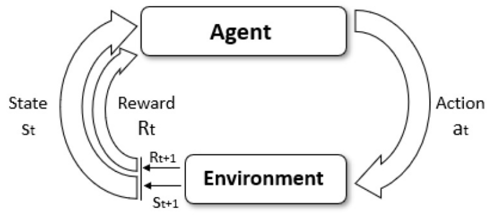


FIGURE 5. Reinforcement learning illustration.

where ImW is the image width and $AFoV$ is the horizontal Angular Field of View (AFoV). The latter is a constant specific for every camera. In practice, AFoV is a measure of the vantage point from the lens distance as shown in Fig. 4. For the Orbbec 3D sensor, $AFoV = 60°$. The camera is positioned at $(\frac{ImW}{2}, 0, 0)$ in the Cartesian coordinate system and the central pixel of the bounding box is positioned at (i, j) without taking into consideration the third dimension. As a start, we proceed by calculating the angle of deviation, denoted by θ, using the following expression:

$$\theta = \Gamma_x \times \left(i - \frac{ImW}{2}\right), \tag{4}$$

where $(i - \frac{ImW}{2})$ denotes the number of pixels from center to the representative pixel.

Consecutively, we compute the distance D separating the follower and the leader as $D = \frac{Z}{cos(\theta)}$, where Z is the distance between the sensor plane and the object, which is directly extracted from the depth features of the sensor. Each depth sensor has a depth scale value that translates pixels to meters. Accordingly, we define the distances $D_{\min}$ and $D_{\max}$, respectively, as the maximum distance to ensure the detection of the leader-vehicle and a danger distance that refers to the minimum distance separating the leader and the follower at which the probability of crashing is significant.

V. REINFORCEMENT LEARNING FOR AUTONOMOUS CAR-FOLLOWING

In this section, we present two different RL algorithms to enable autonomous car-following. For each algorithm, we describe the mathematical methodology behind it and we define the space of states and actions that identify the possible navigation decisions during the navigation decision-making process. But first, we briefly overview RL and developed the reward function employed for the RL car-following phase.

A. REINFORCEMENT LEARNING

Considered as the best paradigms in the realm of Artificial Intelligence (AI), the RL can be applied to teach machines how to accomplish the optimal behavior within a given context and how to map circumstances to behaviors. RL is a type of AI technique that allows an agent to learn from their own actions and experiences, either positive or negative.

An RL setup usually consists of two parts, the agent and the environment as shown in Fig. 5 here an agent needs to autonomously learn the actions to take given the state of its environment in order to maximize a certain reward.

In the context of the car-following problem, the environment is what the camera can observe, while the agent is the following vehicle which uses the RL algorithm to navigate and follow its leader. To decide what action a_t should be taken in response to a current state s_t i, the agent starts to capture images through its front camera and processes them to extract the features needed by the algorithm, as described in Section IV. After executing the action a_t, a reward R_t is then computed and the agent will move to a next state s_{t+1}. This process will be repeated during the training phase of the RL algorithm. At training iteration, the model will update its parameters based on its own behavior. The model convergence is reached if the accuracy of the model does not improve for a fixed number of consecutive iterations.

B. REWARD FUNCTION: COLLISION WITH THE LEADER VS. DETECTION LOSS

The reward function, in the context of the investigated car-following problem, should consider the distance D that separates the leader and the follower, the deviation angle θ, and the collision risk with the leader-vehicle. Consequently, the AV will be forced to follow the leader. Therefore, we suggest a piecewise function that considers the maximum distance, $D_{\max}$, to successfully detect the leader vehicle and a danger distance, $D_{\min}$, that refers to the minimum distance separating the leader and the follower at which the probability of crashing is significant. The reward function of the RL algorithm for the proposed car-following approach, denoted by $R_t(D, \theta)$, can be expressed as follows:

$$R_t(D, \theta) = \begin{cases} \zeta - \alpha\left((\beta \times D - \delta)^2 + \theta^2\right) & \text{if } D_{\min} \leq D \leq D_{\max}, \\ -\infty & D < D_{\min}, \end{cases} \tag{5}$$

where α is the coefficient of proportionality between the reward and the variables (i.e., Distance D and angle θ), $\frac{\delta}{\beta}$ is the ideal distance between the leader and the follower, and ζ is the maximum reward. To decrease the probability of crashing with the leader, the model considers any case with distance D less than $D_{\min}$ as a potential crashing situation. Distances higher than $D_{\max}$ may lead to non-detection cases.

In Fig. 6, we plot the reward function, expressed in (5), versus the most parameters affecting the navigation of the follower, namely the distance D and the angle θ. We notice that the reward function is maximized if θ is around 0o and the distance is equal to $\frac{D_{\max}}{2}$ which means that the autonomous agent is attempting to maintain an angle to stay right behind the leader within a safe distance to keep its smooth detection and avoid any risk of crash.

C. AUTONOMOUS CAR-FOLLOWING DRIVING WITH FINITE STATE SPACE: QL ALGORITHM

In this section, we use a rapid and simple-structured RL method: the QL, where $\boldsymbol{Q}$ stands for quality which reflects how efficient is a given action in maximizing coverage on

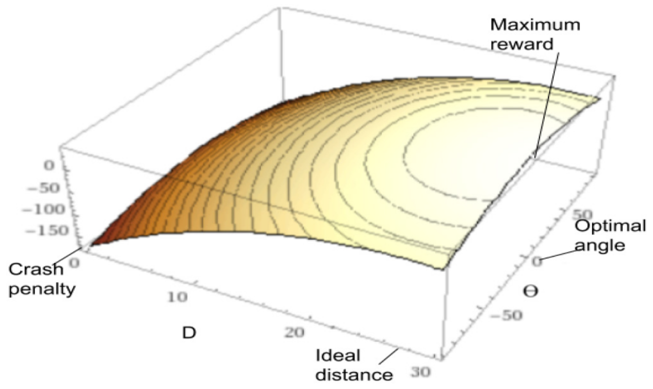


FIGURE 6. Plot of an example of the reward function for $D_{\min} < D < D_{\max}$ where $\alpha = 0.005$, $\beta = 7$, $\delta = 190$, and $\zeta = 40$.

highest priority events [31]. QL is called a tabular method where the spaces for the state and the behavior in such problems are small enough and their approximate value functions can not be represented as arrays and tables. In addition, QL is a model-free RL method which allow the achievement of car-following solutions with reduced complexity, i.e., less run-time compared to other RL techniques. Before starting the learning phase, the $\boldsymbol{Q}$-table is initialized with zero-elements. Then, the agent chooses an action a_t, gets a reward R_t, moves to a new state s_{t+1}, and then updates its $\boldsymbol{Q}$-table according to the Bellman function [32]:

$Q^{\pi}: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ represents the expected return of a state-action pair given by the policy π and updated as follows:

$$Q(s_t, a_t) = (1-\alpha)\, Q(s_t, a_t) + \alpha \left(R_t + \gamma \max_a Q(s_{t+1}, a) \right), \tag{6}$$

where $Q(s_t, a_t)$ is the estimate for the action-value function of the pair state-action, $\alpha \in [0, 1]$ is the learning rate that reflects how the new information will override the old one and $(R_t + \gamma \max_a Q(s_{t+1}, a)$ is the learned value, which is composed of the immediate reward R_t and the estimate of the future $\gamma \max_a Q(s_{t+1}, a)$ where $\gamma \in [0, 1]$ is the discount factor which reflects the effect of the previous actions for the evaluation of the current state. The sets $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces of the car-following problem. In this section, we denote by D_e the distance D previously defined in Section IV-D. During the training phase of an QL model, we put the leader in different states and let the follower takes random actions to follow it until it learns how to navigate. In case of non-detection or loosing the simulation will be interrupted and a new episode is started. For the testing phase of the proposed approach, the follower tries to follow the leader by choosing the action to ensure the highest reward from the $\boldsymbol{Q}$-table corresponding to the state of the leader.

To summarize the solution, the pseudo code in Algorithm 1 describes the training method of the QL algorithm to enable autonomous car-following. Since the leader vehicle can be located at any position with respect to the follower, we represent the set of states, denoted by $\mathcal{S}$, as follows: $\mathcal{S} = \{s_0, s_1, \ldots, s_I\}$ where I is the number of regions obtained from the first phase and s_0 is the state where the follower fails in detecting the leader.

Algorithm 1 QL Algorithm for Car-Following Training

```
1: **Require:**
2: States $\mathcal{S} = \{s_0, \cdots, s_I\}$,
3: Actions $\mathcal{A} = \{a_0, \cdots, a_T\}$,
4: Reward function $\mathcal{R} = S \times A \rightarrow \{\mathbb{R}\}$,
5: Learning rate $\alpha \in [0, 1]$,
6: Discount factor $\gamma \in [0, 1]$,
7: procedure ObjectDetection()
8: Image ← get a frame,
9: Bounding boxes ← YOLOv3 (Image),
10: for each bounding box do
11:     Calculate the distance $D_{box}$,
12:     Estimate the distance $D_e$,
13:     Calculate the angle $\theta$,
14:     Identify the state $\hat{s} \in \mathcal{S}$ given $D_e$ and $\theta$,
15: end for
16: return $\hat{s}, D_e$
17: procedure Q-learning()
18: Set $\boldsymbol{Q}$ to zero,
19: while Training do
20:     $(\hat{s}, D_e) \leftarrow$ ObjectDetection(),
21:     Start in state $\hat{s}$,
22:     while Leader is detected $(\hat{s} \neq s_0)$ do
23:         Select the strategy $\pi$ according to $\varepsilon$ (exploration or
            exploitation),
24:         $a \leftarrow \pi(s)$ using (9),
25:         $(\hat{s'}, D'_e) \leftarrow$ ObjectDetection(),
26:         $R \leftarrow \mathcal{R}(\hat{s'}, D'_e)$,
27:         $Q(\hat{s}, a) \leftarrow (1-\alpha)Q(\hat{s}, a) + \alpha(R + \gamma max_a Q(\hat{s'}, a))$,
28:         Update $\varepsilon$ using (10),
29:         $\hat{s} \leftarrow \hat{s'}$
30:     end while
31: end while
32: return $\boldsymbol{Q}$
```

1) OBSTACLE-FREE CAR-FOLLOWING SCENARIO

In the case of an obstacle-free car-following, the AV is driving in a situation where there is no detected objects other than the leader, and even if there is another vehicle, it will be ignored by the computer vision phase. In fact, this scenario is useful in some cases such as highways and no overtaking roads. In this section, we define the states and actions space for the obstacle-free car-following scenario as follows:

- The environment is what the front camera of the follower vehicle can see in the simulation environment.
- The agent is an AV trying to follow one leader vehicle: For this scenario, we consider the existence of one leader and one follower vehicles in a straight road. Obstacles will be investigated in Section V-C2.
- The actions are all the possible moves. Due to the use of low number of actions and states, we choose a low speed leader-follower (no more than 25 km/h). As described in Table 1, we choose a set of $T = 7$ actions where

TABLE 1. The set of actions $\mathcal{A}$.

Actions	In practice	In virtual environment
a_0	Steering left +	Press left button +
a_1	Steering right +	Press right button +
a_2	Going straight	Press up button +
a_3	Steering left ++	Press left button ++
a_4	Steering right ++	Press right button ++
a_5	Speed up	Press up button ++
a_6	Stop the vehicle	Press down button

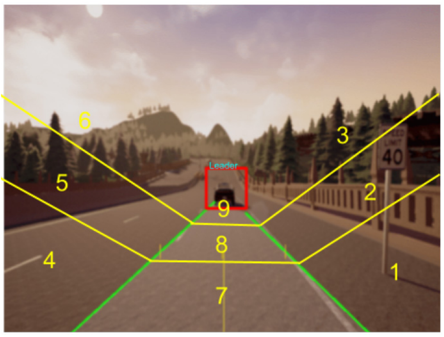


FIGURE 7. Obtained states after image segmentation for the obstacle-free scenario.

TABLE 2. The set of states $\mathcal{S}$.

States	Description
s_0	ND: No object detected
s_1	$\theta \leq 35°$ and $\mathrm{Y}_1(x_p, y_p) \leq 0$
s_2	$\theta \in]35°, 55°]$ and $\mathrm{Y}_1(x_p, y_p) \leq 0$
s_3	$\theta > 55°$ and $\mathrm{Y}_1(x_p, y_p) \leq 0$
s_4	$\theta \leq 35°$ and $\mathrm{Y}_2(x_p, y_p) \leq 0$
s_5	$\theta \in]35°, 55°]$ and $\mathrm{Y}_2(x_p, y_p) \leq 0$
s_6	$\theta > 55°$ and $\mathrm{Y}_2(x_p, y_p) \leq 0$
s_7	$\mathrm{Y}_1(x_p, y_p) > 0$, $\mathrm{Y}_2(x_p, y_p) > 0$ and $D_e \leq D_{\text{min}}$
s_8	$\mathrm{Y}_1(x_p, y_p) > 0$, $\mathrm{Y}_2(x_p, y_p) > 0$ and $D_e \in]D_{\text{min}}, D_{\text{max}}]$
s_9	$\mathrm{Y}_1(x_p, y_p) > 0$, $\mathrm{Y}_2(x_p, y_p) > 0$ and $D_e > D_{\text{max}}$

$\mathcal{A} = \{a_0, a_1, a_2, a_3, a_4, a_5, a_6\}$. In this simulation, two acceleration rates have been used that we refer to as "+" and "++", respectively.

- The states are the regions of the captured image defined according to the distance D_e and the angle θ indicating the relative position of the leader. As shown in Fig. 7, we choose to divide the image, having as origin its top left corner, into a set of $I = 10$ states $\mathcal{S} = \{s_0,\ s_1,\ s_2,\ s_3,\ s_4,\ s_5,\ s_6,\ s_7,\ s_8,\ s_9\}$. The right-hand and left-hand side lines in Fig. 7 have the equations $\mathrm{Y}_1(x, y)$ and $\mathrm{Y}_2(x, y)$, respectively. In Table 2, we can determine the states of the follower by calculating $\mathrm{Y}_1(x_p, y_p)$ and $\mathrm{Y}_2(x_p, y_p)$ where x_p and y_p are the (x, y) coordinates of the center of the bounding box of the detected vehicles. Table 2 provides all the possible states given D_e and θ. Note that the states are independent of the road lanes; they are fixed for the input images and are only subject to the follower vehicle behavior.
- The learning rate α is set to 0.5 and the discount factor to 0.4. For the exploration-exploitation trade-off described in Section V-E, we set ε_{max} to 1 and ε_{min} to 0.01.

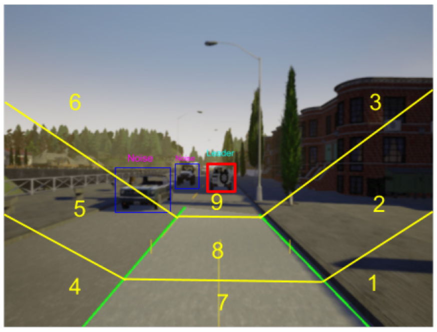


FIGURE 8. States segmentation for the obstacle-Aware scenario.

TABLE 3. Example of possible states for the obstacle-aware scenario.

States	Leader Description	Obstacle Description
s_{00}	leader in state ND	obstacle in state ND
s_{10}	leader in state 1	obstacle in state ND
s_{05}	leader in state ND	obstacle in state 5
s_{68}	leader in state 6	obstacle in state 8
s_{44}	leader in state 4	obstacle in state 4
s_{17}	leader in state 1	obstacle in state 7
s_{99}	leader in state 9	obstacle in state 9

2) OBSTACLE-AWARE CAR-FOLLOWING SCENARIO

In this section, we consider the existence of an obstacle in the environment in which the AV is navigating. This scenario is more complex for the AV. Indeed, the follow must take into consideration other objects in the environment to avoid any erroneous navigation decision that could lead to a crash. In this section, we assign and set the different parameters and components of the QL model as follows:

- The agent is an AV trying to reach one leader car: The investigated scenario consists in one leader, one obstacle, and one follower vehicles where the obstacle is another vehicle that should be avoided and not followed by the follower.
- The actions are kept the same as presented earlier in Section V-C1.
- The states are defined in this case using the same approach as previously. However, the segmentation of the captured image is established according to the distance D_e and the angle α, previously defined, in addition to the distance D_{Oe} and the angle α_O indicating the relative position of the obstacle compared to the follower. Using the same segmentation methodology as in Fig. 8, every image frame is segmented, having as origin its top left corner, into a set of $I = 100$ states $\mathcal{S} = \{s_{00},\ s_{01},\ s_{02},\ s_{03},\ s_{04},\ s_{05},\ s_{06},\ s_{07},\ s_{08},\ \ldots,\ s_{ij},\ \ldots,\ s_{99}\}$ where $i \in [0, 9]$ refers to the state of the leader and $j \in [0, 9]$ refers to state of the obstacle. To obtain s_{ij}, we need to first determine both the leader and the obstacle states and then, combine them into one state. Table 3 provides some of the possible states used in the obstacle-aware car-following scenario.

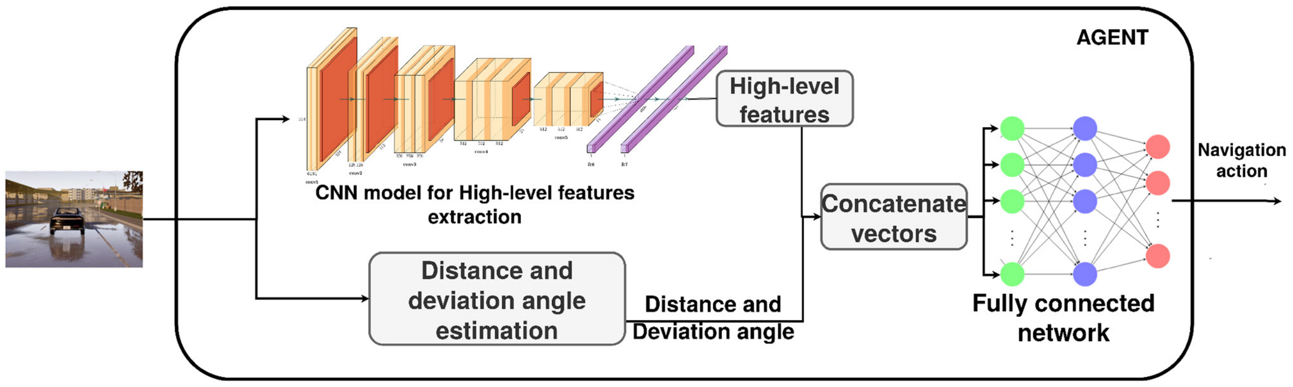

AGENT
CNN model for High-level features extraction
High-level features
Concatenate vectors
Distance and deviation angle estimation
Distance and Deviation angle
Fully connected network
Navigation action

FIGURE 9. Architecture of the DQN agent.

Algorithm 2 DQN Algorithm for the Car-Following Training

```
1: Initialize replay memory $\mathcal{B}$,
2: Initialize number of episodes $M$,
3: Initialize action-value function $Q$ with two random sets of weights $\Theta$ and $\Theta'$,
4: $\varepsilon = 1$,
5: for $i = 1, \cdots, M$ do
6:     for $t = 1, \cdots, T$ do
7:         Choose $N \in [0..1]$,
8:         Set $a_t = \begin{cases} \arg\max_a Q(s_t, a; \Theta), & \text{if } N > \epsilon \\ \text{random action}, & \text{otherwise}, \end{cases}$
9:         Execute action $a_t$, collect reward $R_{t+1}$, and observe next state $s_{t+1} = (Frame_{t+1},\ D_{t+1},\ \theta_{t+1})$,
10:        Store the transition $(s_t, a_t, R_{t+1}, s_{t+1})$ in $\mathcal{B}$,
11:        Sample mini-batch of transitions $(s_j, a_j, R_{j+1}, s_{j+1})$ from $\mathcal{B}$,
12:        Set $y_j = \begin{cases} R_{j+1}, & \text{if } s_{j+1} \text{ is terminal} \\ R_{j+1} + \gamma \max_{a'} Q(s_{j+1}, a'; \Theta'), & \text{otherwise}, \end{cases}$
13:        Perform a gradient descent step using targets $y_j$ with respect to the online parameters $\Theta$,
14:        Every $C$ steps, set $\Theta' \leftarrow \Theta$,
15:    end for
16:    $\epsilon = \epsilon + (\epsilon_{max} - \epsilon_{min}) \times \exp(-k \times i)$,
17: end for
```

D. AUTONOMOUS CAR-FOLLOWING DRIVING WITH INFINITE STATE SPACE:DEEP Q-NETWORK ALGORITHM

In this section, we present the RL approach to address the car-following problem. The architecture of the RL model is given in Fig. 9.

Indeed, the RL Algorithm, DQN [33], exploits the CNN to extract the relevant information in the image, with the goal of approximating the Q-value function, defined later in (8), that will be used for the decision making process. In our proposed model, we suggest to change the architecture of the CNN model inside the DQN algorithm by adding additional features that help the model avoid crashing with the leader and ensure a high-level safety driving. The RGB-D frame alongside the extra features are given as the input and the maximum Q-values of all possible actions are generated as the output. At each step t, in Algorithm 2, the agent inspects the current state S_t of the environment, decides which action to be taken from the predefined action-space according to a policy π and finally, observes a reward signal r_t to evaluate this action. The goal of the agent is to identify the best policy that maximizes the expected sum of discounted rewards r_t expressed as follows:

$$r_t \doteq R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots + \gamma^T R_{t+T} \doteq \sum_{k=0}^{T} \gamma^k R_{t+k}, \tag{7}$$

where $\gamma \in [0, 1]$ is a discount factor that determines the importance of future rewards and T is the number of steps. The Q-function of a given policy π is defined as the expected return from executing an action a in a state S as follows:

$$Q^{\pi}(s, a) = \mathbb{E}[R_t \mid s_t = s, a_t = a, \pi]. \tag{8}$$

In the algorithm, we stock last agent experiences (s_t, a_t, R_t, s_{t+1}) into a buffer B. Then, we take samples already stocked to train the deep network. The idea behind taking samples from the replay buffer is that the data becomes almost independent and identically distributed (i.i.d.). Consequently, we obtain a more generalized model. In particular, DQN uses deep learning into the aim of obtaining an estimate of the Q-function for the current policy which is close to the optimal Q-function Q^* defined as the highest return we can expect to achieve by following any strategy. Hence, to successfully reach the Q^*, we should find the optimal model weights, denoted by Θ. To reach a more stable input and output, we create two deep networks, the target network (weighted with Θ') and the standard network (weighted with Θ). We use the first one to retrieve Q values while the second one includes all updates in the training. After C steps, we synchronize Θ' with Θ that fixes Q-value targets temporarily, in consideration of the fact that the data is not 100% independent and identically distributed.

- State space: In our model, every state is defined by the distance D and the deviation angle θ, between the leader and the follower vehicle, in addition to the high-level features extracted by the Convolutional Neural Network

TABLE 4. Actions space of the DQN model.

Action	Description
a_0	Accelerate
a_1	Steer left at same speed
a_2	Steer right at same speed
a_3	Accelerate and steer left
a_4	Accelerate and steer right
a_5	Decelerate
a_6	Steer left and Decelerate
a_7	Steer right and Decelerate

(CNN) from every frame. Hence, the autonomous follower extracts the previous features from the frames and concatenates them with the output of the CNN that contains all semantic information about the environment to identify the state.

- Action space: Our model is trained to choose an action from an action-space composed of eight possible behaviors as shown in Table 4. The possible actions include deceleration, acceleration, and steering left and right. To sum up, the training process of the DQN model is presented in Algorithm 2. The latter algorithm contains an initialization part, in which the training environment is prepared and a second repetitive part where the weights will be updated in each epoch.

E. EXPLORATION AND EXPLOITATION INTERPLAY

In the training phase of both algorithms (QL or DQN), we consider to balance between two phases: the exploration and exploitation phases. We aim to achieve a balance between exploring the environment and exploiting the already acquired knowledge while training the RL algorithms to maximize their efficiency. The logic behind this technique is that since the agent needs to get the highest reward, it can choose between keep trying new actions to bring higher reward (exploration phase) or just selecting the action that leads to the highest reward based on current information (exploitation phase). Depending on the value of a parameter $\varepsilon \in [0, 1]$, the decision to explore or exploit is made. It is thus important to find a balance between these two extremes. The idea is that we must have a large value of ε at the beginning of the training and then, we gradually decrease it since the follower becomes more confident at making decisions (i.e., estimating $\boldsymbol{Q}$-values). We employ the ε-greedy strategy as it is efficient and easy to implement. In this context, the action a_t is taken as follows:

$$a_t = \begin{cases} \arg\max_a Q(s_t, a) & \text{when } N \geq \varepsilon \\ \textit{random action } a & \textit{otherwise}, \end{cases} \tag{9}$$

where N is a random number generated between 0 and 1.

To update the ε and decrease it at each step, we apply the following rule:

$$\varepsilon_{t+1} = \varepsilon_t + (\varepsilon_{max} - \varepsilon_{min})e^{(-k\times i)}, \tag{10}$$

where ε_{min} is the minimum exploration probability, ε_{max} is the first exploration probability, k is the rate at which our exploration factor decays, and i refers to the i^{th} step of the training phase. In regards to the previous equation, we can notice that, initially, ϵ will be closer to its maximum, and hence the model prefer the exploration of the environment. Afterward, ϵ is going to decrease until we attend the its minimum. In other words, at the last episodes the model will train based on his previous experiences more than doing exploration decisions. As a result, we build an agent that learns the dynamics of the stochastic environment.

The proposed framework is based on two models. Each model has a set of possible actions, denoted by $\mathcal{A} = \{a_0, a_1, \ldots, a_J\}$. The cardinality of $\mathcal{A}$ is determined as $J = (3 \times (AC + DC) + 1)$ where 3 refers to the number of possible directions (left, straight, right), AC and DC are the number of possible acceleration and deceleration rates, respectively, and 1 refers to the action of braking till completely stopping.

VI. SIMULATION RESULTS

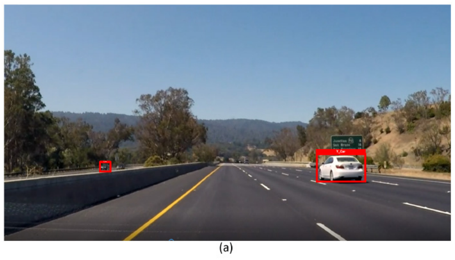
In this section, we investigate the performance of the object detection learning models using the proposed algorithms. Indeed, to train our model and test it, we simulate an environment who is on the verge of reality, using the CARLA simulator. Actually, CARLA is very practical in this case due to the high cost of testing such a solution in a realistic environment. The latter simulator is open-source and it has been designed to support the development, training, and validation of autonomous driving systems. In addition to the open-source code and protocols, CARLA provides open digital assets (urban layouts, buildings, vehicles) that were created for this purpose and can be used freely.

A. PERFORMANCE COMPARISON OF THE OBJECT DETECTION TECHNIQUES

In this simulation, a video of 50 seconds obtained from a real-world setting and taken from the front camera of a self-driving car was chosen to compare the SVM, YOLO, and SSD models. We choose a sub-division of 1250 frames for this video which means that an image is captured every 40 ms. Then, the set of frames is fed to the learning models to detect different vehicles.

In Fig. 10, we provide two instantaneous snapshots of the recorded video showing the detection of the SSD and YOLO algorithms. The YOLO model, illustrated in Fig. 10(a), detects the nearby vehicles but struggles with smaller objects unlike the SSD, which can reach higher accuracy due to the presence of default boxes and multi-scale features as shown in Fig. 10(b). The SSD has more portability to detect vehicles even moving in the opposite direction.

In Fig. 11, we present an investigation of the number of frames treated by each model: The Frame Per Second (FPS) was computed to reflect the speed of each learning model in treating the video-frames. We can notice that SVM is very slow and its speed cannot exceed 2 FPS because of its incorporated sliding window approach. Thus, this model cannot be run for real-time applications. The YOLO model


(a)

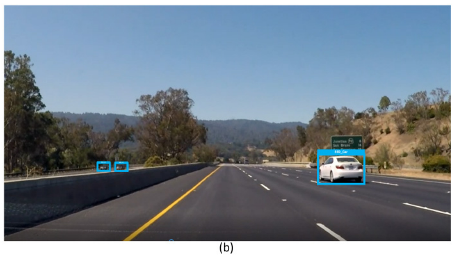

(b)

FIGURE 10. Snapshots taken at the same instant showing the detected vehicles using (a) YOLO, and (b) SSD.

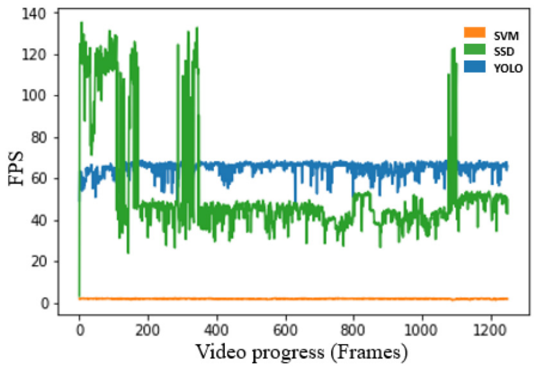


FIGURE 11. FPS versus the video progress for SVM, YOLO, and SSD.

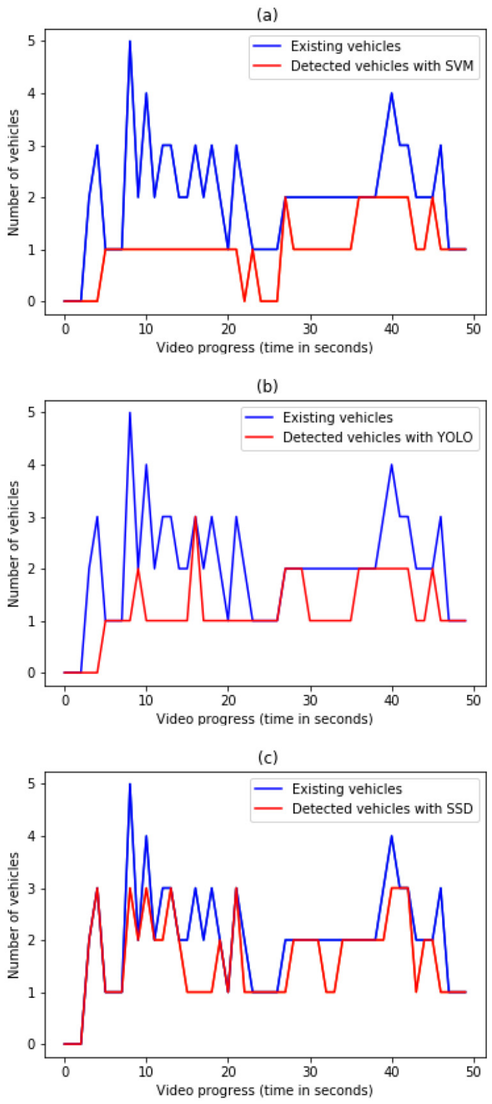


FIGURE 12. Number of detected and actual existing vehicles versus time with (a) SVM, (b) YOLO, and (c) SSD.

has shown a very fast processing time and can achieve 50-70 FPS due to the fact that it looks at the image once. Otherwise, the SSD model can attain on average 40-60 FPS and can be run for real-time object detection where rapid driving decisions should be made. Due to the absence of objects in some frames, SSD presents spikes where it can generate the output result rapidly.

In Fig. 12, a performance evaluation of the learning models in performing true detection was made on the input video, which means a comparison between the number of detected and existing vehicles during the duration of the input video. The existing vehicles include all the vehicles moving in both directions. The SVM, shown in Fig. 12(a) was not able to detect all the presented vehicles mainly those going in the opposite direction. Although, the YOLO model, shown in Fig. 12(b) detects more vehicles than the SVM model, but still poorly performs where objects are distant from the front camera, unlike the SSD, where it can detect vehicles going in both directions and small objects have higher chance to be detected as illustrated in Fig. 12(c). The Yolo and SSD have detected at least one vehicle during the test. Sill, it may

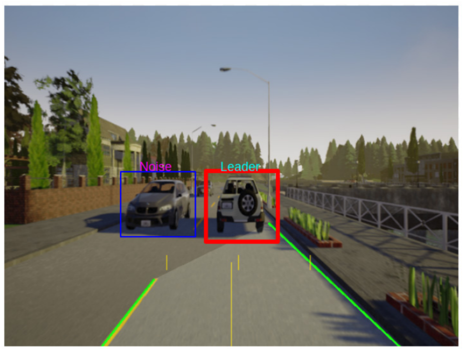


FIGURE 13. YOLOv3 custom object detection prediction.

happen that none of the model is able to detect a vehicle. This may happen when vehicles existing at the far lane of the opposite direction with respect to the vehicle of interest are manually labeled but not detected by the model. These vehicles are barely appearing in the video frame. If they

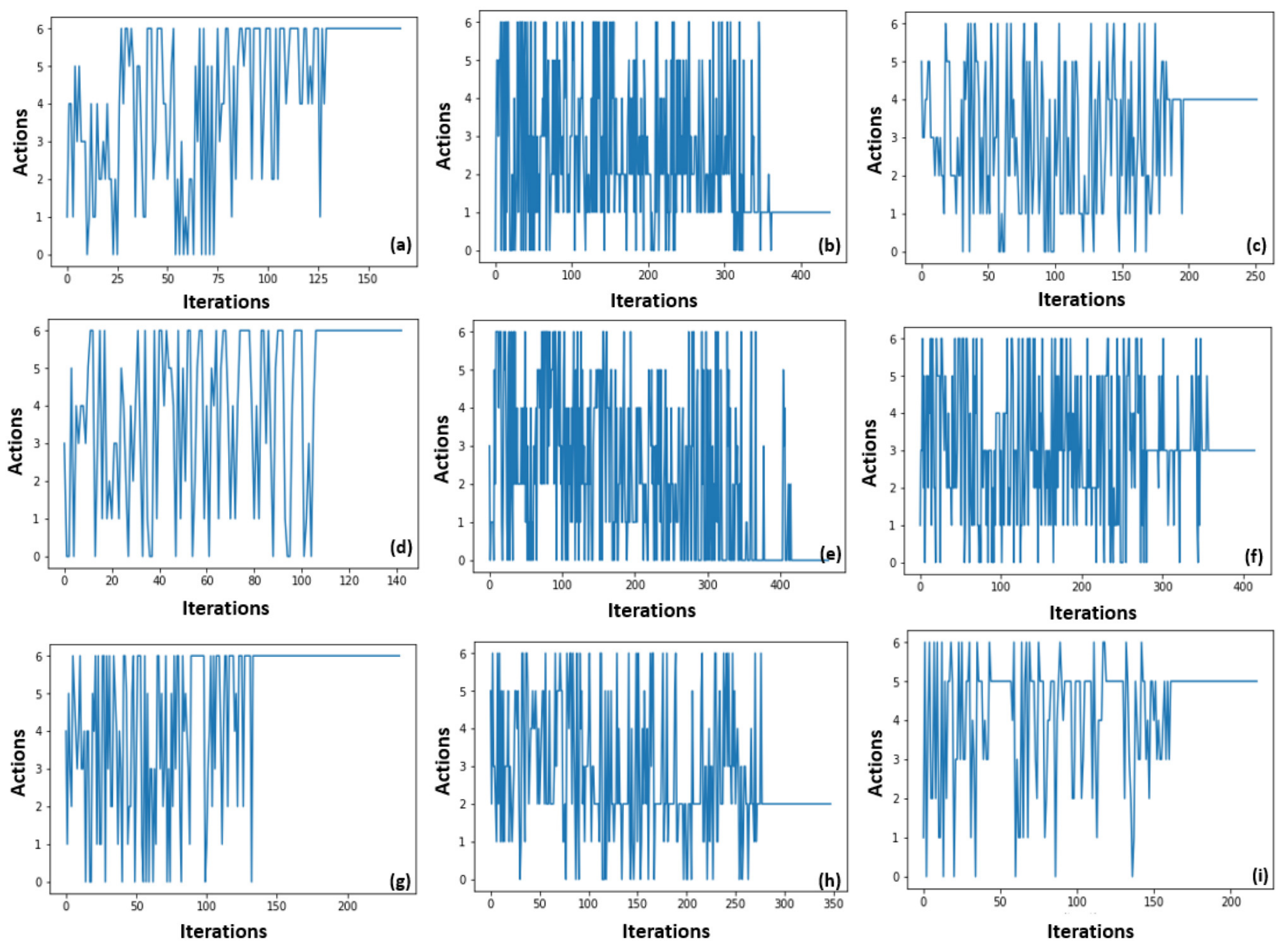


FIGURE 14. The taken actions during the QL training phase of the obstacle-free car-following scenario for each state: s_1 (a), s_2 (b), s_3 (c), s_4 (d), s_5 (e), s_6 (f), s_7 (g), s_8 (h), and s_9 (i).

appear, they are very small, which makes it hard for the object detector to identify them as vehicles.

B. YOLOV3 CUSTOM OBJECT DETECTION TECHNIQUE CONFIGURATION

The YOLOv3 model needs a configuration of basic training parameters:

- Batch hyper-parameters: The batch parameter indicates the batch size used during the training and test phases. We set the batch size to 16. It means 16 images per iteration and a subdivision of eight, in other words, we split the batch into eight fractions.
- Width, height, channels: These parameters indicate the size of the input images and the number of channels used. We select 608×608 RGB images.
- Momentum and decay: The momentum is used to penalize large weight changes between iterations and the decay to penalize large values for weights to avoid the overfitting. We set the momentum to 0.9 and the decay to 0.0005.
- Data augmentation: To maximize the use of the data, we can rotate the image by $\pm$ angle or changing its colors by changing the saturation. We set the angle to 0 and the saturation to 1.5.
- Number of iterations: For multi-class detection, it is recommended to run the training process for at least 2000 iterations per class. We set the iteration number sufficiently high (50000) to guarantee convergence.

Fig. 13 shows an example of the output of the YOLOv3 custom object detector applied to our dataset where it clearly distinguishes between the leader and the obstacle.

With the YOLOv3 custom object detection technique, we successfully distinguish between the leader and other units for any obstacle-aware car-following scenario. This technique can be extended to detect multiple units for several purposes.

C. SIMULATION RESULTS FOR THE Q-LEARNING ALGORITHM

1) OBSTACLE-FREE CAR-FOLLOWING SCENARIO SIMULATION

Fig. 14 presents the evolution of the actions taken during the training process when the follower is at a given state s_i. To train the QL model, we manually control the leader to

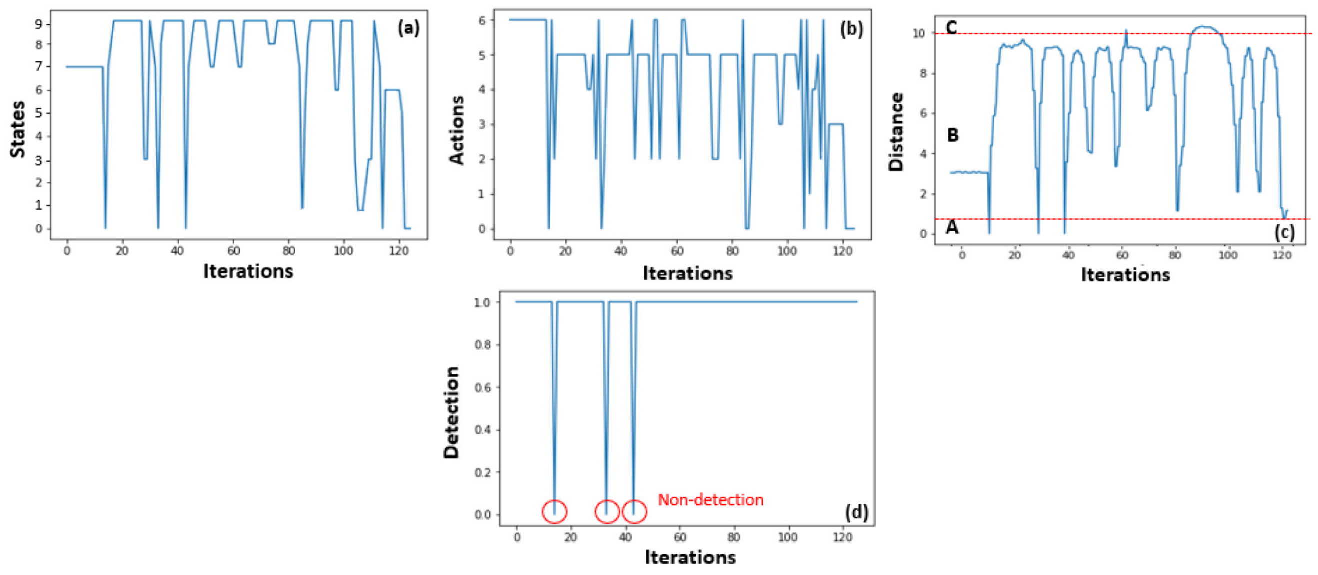


FIGURE 15. Performance of the follower during the QL testing phase for the obstacle-free car-following scenario: (a) states of the follower with respect to the leader, (b) chosen actions by the follower, (c) relative distance between the leader and follower, and (d) non-detection occurrence.

locate it in different states so as to the follower learns how to follow it by exploring the different states and by choosing random actions. For each taken action, the follower receives a reward based on the new state of the leader vehicle and the Q-table is updated. We repeat this process until the follower fails to detect the leader and declares the game over. As the game is over, the value of ε is decreased and the chance to exploit becomes higher. To this end, the agent becomes more confident at estimating Q-values and starts to choose actions that lead to the highest reward. By continuing this process, for each state, the algorithm converges to the actions which have the highest cumulative reward. Table 5 shows the final actions associated to all the states chosen by the follower. Fig. 14 illustrates the behavior of the follower over the training phase. The follower AV takes different actions at the beginning of the process for the same state till it reaches convergence. The number of iterations at the x-axis indicates the number of times an action is chosen while the AV is at a state s_i and what is the action that is chosen at that moment. For example, in Fig. 14(a), the first action taken when the AV is at state s_1 is action a_3. After several tests for each action, the QL converges towards a single action for each state as indicated in Table 5. Notice that the number of iterations differ from a state to another. This is due to the random trajectories of the Leader chosen and also, due to the random action taken by the AV during the training, which results in different scenarios. As example, the state s_7 occurred more than 400 times during the training while the state s_4 occurred around 140 times.

Fig. 15 shows the performance of the proposed framework during the testing phase. During this phase, we set a fixed trajectory for the leader vehicle in a straight road and we investigate the behavior of the AV in different states. Depending on the value of the distance D_e between the two

TABLE 5. States and actions taken using the QL model for the obstacle-free car-following scenario.

States	Actions	States	Actions
s_0	ND	s_5	a_0
s_1	a_6	s_6	a_3
s_2	a_1	s_7	a_6
s_3	a_4	s_8	a_2
s_4	a_6	s_9	a_5

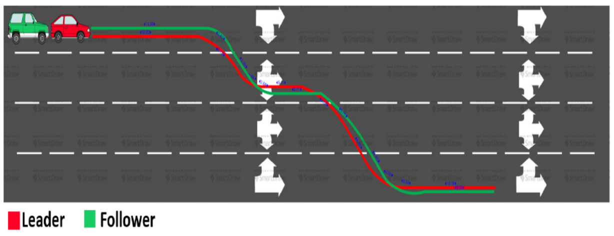


FIGURE 16. Obstacle-free trajectory.

vehicles, we define three zones as shown in Fig. 15c: A zone of crash risk A where $D_e < 1$, a safe zone B where $D_e \in [1, 10]$ and a zone of a risk of loosing the leader vehicle C where $D_e > 10$. Testing the model for 125 frames which represent the number of iterations, the follower vehicle is in zone B for the most of the time and entered in zone A three times where it is impossible to make decisions as shown in Fig. 15d. To avoid this risk, more states and actions should be added. Moreover, integrating the inputs of other technologies and sensors will significantly help in complementing the video frames based decision making.

Fig. 16 shows the trajectory of the follower (green vehicle) and the leader (red vehicle) in an obstacle-free environment. In the following process, the follower follows its leader in a general way. However, it fails in reproducing the same

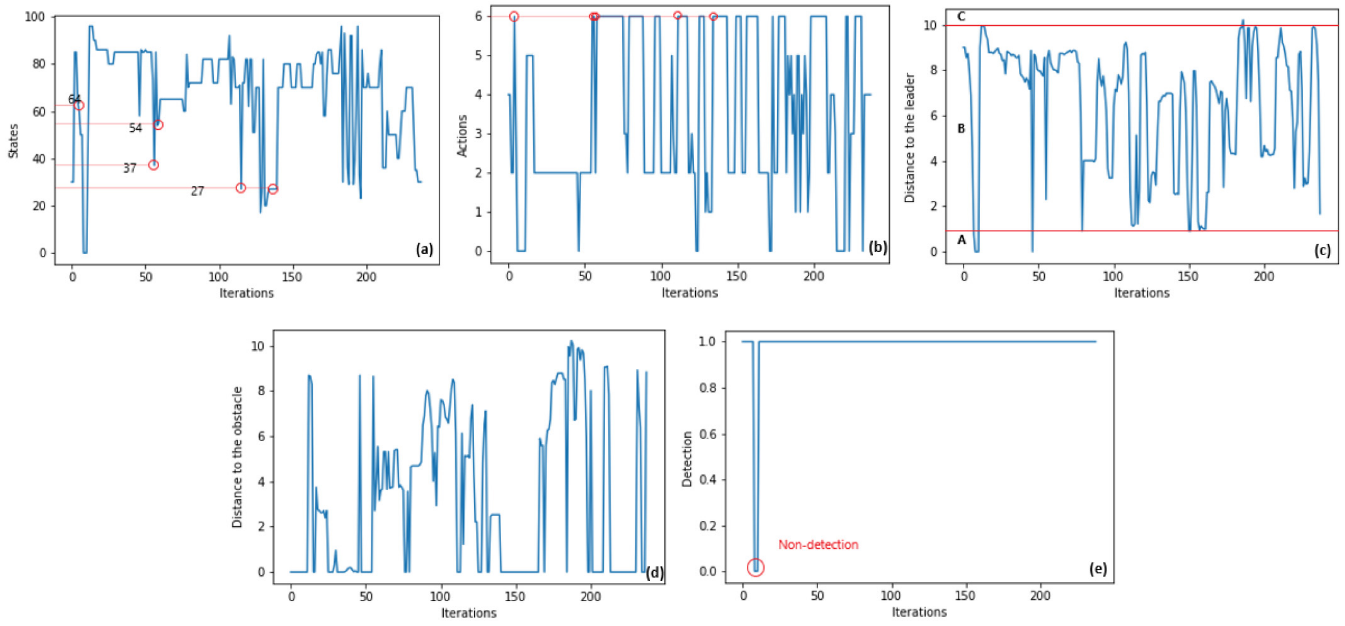


FIGURE 17. Performance of the follower during the testing phase for the obstacle-aware car-following scenario: (a) states of the follower with respect to the leader and the obstacle, (b) chosen actions by the follower, (c) relative distance between the leader and follower, (d) relative distance between the obstacle and follower, and (e) non-detection occurrence.

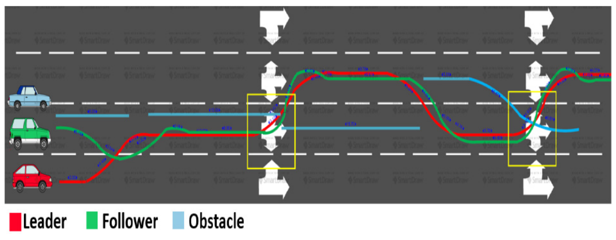


FIGURE 18. Obstacle-aware trajectory.

leader trajectory due to the complexity and the needed time for the decision making and the image processing.

2) OBSTACLE-AWARE CAR-FOLLOWING SCENARIO SIMULATION

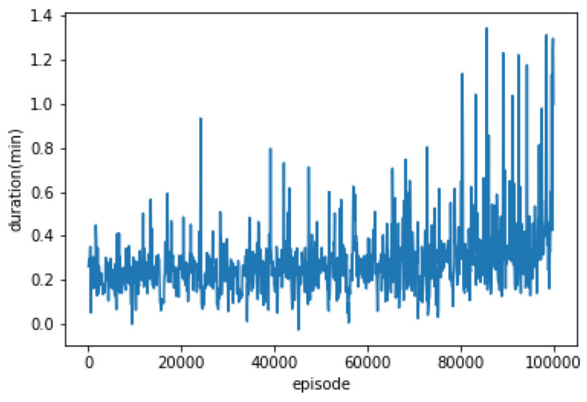
Fig. 17 shows the performance of the proposed framework during the testing phase. During this phase, we set a fixed trajectory for the leader and the obstacle vehicles in a straight road. The objective is to investigate the behavior of the AV in different states where an obstacle is presented near the leader. Testing the model for 280 frames, the follower vehicle is in zone B for the most of the time and entered in zone A one time where it is impossible to make decisions as shown in Fig. 17e. In Fig. 17a and Fig. 17b, when the leader and the obstacle are in the state 27 (leader in state 2 and obstacle in state 7), 37, 54, and 64 the follower chooses the action 6 (braking until stopping action) where an accident with the obstacle may occur in following the leader vehicle.

Fig. 18 shows the trajectory of the follower (green vehicle), the obstacle (blue vehicle), and the leader (red vehicle) in an obstacle-aware environment. In this trajectory, the obstacle is not presented all the time and changes its position in order to test different states. For this reason, we represent its trajectory in a discontinuous way. The objective is to test the interception of the leader's trajectory by the obstacle vehicle. In the following process, the follower follows its leader in a general way. However, when the obstacle intercepts the leader's trajectory (yellow rectangles), the follower starts braking until stopping to avoid a crash as explained in Fig. 17. The follower resumes its following process when the obstacle has exceeded the leader's trajectory. Due to the complexity of the process and the needed time to capture the frame and generate the output result, the follower does not reproduce the same leader's trajectory.

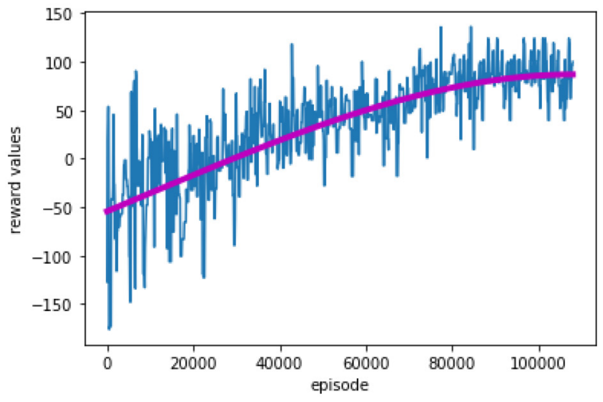
D. SIMULATION RESULTS FOR THE DQN ALGORITHM

In this section, we investigate the performance of the DQN algorithm by showing results of the training phase and statistics of the validation phase. For the training process, we set $D_{\min}$ to 10% of $D_{\max}$. This value is sufficiently high to avoid crashes with the leader. The discount factor is set to 0.4 and, for the exploration-exploitation trade-off, we set ϵ_{max} to 1 and ϵ_{min} to 0.01.

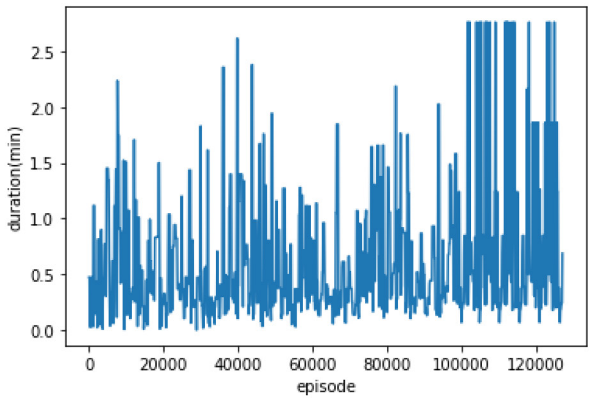
In Fig. 19, we plot the training results of the DQN model based on the RGB-D frames and the Deep Deterministic Policy Gradient (DDPG) model as described in [15], [22]. We can notice that the cumulative reward function is generally increasing over episodes until convergence at the end of the training phase where the model is not learning new possibilities from its exploration decisions. Hence, less autonomous driving and car-following errors occur due to the actions taken by the vehicle. In Fig. 19(c), we plot the duration of training episodes which reflects the driving



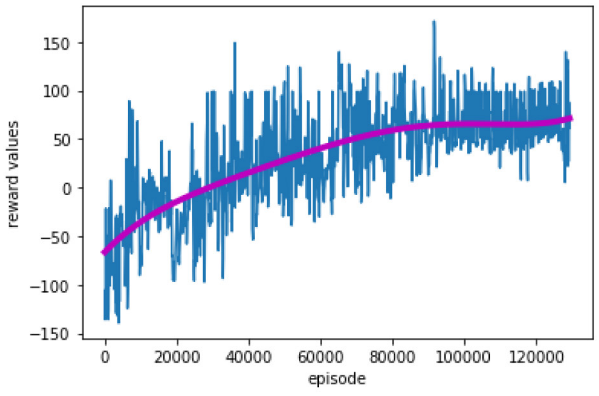
(a) DDPG: Duration of episodes



(b) DDPG :Cumulative reward



(c) DQN: Duration of episodes



(d) DQN: Cumulative reward

FIGURE 19. **Training results of the autonomous car-following framework using DQN and DDPG algorithms.**

duration of the vehicle. At early episodes, the training time is short as the vehicle crashes very early by taking wrong actions. The model makes some random mistakes during the exploration phase, which explains the fluctuation of the duration of episodes. However, with the decrease of the training loss (increase of achieve reward), i.e., the exploitation phase, the duration of the training episodes increases on average as fewer crashes occur.

During the training phases, both algorithms converge to the same range of values. However, the duration of episodes for the algorithms is significantly different. Indeed, the two models do not learn autonomous driving and car-following in the same manner, as we can notice from Fig. 19(a) where the duration of the DDPG episodes are increasing smoothly. Hence, the model is learning from its previous experience without knowing any new navigation decision or possibilities. On the contrary, the duration of the DQN episodes is wildly changing from an episode to another, and this behavior is explained by the fact that the model is attempting to explore all available navigation decisions in the environment. This attitude during the training is the success key of the algorithm that will increase, considerably, the efficiency of the model in navigating in any unknown environment.

In order to evaluate the model, we proceed by testing the capability of the model to avoid crashing with the leader vehicle or any other object in the simulated environment. In every testing episode, the leader will travel from a point A to another point B and the AV will follow the leader through the entire trajectory. Every time the follower crashes into the leader or the surrounding environment, the test is considered a failure. If the follower can successfully follow the leader across the whole trajectory without incident, the test is considered as a success.

TABLE 6. **DQN and DDPG testing statistics.**

Testing Result	Total DQN	Total DDPG
Successful driving	975	967
Crash with the leader	2	18
Crash onto the sidewalk	3	5
Detection lost with leader	20	10

The performances of the DQN model we generated were compared to those of the DDPG RL algorithm. DQN and DDPG had similar performance. DQN held an advantage, however, by minimizing crashes with the leader – a key characteristic for safe operation in real-world applications. We present the results of the comparison between DDPG and DQN in Table 6. The table provides statistics about the failing and successful incidents during the testing episodes. It is shown that the DQN slightly outperforms the DDPG algorithm in following the leader vehicle. In the testing phase, we utilize three maps, and 1000 runs are split amongst the three maps for both the DQN and the DDPG approaches. In each map, the leader vehicle moves from a point A to a point B. The DQN approach achieves success in following the leader vehicle without incidents for 975 of the 1000 runs. Among the 20 failures, the follower loses the detection of the leader vehicle. In the remaining cases, the follower crashed into the leader only twice and goes out of the road in three runs.

On the other hand, the DDPG approach succeeds in 967 of the runs, but crashes into the leader at a much higher rate of 18 runs, versus the two runs while testing the DQN approach. This DQN achieves a higher level of safety compared to DDPG since most of its failed tests correspond to a detection loss while the DDPG leads to higher crash rate either with the leader or the sidewalk. We show that the DQN model rapidly learns that collisions and detection loss are very high-priority objectives that must be avoided and hence, it delivers in most of the cases safe navigation. Recall that these tests are executed for short operations of the vehicles and further improvement are required to enhance the performance as it will be discussed in the next section.

VII. CONCLUSION

The experimental results have demonstrated that, with real-time video frames only, the proposed two-phased AI-driven approach is enabling the AV to make accurate decisions to follow a given leader vehicle. Our objective was to show that with the QL and DQN models implemented back-to-back to a computer vision module, we were able to build a car-following framework for AVs with a limited source of information. In order to improve the efficiency of the proposed approach, it is potentially possible to switch between both RL algorithms given the driving situation, the speed of the AV, and other environmental features. Indeed, it is shown that the Q-learning is very efficient in making rapid self-driving decisions with low training time but with limited accuracy. However, the DQN can provide much accurate decisions but requires more computational resources.

The main challenge in training these RL algorithms in real-world is to find an efficient way to train them for diverse and different environments such that the training time is significantly accelerated. Novel and rapid transfer learning techniques can play a significant role in improving the efficiency and rapidity of the computer vision and RL algorithms. Moreover, in practice, safety is the most important criterion for the AVs. Hence, it is necessary to achieve high levels of accuracy to guarantee a safe and secure navigation. Therefore, as a future work and based on these promising results, the car-following problem can be significantly enhanced for AVs by integrating other object detection components such as LiDAR and RADAR and data fuse all these inputs to make more effective self-driving decisions not only in straight roads.

The car-following problem is usually addressed for the case of a single leader followed by a single follower. However, with the expected proliferation of AVs, it is worth to revamp the problem by introducing an extra layer of cooperation among the AVs and devise multi-agent coordinated reinforcement learning approaches for better and cooperative decision making. Enabling effective car-following approaches will help improving the navigation and management of future AVs circulating in smart city roads. Indeed, it brings them out from a closed and isolated system where AVs operate solely to a more cooperative and open system where AVs share much more information, e.g., about the traffic, their navigation experience, and the environment, which will help better the assessment of the road situation and hence, improve the navigation of AVs.

REFERENCES

[1] M. Masmoudi, H. Ghazzai, M. Frikha, and Y. Massoud, "Autonomous car-following approach based on real-time video frames processing," in *Proc. IEEE Int. Conf. Veh. Electron. Safety (ICVES)*, Cairo, Egypt, Sep. 2019, pp. 1–6.

[2] C. Zhao, S. Jiang, Y. Lei, and C. Wang, "A study on an anthropomorphic car-following strategy framework of the autonomous coach in mixed traffic flow," *IEEE Access*, vol. 8, pp. 64653–64665, 2020.

[3] Y. Zhang, P. Ni, M. Li, H. Liu, and B. Yin, "A new car-following model considering driving characteristics and preceding vehicle's acceleration," *J. Adv. Transp.*, vol. 2017, Oct. 2017, Art. no. 2437539. [Online]. Available: https://www.hindawi.com/journals/jat/2017/2437539/

[4] Y. Guo, Q. Sun, R. Fu, and C. Wang, "Improved car-following strategy based on merging behavior prediction of adjacent vehicle from naturalistic driving data," *IEEE Access*, vol. 7, pp. 44258–44268, 2019.

[5] D. Yang, L. Zhu, Y. Liu, D. Wu, and B. Ran, "A novel car-following control model combining machine learning and kinematics models for automated vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 6, pp. 1991–2000, Jun. 2019.

[6] M. C. Lucic, X. Wan, H. Ghazzai, and Y. Massoud, "Leveraging intelligent transportation systems and smart vehicles using crowdsourcing: An overview," *Smart Cities*, vol. 3, no. 2, pp. 341–361, 2020.

[7] S. Lefevre, A. Carvalho, and F. Borrelli, "Autonomous car following: A learning-based approach," in *Proc. IEEE Intell. Veh. Symp. (IV)*, 2015, pp. 920–926.

[8] Z. Z. M. Kassas, M. Maaref, J. J. Morales, J. J. Khalife, and K. Shamei, "Robust vehicular localization and map matching in urban environments through IMU, GNSS, and cellular signals," *IEEE Intell. Transp. Syst. Mag.*, vol. 12, no. 3, pp. 36–52, 2020.

[9] L. Claussmann, M. Revilloud, D. Gruyer, and S. Glaser, "A review of motion planning for highway autonomous driving," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 5, pp. 1826–1848, May 2020.

[10] L. Wang and B. K. P. Horn, "On the stability analysis of mixed traffic with vehicles under car-following and bilateral control," *IEEE Trans. Autom. Control*, vol. 65, no. 7, pp. 3076–3083, Jul. 2020.

[11] S. Milani, H. Khayyam, H. Marzbani, W. Melek, N. L. Azad, and R. N. Jazar, "Smart autodriver algorithm for real-time autonomous vehicle trajectory control," *IEEE Trans. Intell. Transp. Syst.*, early access, Oct. 21, 2020, doi: 10.1109/TITS.2020.3030236.

[12] M. Da Lio, R. Dona, G. P. R. Papini, F. Biral, and H. Svensson, "A mental simulation approach for learning neural-network predictive control (in self-driving cars)," *IEEE Access*, vol. 8, pp. 192041–192064, 2020.

[13] S. Aradi, "Survey of deep reinforcement learning for motion planning of autonomous vehicles," *IEEE Trans. Intell. Transp. Syst.*, early access, Sep. 30, 2020, doi: 10.1109/TITS.2020.3024655.

[14] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," Apr. 2018. [Online]. Available: arXiv:1804.02767v1.

[15] H. Friji, H. Ghazzai, H. Besbes, and Y. Massoud, "A DQN-based autonomous car-following framework using RGB-D frames," in *Proc. IEEE Global Conf. Artif. Intell. Internet Things (GCAIoT)*, 2020, pp. 1–6.

[16] L. A. Pipes, "An operational analysis of traffic dynamics," *J. Appl. Phys.*, vol. 24, no. 3, pp. 274–281, Mar. 1953.
[17] T. Forbes, *Human Factor Considerations in Traffic Flow Theory*. Washington, DC, USA: Highway Res. Rec., Apr. 1956.
[18] H. Ou and T.-Q. Tang, "An extended two-lane car-following model accounting for inter-vehicle communication," *Physica A Stat. Mech. Appl.*, vol. 495, pp. 260–268, Apr. 2018.
[19] H. A. Ameen *et al.*, "A deep review and analysis of data exchange in vehicle-to-vehicle communications systems: Coherent taxonomy, challenges, motivations, recommendations, substantial analysis and future directions," *IEEE Access*, vol. 7, pp. 158349–158378, 2019.
[20] Q. Xue, K. Wang, J. J. Lu, and Y. Liu, "Rapid driving style recognition in car-following using machine learning and vehicle trajectory data," *J. Adv. Transp.*, vol. 2019, p. 11, Jan. 2019. [Online]. Available: https://www.hindawi.com/journals/jat/2019/9085238/
[21] X. Wang, R. Jiang, L. Li, Y. Lin, X. Zheng, and F.-Y. Wang, "Capturing car-following behaviors by deep learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 3, pp. 910–920, Mar. 2018.
[22] M. Zhu, X. Wang, and Y. Wang, "Human-like autonomous car-following model with deep reinforcement learning," *Transp. Res. C Emerg. Technol.*, vol. 97, pp. 348–368, Dec. 2018.
[23] M. M. Abbas, B. Higgs, L. Chong, and A. Medina, "Combined car-following and unsafe event trajectory simulation using agent based modeling techniques," in *Proc. Winter Simulat. Conf. (WSC)*, Dec. 2012, pp. 1–10.
[24] M. Zhou, Y. Yu, and X. Qu, "Development of an efficient driving strategy for connected and automated vehicles at signalized intersections: A reinforcement learning approach," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 1, pp. 433–443, Jan. 2020.
[25] H. Gao, G. Shi, G. Xie, and B. Cheng, "Car-following method based on inverse reinforcement learning for autonomous vehicle decision-making," *Int. J. Adv. Robot. Syst.*, vol. 15, no. 6, pp. 1–11, Oct. 2018.
[26] M. Masmoudi, H. Ghazzai, M. Frikha, and Y. Massoud, "Object detection learning techniques for autonomous vehicle applications," in *Proc. IEEE Int. Conf. Veh. Electron. Safety (ICVES)*, Sep. 2019, pp. 1–5.
[27] W. Liu and D. Anguelov, "SSD: Single shot multibox detector," Dec. 2016. [Online]. Available: arXiv.abs/1512.02325.
[28] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 936–944.
[29] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.
[30] T. Zhe, L. Huang, Q. Wu, J. Zhang, C. Pei, and L. Li, "Inter-vehicle distance estimation method based on monocular vision using 3D detection," *IEEE Trans. Veh. Technol.*, vol. 69, no. 5, pp. 4907–4919, Mar. 2020.
[31] R. Munos, T. Stepleton, A. Harutyunyan, and M. G. Bellemare, "Safe and efficient off-policy reinforcement learning," 2016. [Online]. Available: arXiv.abs/1606.02647.
[32] R. Bellman, *Dynamic Programming* (Dover Books on Computer Science). London, U.K.: Dover, Apr. 2013. [Online]. Available: https://books.google.com/books?id=CG7CAgAAQBAJ
[33] T. Okuyama, T. Gonsalves, and J. Upadhay, "Autonomous driving system based on deep *Q* learnig," in *Proc. Int. Conf. Intell. Auton. Syst. (ICoIAS)*, 2018, pp. 1–9.

MEHDI MASMOUDI received the Diplôme d'Ingénieur degree (Hons.) in telecommunication engineering from the École Supérieure des Communications de Tunis, Tunis, Tunisia, in 2019. In 2019, he worked as a Research Assistant with the Stevens Institute of Technology, Hoboken, NJ, USA with interest in video processing, object detection and autonomous vehicules. He is currently a Technology Services Consultant with EY with an interest in data analytics, digital transformation, and strategic management.

HAMDI FRIJI received the National Diploma of Engineering degree (Hons.) in information and communication technology from the Higher School of Communication of Tunis, University of Carthage, Tunisia, in 2020. He is currently working as a Research Assistant with the Stevens Institute of Technology, Hoboken, NJ, USA. His study area is the intersection of artificial intelligence, computer vision, mathematical modeling, optimization, the Internet of Things, and graph theory.

HAKIM GHAZZAI (Senior Member, IEEE) received the Diplome d'Ingenieur degree (Hons.) in telecommunication engineering and the master's degree in high-rate transmission systems from the École Superieure des Communications de Tunis, Tunis, Tunisia, in 2010 and 2011, respectively, and the Ph.D. degree in electrical engineering from KAUST, Saudi Arabia, in 2015. He was a Visiting Researcher with Karlstad University, Sweden, and a Research Scientist with the Qatar Mobility Innovations Center, Doha, Qatar, from 2015 to 2018. He is currently a Research Scientist with the Stevens Institute of Technology, Hoboken, NJ, USA. He has authored over 130 articles in peer-reviewed journals and conferences. He is on the editorial board of the IEEE COMMUNICATIONS LETTERS, the IEEE OPEN JOURNAL OF THE COMMUNICATIONS SOCIETY, and *Frontiers in Communications and Networks*. His general research interests are in the areas of wireless networks, UAVs, Internet-of-Things, and intelligent transportation systems.

YEHIA MASSOUD (Fellow, IEEE) received the Ph.D. degree in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, MA, USA. He is currently the Dean of the School of Systems and Enterprises, Stevens University of Science and Technology, Hoboken, NJ, USA. He was selected as one of ten MIT Alumni Featured by MIT's Electrical Engineering and Computer Science Department in 2012. He has held several academic and industrial positions, including a member of the Technical Staff with the Advanced Technology Group, Synopsys, Inc., Mountain View, CA, USA, a Tenured Faculty Member with the Departments of Electrical and Computer Engineering and Computer Science, Rice University, Houston, USA, the W. R. Bunn Head of the Department of Electrical and Computer Engineering, UAB, Birmingham, USA, and the Head of the Department of Electrical and Computer Engineering, Worcester Polytechnic Institute, USA. He has authored over 325 articles in peer-reviewed journals and conferences. He was a recipient of the Rising Star of Texas Medal in 2007, the National Science Foundation CAREER Award in 2005, the DAC Fellowship in 2005, the Synopsys Special Recognition Engineering Award in 2000, and several best paper award nominations. He has served as an Editor of Mixed-Signal Letters—The Americas and also as an Associate Editor of IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS and IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I: REGULAR PAPERS. He has served as a Distinguished Lecturer by the IEEE Circuits and Systems Society and as an elected member of the IEEE Nanotechnology Council.