Article

Reinforcement Learning-Based Autonomous Driving at Intersections in CARLA Simulator

Rodrigo Gutiérrez-Moreno, Rafael Barea *, Elena López-Guillén, Javier Araluce and Luis M. Bergasa

Department of Electronics, University of Alcalá, 28805 Alcalá de Henares, Spain
* Correspondence: rafael.barea@uah.es

Abstract: Intersections are considered one of the most complex scenarios in a self-driving framework due to the uncertainty in the behaviors of surrounding vehicles and the different types of scenarios that can be found. To deal with this problem, we provide a Deep Reinforcement Learning approach for intersection handling, which is combined with Curriculum Learning to improve the training process. The state space is defined by two vectors, containing adversaries and ego vehicle information. We define a features extractor module and an actor–critic approach combined with Curriculum Learning techniques, adding complexity to the environment by increasing the number of vehicles. In order to address a complete autonomous driving system, a hybrid architecture is proposed. The operative level generates the driving commands, the strategy level defines the trajectory and the tactical level executes the high-level decisions. This high-level decision system is the main goal of this research. To address realistic experiments, we set up three scenarios: intersections with traffic lights, intersections with traffic signs and uncontrolled intersections. The results of this paper show that a Proximal Policy Optimization algorithm can infer ego vehicle-desired behavior for different intersection scenarios based only on the behavior of adversarial vehicles.

Keywords: reinforcement learning; decision making; autonomous driving

Citation: Gutiérrez-Moreno, R.; Barea, R.; López-Guillén, E.; Araluce, J.; Bergasa, L.M. Reinforcement Learning-Based Autonomous Driving at Intersections in CARLA Simulator. *Sensors* **2022**, *22*, 8373. https://doi.org/10.3390/s22218373

Academic Editor: Marco Giordani

Received: 3 October 2022
Accepted: 28 October 2022
Published: 1 November 2022

1. Introduction

Intersections in urban environments are one of the most complex and challenging scenarios in autonomous driving. These scenarios are accident-prone; numerous traffic crash statistics show that 60% of severe traffic injuries in Europe are related to intersections [1]. In the US, the National Highway Traffic Safety Administration (NHTSA) publishes an annual report related to traffic safety [2]. These data show that 29% of all car crashes and 18% of pedestrian fatalities occur at intersections. For these reasons, an autonomous vehicle needs to make safe and efficient decisions and develop reliable driving strategies [3].

One of the main reasons for this high accident rate is the large amount of information that the driver perceives from the environment. For example, in an intersection, the driver has to be aware of traffic signals such as stop signs or traffic lights and at the same time has to monitor other vehicles and estimate their velocities and intentions. This requires great attention to driving when making a decision. Therefore, developing an agent that allows safe and reliable decisions is a hard task to implement manually [4].

In the last few years, several strategies have already been applied to find a solution for the intersections challenge. These approaches are mainly focused on making decisions considering motion prediction and collaborations by applying vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) technologies [5,6]. Currently, the most widely used agent is based on time-to-collision (TTC). This parameter has been used in multiple works as a safety indicator, both in research and the automotive industry [7,8]. The main problem is that some tuning parameters have to be adjusted, and this task can be laborious because of its dependence on environmental situations. As a result, this agent requires several rules to deal with different scenarios in an urban environment, and therefore, it is not reliable enough for autonomous driving. On the other hand, several machine learning based approaches have been implemented to tackle the intersection problem. These approaches include imitation learning, where the policy is learned from a human driver, and reinforcement learning, where an agent learns in an interactive environment by trial and error using feedback from its actions and experiences [9,10].

In this paper, we provide a Deep Reinforcement Learning (DRL) approach for intersection handling that combines DRL with Curriculum Learning (CL) [11]. Our goal is to train an algorithm that can autonomously learn to solve different types of intersections and accelerate the training by incorporating reinforced CL into the process. To do this, we propose a two-stage training method. A simple state representation and a light simulator such as SUMO [12] allow a fast first training stage. The obtained model is used as a prior in a second training stage in a realistic CARLA simulator where vehicle dynamics are introduced [13]. Finally, we present performance in the two simulators focusing on the domain adaptation problem.

1.1. Related Work

The first RL approaches in autonomous vehicles were focused on maneuvers execution [14]. Learning from environment data, these agents directly provide the steering and braking control inputs for the vehicle to execute driving maneuvers in simple scenarios. However, traditionally classic algorithms have been used for low-level control, obtaining good performance [15,16], while RL approaches have slower learning rates and poorer performance. Therefore, we propose to use RL in high-level decision making in much more complex scenarios with multiple adversarial vehicles.

Separating the planning task into high-level decision making and maneuver execution [17] is a way of handling this complexity. An execution layer is in charge of the motion, while a decision-making layer executes the high-level actions. This is known as the hierarchical approach [18].

A second approach that can directly infer control commands from an input obtained directly from sensors is end-to-end learning [9,10]. With the last year’s improvements in deep learning algorithms and the modern hardware computation capabilities, researchers at NVIDIA proposed an end-to-end supervised learning method based on CNN capable of directly steering an automobile [19]. From this pioneering work, many others have used this approach [20,21].

In particular, we review the high-level decision making RL-based approaches at intersections. A Deep Q-Network (DQN) algorithm to deal with intersections is commonly used [22]. In addition, in [23,24], a DQN algorithm to deal with different driver behaviors is used. Some works such as [25,26] train at occluded intersections using a risk-based reward function. The input vector contains information about the ego vehicle, the existing lanes, and the surrounding vehicles. Others, such as [27], are focused on the intentions of the adversarial vehicle, where an RL agent based on cooperation with adversarial vehicles in dense traffic is presented. This author also designs a curriculum learning agent using the concept of level-k behavior [28]. Furthermore, a POMDP-based low-level planner is defined in [29], achieving safe planning results with high-commute efficiency at unsignalized intersections in real time. In [30], an invariant environment representation from the perspective of the ego vehicle based on an occupancy approach is proposed. The agents are capable of performing successfully in unseen scenarios. An additional study with a simple occlusion model is described in this work, where the simulation environment is generated using SUMO. There are also some studies where the state vector is obtained directly from sensor data, such as [3], where a monocular camera is used.

Since our architecture is trained using a curriculum reinforcement learning approach, we present a review of this topic. This process trains first a model with easy tasks and then gradually increases the difficulty of the tasks, improving the performance of the model. A survey of CL can be found in [31,32]. In [4], an automatically generated curriculum-based reinforcement learning for autonomous vehicles in an urban environment is proposed and demonstrates a significant reduction of training time, achieving better performance than traditional methods in intersection traversing and approaching scenarios. Finally, in [11], an end-to-end competitive driving policy for the CARLA simulator is implemented.

1.2. Contribution

This paper focuses on developing a safe high-level decision-making framework for different types of intersections. A training environment in which the RL agent will be trained under different situations is defined for this purpose. More specifically, in this work, we present the following achievements:

- An RL agent capable of learning the desired policy for three types of intersections. Most approaches in the literature use RL to estimate the intentions of the adversarial vehicles, and the agent uses this information to cross an intersection. In our proposal, without any prior information about the scenario, our agent infers not only the intentions of the adversarial vehicles but also the type of intersection.
- A PPO algorithm with a preprocessing layer for the tactical level. We compare the training performance and the results obtained for both architectures with and without the features extractor. This module is introduced using a hybrid AD architecture, and the whole architecture is validated in a realistic simulator, which includes vehicle dynamics. This approach is an alternative to the trendy end-to-end approaches, showing good results in complex scenarios where end-to-end options do not converge.
- A novel two-stage training method. We use a model trained in a light simulator as a prior model for training in a realistic simulator. This allows convergence in CARLA within a reasonable training time.

2. Hybrid Architecture

This section addresses a completely autonomous driving system, including high-level decision making and maneuver control in a hybrid architecture. It is divided into three levels, as is described in Figure 1. The strategy level defines the trajectory; the tactical level is in charge of the high-level decisions, and the operative level executes the control maneuvers. This paper is focused on the tactical level.

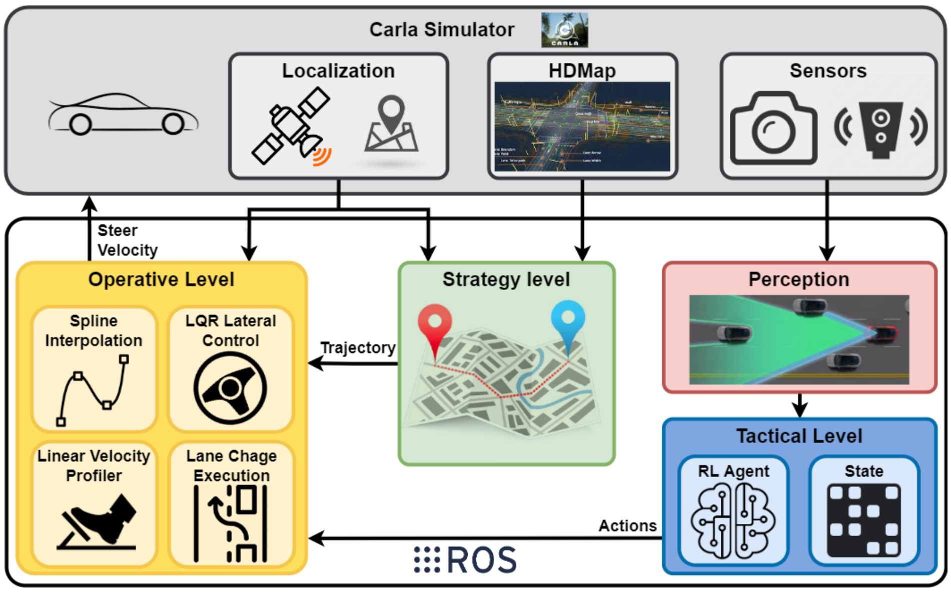

Figure 1. The proposed hybrid architecture. The strategy level defines a trajectory with the map information and the ego vehicle location. The tactical level executes high-level actions in correlation with the perception information. The operative level combines the trajectory and the actions, calculating the driving commands.

2.1. Strategy Level

The path-planning module is described in our previous paper [33]. This module first generates a directed graph of roads and lanes using the HD map input. Then, it calculates a topological (road-lane) route between the ego vehicle location, which is provided by the simulator, and the goal location, applying the Dijkstra algorithm to the previously generated graph. Finally, the route is calculated as a list of way-points centered in the lanes and separated by a given distance. The route is calculated and published in a Robot Operating System (ROS) [34] topic every time a new goal location is set. The road information, such as velocity limit, road lanes, or road type, is also defined in this ROS message.

2.2. Tactical Level

High-level decisions are executed at the tactical level, where the simulator information is processed to define the state vector. The RL agent executes an action at every time step. This level will be explained in detail in Section 4.

2.3. Operative Level

The maneuver execution is carried out by a classic controller [35] that performs a smooth interpolation of the way-points given by the planner. Before the navigation starts, a velocity profile is generated using the curvature radius of each trajectory section and the velocity limit defined for these sections. During the navigation, the velocity command is adjusted using this profile, and the steering command is set using Linear Quadratic Regulator (LQR) techniques to ensure trajectory tracking and a smooth lane change execution.

2.4. Perception

The perception data are directly obtained from the simulator's ground truth. In real applications, these data could be obtained using the fusion of cameras and lidars.

3. Background

A Markov decision process (MDP) is defined by a tuple (S, A, P, R) in which S is a set of states, A is a set of actions, P is the probability that action a in state s at time t will lead to state s' at time $t+1$ and R is the reward function. An agent in state $s \in S$ takes an action $a \in A$ transitioning to s' and receiving a reward $R(s, a, s')$, as shown in Figure 2. In an MDP, the agent receives a s' directly. The optimal policy $\pi^*(s)$ maps a state to action.

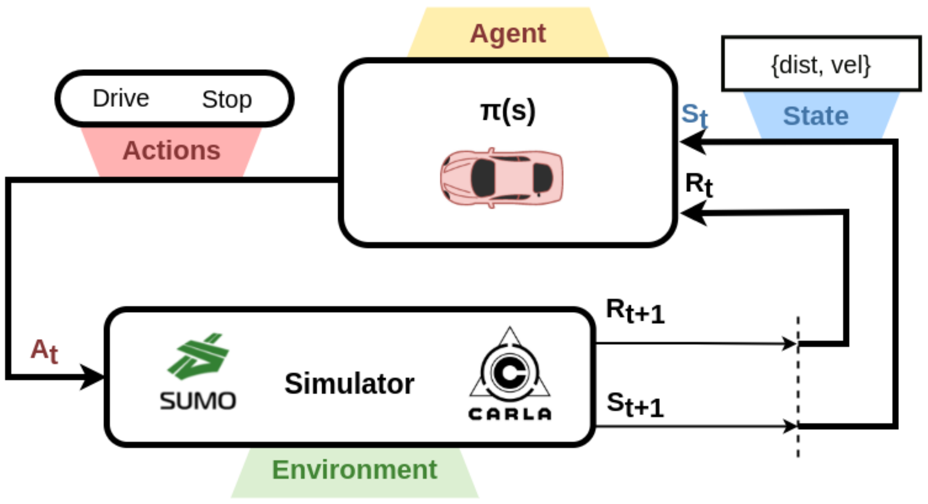

Figure 2. The state vector is extracted from the simulator. The agent selects an action considering this matrix and the reward for this action.

In this work, we propose DRL to solve MDPs. Our approach focuses on a policy-based method, which directly learns the policy function that maps state to action. These methods work by computing an estimator of the policy gradient and plugging it into a stochastic gradient ascent algorithm. The loss function for updating an RL policy has the general form:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t[log\pi_\theta(a_t|s_t)\hat{A}_t] \tag{1}$$

where $\mathbb{E}_t$ is the expectation, π_θ is the policy and $\hat{A}_t$ is an estimator of the advantage function at a time step t.

While it is appealing to perform multiple steps of optimization on this loss function $L^{PG}(\theta)$, doing so is not well-justified, and empirically, it often leads to destructively large policy updates.

In particular, we use the Proximal Policy Optimization (PPO) algorithm [36], which proposes a clipped surrogate loss function and combines the policy surrogate and a value function error term:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)] \tag{2}$$

where c_1, c_2 are coefficients, S denotes an entropy bonus, and $L_t^{VF}(\theta)$ is a squared-error loss $(V_\theta(s_t) - Vt^{targ})^2$. L_t^{CLIP} is the clipped surrogate objective, which takes the form:

$$L_t^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{3}$$

where epsilon is a hyperparameter and $r_t(\theta)$ is the probability ratio:

$$r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{old}}(a_t|s_t) \tag{4}$$

The value of L_t^{CLIP} is the minimum of the clipped and the unclipped objective, so the final objective is a lower bound on the unclipped objective. Large policy updates are avoided by clipping this probability ratio inside the interval $[1-\epsilon, 1+\epsilon]$.

The PPO algorithm runs the policy for T timesteps within a given length T trajectory segment and uses the collected samples for an update. A truncated version of generalized advantage estimation is the choice taken, which has the form:

$$\begin{gathered} \hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \\ where\ \delta_t = r_t + \gamma V(s_t+1) - V(s_t) \end{gathered} \tag{5}$$

In each iteration, each of N (parallel) actors collects T timesteps of data. The surrogate loss function is constructed on these NT timesteps of data and is optimized with minibatch SGD for K epochs.

4. Reinforcement Learning at Intersections

This section describes the application of RL techniques to realistic environments. First, we define the representation of the intersection scenarios; then, we explain the possible scenarios the vehicle will find at intersections; and finally, the neural network structure and the features extractor are described.

4.1. Modeling Intersections

Our goal is to use an RL agent to execute high-level decisions at intersection scenarios, dealing with a continuous state vector and a set of discrete actions. For this purpose, we define the intersection scenarios as an MDP.

4.1.1. State

The state of a vehicle, as shown in Figure 3, is defined by its distance to the intersection point and its longitudinal velocity: $s_i = \{d_i, v_i\}$. These values are normalized between

$[0, 1]$. We define the state vector as the collection of the individual states of the two closest vehicles for each lane and the ego vehicle.

$$s = \{d_{ego}, v_{ego}, d_{adv1}, v_{adv1}, d_{adv2}, v_{adv2}, d_{adv3}, v_{adv3}, d_{adv4}, v_{adv4}\} \tag{6}$$

4.1.2. Action

We propose a discrete action space formed by just two high-level actions. The low-level controller is in charge of performing smooth driving based on these actions. These actions are focused on when the ego vehicle has to cross the intersection and when it has to stop. The action space is defined as: $a = \{stop, drive\}$. Both actions set a desired velocity, *stop* sets 0 m/s and *drive* sets the nominal velocity of 5 m/s.

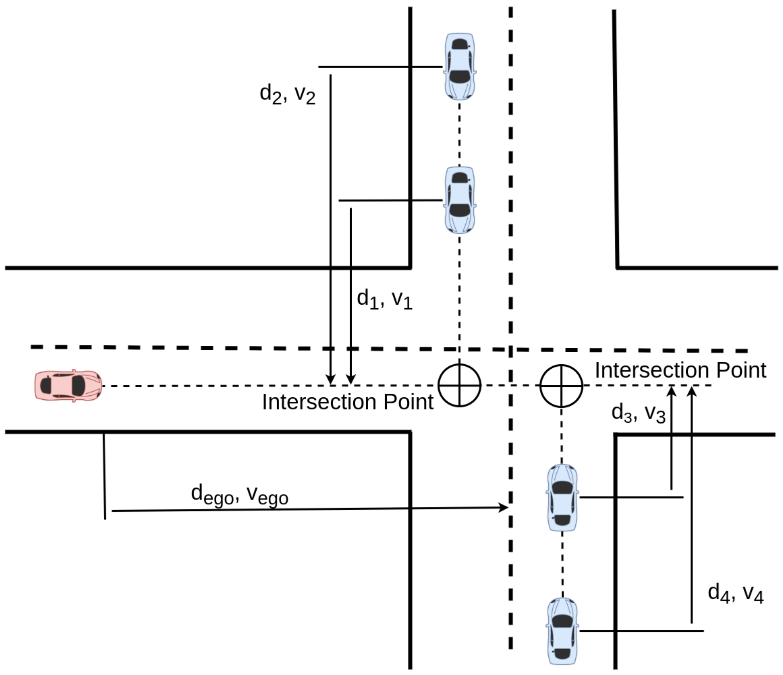

Figure 3. The ego vehicle (red) and the adversary (blue) are represented. The velocity of each vehicle and the distances to the intersection point are represented. This point is defined by the intersection of the trajectories of the vehicles.

4.1.3. Reward

An RL algorithm aims to maximize the expectation of the discounted future reward. This means that different reward functions will lead to different optimal policies. Our goal is to drive through the intersection as fast as possible, avoiding the adversarial vehicles, being the reward function defined in terms of success or failure. A negative reward is given when a collision takes place, and a positive reward is given when the vehicle reaches the success point once it has crossed the intersection. To encourage the ego vehicle to move, we propose a cumulative reward based on its longitudinal velocity. In addition, when an episode ends, a negative reward is given relative to its duration, where t is the training episode duration and t_{out} is the timeout time. The reward function consists of the following terms:

- Reward based on the velocity: $k_v * v_{ego}$;
- Reward for crossing the intersection: $+1$;
- Penalty for collisions: -2;
- Penalty relative to the episode duration: $-0.2t/t_{out}$.

Large reward values can lead to large gradients according to [37]. Therefore, all the rewards are scaled to keep the total reward $r_t \in [-1, 1.2]$.

4.2. Intersection Scenarios

We present the hypothesis that the RL agent (ego vehicle) is capable of driving only based on the position and speed of the adversarial vehicles (adversaries) since the type of intersection is not given in the state vector. To prove this, we have designed three different scenarios, all of them with the same dimensions but with different features: traffic light intersection, stop sign intersection and uncontrolled intersection.

The complexity of managing regulatory elements, such as a stop signal or a traffic light, is solved using classic approaches in the decision-making module. We implement a simple high-level module for this purpose in order to obey these traffic rules. As a consequence, we evaluate the ego vehicle without this module. The RL agent should infer the type of intersection and adapt its behavior only based on the information of the adversaries. We just focus on crossing the intersection in the fastest way ensuring collision avoidance.

The definition and the implementation of these scenarios are completed in two simulators, first in SUMO, which does not consider vehicle dynamics and in CARLA, in which vehicle dynamics are included. In this section, we focus on the definition of the physical environment and the behavior of the vehicles. Traffic generation is one of the pillars to obtain a realistic simulation.

In SUMO, the traffic density and vehicle features are similar for the three scenarios. A new vehicle is generated in the simulation every $sec \in [5, 10]$ s. These vehicles are 4 m long and follow a standard Intelligent Driver Model (IDM) movement [38] with a target velocity of $v \in [4, 6]$ m/s. The three scenarios are defined by two roads with two lanes each. The length of these roads is 200 m, and they are perpendicular to the intersection point in the middle.

In CARLA, six vehicles are spawned in random locations close to the intersection. These vehicles start moving when the ego vehicle approaches. They are commanded by the CARLA Autopilot, which has different driving modes defined by the CARLA Traffic Manager. All these vehicles have the dimensions of a Tesla Model 3 and a target velocity $v \in [4, 6]$ m/s. These roads are 50 m long, and they are perpendicular to the intersection point in the middle.

We need to define the scenarios in the two simulators slightly different to obtain similar intersection scenarios. In CARLA, the simulation is slower than in SUMO, so we use a shorter road and generate fewer adversaries, but the ego vehicle faces a similar situation when it approaches the intersection.

4.2.1. Uncontrolled Intersection

This is the most difficult scenario, since the ego vehicle has to deal with many different behaviors by the adversaries. Their behavior may vary depending on the ego vehicle and the other adversaries, since they do not follow signal rules. The ego vehicle and the adversaries all continue straight when crossing the intersection. This route, described in Figure 4a, is the same for the three intersection scenarios.

In the absence of a signal that regulates the right of way, a driver is obliged to yield to vehicles approaching from his right. In this scenario, the adversaries tend to follow this rule, but in some cases, this may not happen.

4.2.2. Traffic Light Intersection

Solving controlled intersections may seem a deterministic problem just by following the signals, but in autonomous driving, this is not the case. The task of determining the state of the traffic light is quite hard due to the huge variety of these in different countries.

We propose a novel approach to this problem, learning the desired behavior based on the other vehicles. In this way, collisions can be avoided even though the ego vehicle misses the traffic light. For this purpose, we set four traffic lights at the intersection, one for each lane. A predefined sequence for the simulation is defined in Table 1.

Table 1. Traffic Lights Sequence.

Phase	Duration (s)	State
1	20	GrGr
2	2	yryr
3	20	rGrG
4	2	ryry

Each episode starts at a random phase $p \in [1, 4]$. The adversaries follow the traffic light rules and do not yield if the light is green.

In Figure 4b, the traffic light scenario is represented. Following the SUMO notation, the four traffic lights states are described as *GrGr*, where the traffic lights are green "*G*" and red "*r*", alternatively.

4.2.3. Stop Sign Intersection

The complexity of this scenario is that the ego vehicle must learn to yield. As described in Figure 4c, the ego vehicle has a stop signal in its lane while the adversaries can freely cross the intersection. These adversaries never stop, forcing the ego vehicle to stop and cross when there is a big enough gap.

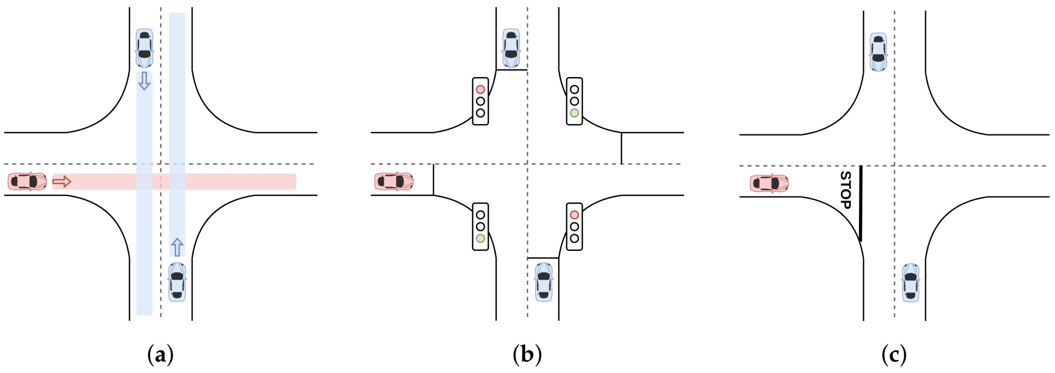

Figure 4. Definition of the three intersection scenarios. (**a**) Uncontrolled intersection scenario: The ego vehicle (red) follows the red trajectory and the adversaries (blue) follow the blue trajectory. (**b**) Traffic light intersection scenario: The ego vehicle (red) has a green light while the adversaries (blue) have a red light. (**c**) Stop intersection scenario: The ego vehicle (red) has a stop signal while the adversaries (blue) can freely cross the intersection.

4.3. Neural Network Architecture

We propose a features extractor module and an actor–critic approach for the RL agent. The algorithm structure is shown in Figure 5.

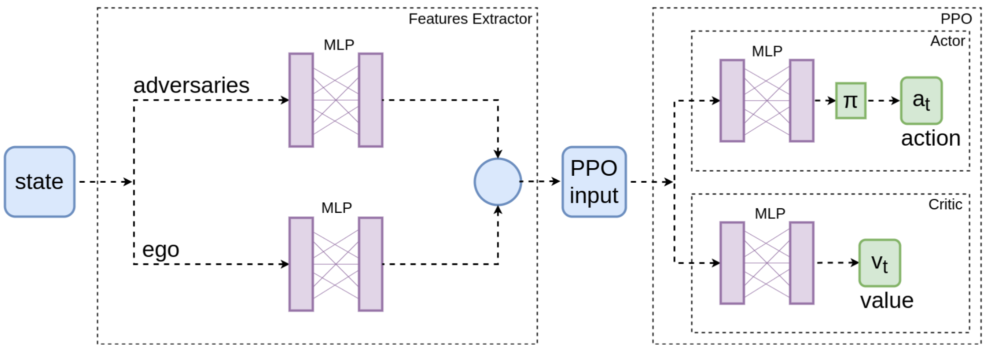

Figure 5. Neural network: After two fully connected layers, both adversaries and ego vehicle features are concatenated and fed the actor–critic structure with two layers of 128 neurons each.

The state vector is defined by the distances and velocities of the vehicles in the scenario as described in Section 4.1.1. Both adversaries and ego vehicle features are introduced in the extractor separately and concatenated in a single vector. The features extractor is shared between the actor/critic network being tuned in an unsupervised way. The concatenated vector is the PPO input. This algorithm has two models, the actor and the critic. The actor chooses the action and corresponds with the policy, while the critic corresponds with the value function. Both models are defined with a simple dense multi-layer perceptron (MLP) formed by two hidden layers of 128 neurons each. By keeping the network simple, we prevent overfitting and achieve faster training.

5. Experiments

In this section, we present some details about the training methodology, the evaluation metrics and the obtained results.

5.1. Training the Algorithm

We present a two-stage training method. We propose a first training stage, with a large number of episodes in a simple simulator. This training stage has two phases. In the first phase, adversaries are only generated in only one lane, and in the second phase, adversaries are generated in both lanes. The second training stage, with is completed with the prior model, is in charge of adapting this model to a realistic simulator. The training methodology is described in Figure 6.

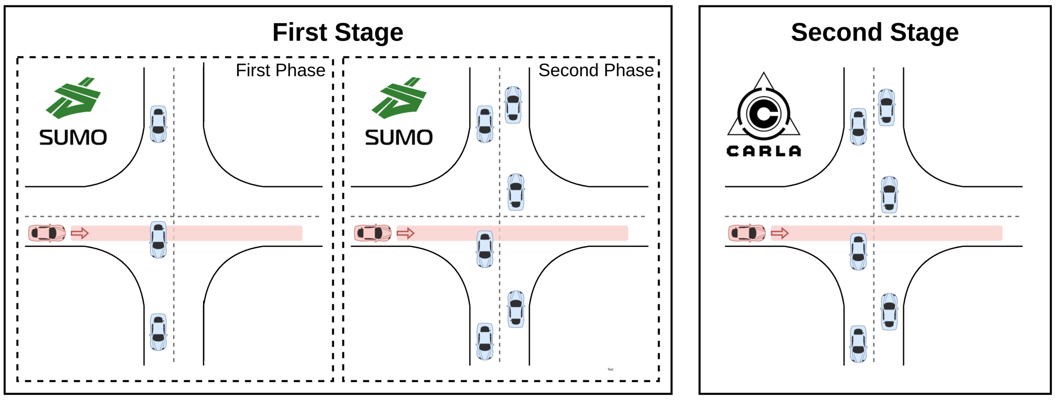

Figure 6. Training methodology: The first stage in SUMO with two phases of increasing complexity and the second stage in CARLA.

In the first training stage, velocity commands and maneuvers are defined by SUMO, while in this second stage, this is performed by CARLA. The communication between the different levels (strategy, tactical and operative) is carried out by ROS.

5.1.1. SUMO Simulator

SUMO is a simulation tool that allows the creation of driving scenarios and access to the environment information. It does not provide sensor implementation or vehicle dynamics, while it focuses on driving and decisions, which allows a really fast simulation. We have designed three intersections as described in Section 4.2. This simulator has been developed using the OpenAI Gym [39] library, which allows an easy way of creating custom environments for RL applications and the traci tool [40], which provides access to the simulation information.

The episode starts with the ego vehicle at the beginning of the road, and a traffic flow is generated in each lane. For each step of the simulation, a state vector is calculated, the RL agent executes an action and a reward is calculated for that action at that moment. The episode ends when the ego vehicle collides, reaches the end of the road or there is a timeout.

We propose a curriculum learning approach by adding complexity to the scenario. Training the RL algorithm under high-density traffic intersections does not converge to an optimal policy. The policy obtained avoids collisions but does not cross the intersection; thus, we propose a first training phase only with vehicles driving in one of the intersecting lanes and a second training phase with vehicles in both lanes. The algorithm is trained over 5M episodes for each phase.

5.1.2. CARLA Simulator

SUMO is an autonomous driving simulator that emulates real driving but does not simulate the dynamics of the vehicle. To add complexity to this study, we propose extrapolating the models obtained in this simulator to the realistic CARLA simulator, which is developed to design and validate autonomous driving systems.

One of the most challenging tasks in RL is domain adaptation. We maintain the essential inputs and outputs, so the content does not vary for the agent. The intersection scenarios of the two simulators are shown in Figure 7.

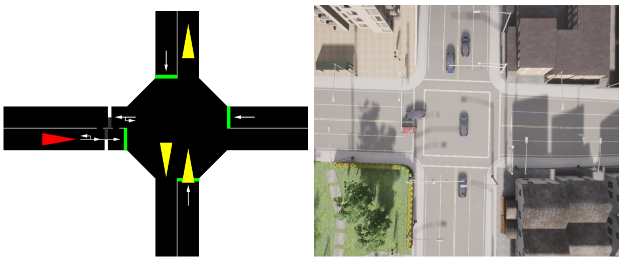
(**a**) (**b**)

Figure 7. CARLA and SUMO intersection scenarios. (**a**) Bird-eye-view of the intersection in SUMO Simulator. (**b**) Bird-eye-view of the intersection in CARLA Simulator.

In CARLA, three vehicles are spawned in each lane. These have different behaviors: they can yield, cross or stop randomly. They are commanded by the CARLA autopilot. As in SUMO, each episode ends when the collision sensor is activated, if the ego vehicle reaches the end of the road or if the duration of the episode exceeds the timeout value. During these episodes, when the tactical level requires an action, this is sent via ROS to the operative level.

5.2. Evaluation Metrics

To quantify the performance of our system, we evaluate the ego vehicle in terms of success and simulation time. These metrics are defined as:

- $success\ (\%) = end\ reached / n_e$
- $t_{avg} = \sum t_n\ /\ n_e$

where the number of episodes n_e is defined for each evaluation, the simulation time is measured in seconds and the commanded velocity is 5 m/s.

It should be pointed out that the road in the SUMO simulator is significantly longer than the road in the CARLA simulator. Because of this, the average time in SUMO is expected to be longer than in CARLA.

We define four scenarios for these experiments, one for each type of intersection defined in Section 4.2 and one combining all of them in a random way. The adversaries are generated in the same way that they were generated during the training. The previously trained algorithms are tested in this validation environment.

First, we focus on SUMO, running 100 test episodes in each scenario for each algorithm trained: one phase training PPO (1-PPO), two-phase training PPO (2-PPO), one-phase training PPO with features extractor (1-FEPPO) and two-phase training PPO with features extractor (2-FEPPO).

The model with better results in SUMO is used as prior data to conduct the second training stage in CARLA, using both the network and the weights. This prior model (SUMO) and the final model (CARLA) are compared in terms of success and simulation time in all four scenarios in CARLA. We run 100 test episodes for each scenario to obtain the results presented in the following section.

5.3. Results

In this section, we describe the results obtained during this research. We define four scenarios and expose the performance of the different algorithms in terms of success and average episode time. In addition, we compare our architecture with other similar architectures in the state of the art. The GPU used for all the experiments described below is the NVIDIA Titan with 12 GB of VRam.

In Table 2, we present the mean values along 100 test episodes in the SUMO simulator for the previously described algorithms and scenarios. The 2-FEPPO obtains the best results with no collisions in the traffic light scenario and a 95% success in the combined scenario.

Table 2. Evaluation metrics results in SUMO simulator: comparison between algorithms trained with and without features structure.

	Traffic Light		**Stop Signal**		**Uncontrolled**		**Combination**	
	success (%)	t_{avg} (s)	*success* (%)	t_{avg} (s)	*success* (%)	t_{avg} (s)	*success* (%)	t_{avg} (s)
1-PPO	53	112	37	93	23	109	30	105
2-PPO	95	67	78	78	87	63	88	71
1-FEPPO	61	104	48	82	30	111	37	102
2-FEPPO	100	43	90	94	95	55	95	85

In Table 3, we present the mean values along 100 test episodes in the CARLA simulator for the previously described algorithms and scenarios. The results obtained with the 2-FEPPO are worse in CARLA than in SUMO. After the second training stage, a 78% success rate in the combined test scenarios is obtained by the RL agent.

Table 3. Evaluation metrics results in CARLA simulator: comparison between the model trained in SUMO (2-FEPPO) and the model trained in CARLA (SUMO + CARLA).

	Traffic Light		**Stop Signal**		**Uncontrolled**		**Combination**	
	success (%)	t_{avg} (s)	*success* (%)	t_{avg} (s)	*success* (%)	t_{avg} (s)	*success* (%)	t_{avg} (s)
SUMO	78	17	35	19	47	19	50	19
CARLA	83	17	70	19	75	16	78	17

It is a difficult task to present a comparison with the state of the art, since different scenarios and problems are defined in different works. However, in Table 4, we present a comparison in terms of success rate at an uncontrolled intersection. This table shows the results of our approach and some other similar approaches that can be found in the literature. The obtained success rates indicate that our results are on par or even better than other competitive approaches.

Table 4. Comparison of success rate between different approaches.

Architecture	**Success Rate (%)**
2-FEPPO	95
MPC Agent [24]	95.2
Level-k Agent [28]	93.8
Sc04 Left Turn [30]	90.3

In addition, we study the influence of virtual sensors in the RL agent regarding an agent trained using only ground truth data. For this experiment, we have not added another training stage, so we can see the effects of the uncertainty produced by the sensor's detection. We present a first approach where the state vector is directly obtained from the SUMO simulator and a second more realistic approach, where this vector is defined using an estimation of the parameters. We have studied this estimation in a 3D object detection benchmark (KITTI dataset [41]) where ground truth is provided by a Velodyne laser scanner and GPS localization system. We have processed these sensor data using the PointPillar algorithm [42], and we have obtained a maximum distance error of ± 5 m (for distances between 0 and 100 m). Finally, we have modified our state vector by adding this error. In Table 5, we present the evaluation metrics of these two approaches. It can be seen that the results are better using ideal ground truth data obtained directly from the simulator (distance error equals 0). However, very good results are also obtained working with data simulating sensor errors (distance error equals 5%).

Table 5. Evaluation metrics results in SUMO simulator: ground truth vs. sensor data simulation.

	Traffic Light		**Stop Signal**		**Uncontrolled**		**Combination**	
	success (%)	t_{avg} (s)	*success* (%)	t_{avg} (s)	*success* (%)	t_{avg} (s)	*success* (%)	t_{avg} (s)
Ground Truth	100	43	90	94	95	55	95	85
Sensor Data	96	42	88	92	94	51	91	80

These results can be replicated through our repository found in the Supplementary Materials.

5.4. Discussion

RL is defined as a trial and error process, where the experience of modeling the environment and the reward is the key to success. Our first approach was training the agent from scratch using the CARLA simulator, but we discarded training in this way due to the large episodes required for convergence. Then, we used the SUMO simulator, where we were able to train the agent during 30 k episodes in a few hours. Using SUMO training as prior in CARLA training took 10.5 h. In Table 6, a comparison between the training time for each simulation is shown.

Table 6. Simulation time.

Simulator	**No. of Episodes**	**Time (h)**
SUMO	30 k	5
CARLA (estimated)	30 k	1650
SUMO + CARLA	30 k + 1 k	10.5

Training the ego vehicle in dense traffic intersections using the PPO algorithm did not converge into an optimal solution: the ego vehicle did not cross the intersection but avoided collisions. With this behavior, the mean value of the total rewards was lower than 1. We solved this by a curriculum learning approach adding vehicles in different phases. By this, we obtained an optimal policy: the ego vehicle was able to cross the intersection avoiding adversarial vehicles and reaching a mean value for the total rewards close to 1.2. This is the maximum value of the total reward and guarantees that the chosen policy is optimal. Then, we presented the features extractor architecture, which led us to obtain better results in performance and convergence. The two-phase trained algorithms obtained higher success percentages, the 2-FEPPO of them being the best. The one-phase trained algorithms were not trained in dense traffic scenarios, so a high number of failures was expected. Regarding the intersection scenarios, all vehicles obtained the best results in the traffic light scenario and the worst metrics in the stop scenario. This makes sense considering that adversaries do not yield when there is a stop sign for the ego vehicle, but they do yield when the traffic light is red for them. Considering that the success rate for the proposed architecture is quite good, we observed that the ego vehicle did not always follow traffic rules. If it finds a gap when the traffic light is red, it will pass. The ego vehicle drives only according to the adversary's behavior, so a simple higher layer would be necessary to satisfy the traffic rules.

Figure 8 shows a visualization of the RL agent at the traffic light intersection, with the time, the actions and the ego vehicle velocity data information. The ego vehicle stops when the adversary is getting close to the intersection; when this adversary goes through the intersection, the ego vehicle takes the action to drive and cross the intersection.

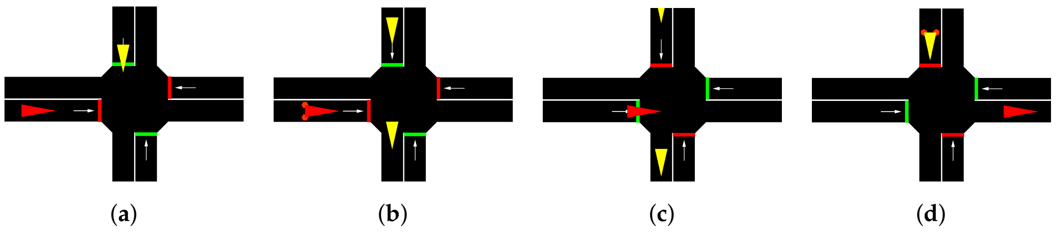

Figure 8. RL agent (red) at the intersection with the traffic light. Velocity (v), action (a) and time (t) are described for each frame. (**a**) v = 5; a = drive; t = 9.5 (**b**) v = 0; a = stop; t = 13.7 (**c**) v = 2; a = drive; t = 16.2 (**d**) v = 5; a = drive; t = 22.5.

In addition, we analyze the performance of adding a features extractor between the state vector and the PPO input. In Figure 9, we show the mean reward for the episodes along the training phase. It is clear that training with the features extractor layer converges faster and even the mean reward is higher. It should be noted that the second phase of the FEPPO converges in around 1 M episodes.

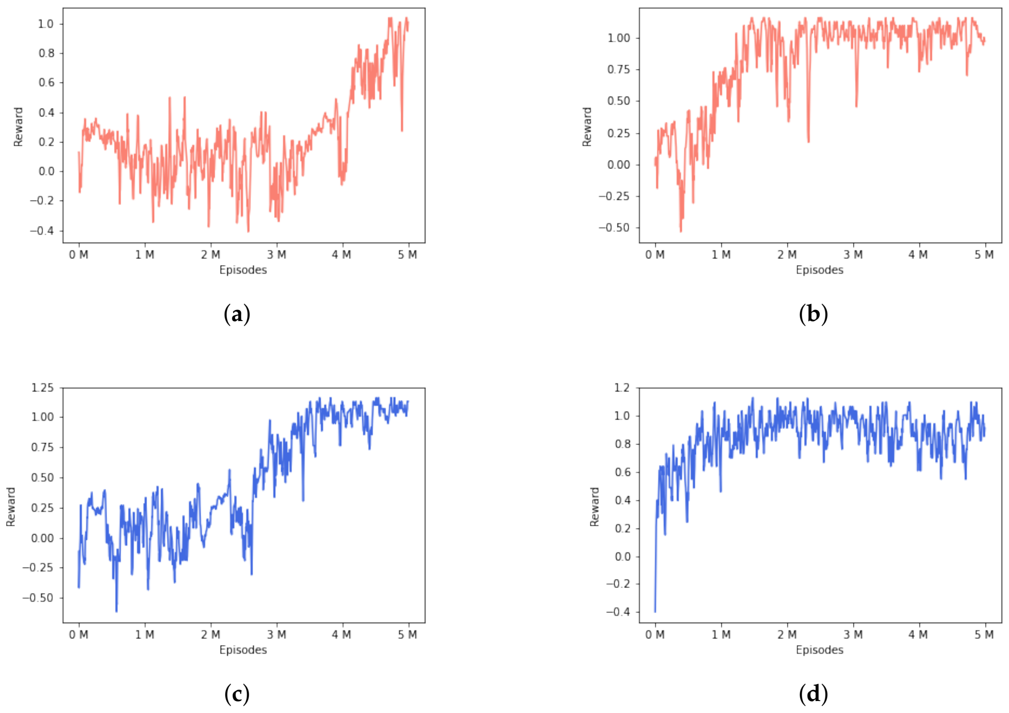

Figure 9. Episodes mean rewards. Comparison between the two approaches described in the paper: with (**right**) and without (**left**) features extractor module. The red charts correspond to the first training phase and the blue charts corresponds to the second training phase. (**a**) One-phase PPO episodes mean reward with adversaries in one lane. No features extractor layer. (**b**) One-phase FEPPO episodes mean reward with adversaries in one lane. Features extractor layer. (**c**) Two-phase PPO episodes mean reward with adversaries in two lanes. No features extractor layer. (**d**) Two-phase FEPPO episodes mean reward with adversaries in two lanes. Features extractor layer.

The transfer of knowledge is a difficult task, since the two environments have some differences. Our priority is to set the state vector and the transition function of both simulators as similar as possible, but they are never the same. As mentioned in previous sections, a second training stage is performed using the 2-FEPPO model weights.

The resulting algorithm of this second stage does not obtain the best policy in the CARLA simulator. The main difference between the two simulators is in the vehicle's dynamics. When an action is taken in SUMO, this action happens immediately, but in CARLA, a response time is added. Adding dynamics to the vehicles' movements adds complexity to the intersection problem. The resulting behavior is that the ego vehicle crosses the intersection consistently, but sometimes it collides with other vehicles.

A visualization of the RL agent at an uncontrolled intersection is shown in Figure 10 along with the time, the actions and the ego vehicle velocity data information. The ego vehicle stops when the adversary is getting close to the intersection, and as it goes away, the ego vehicle drives and crosses the intersection.

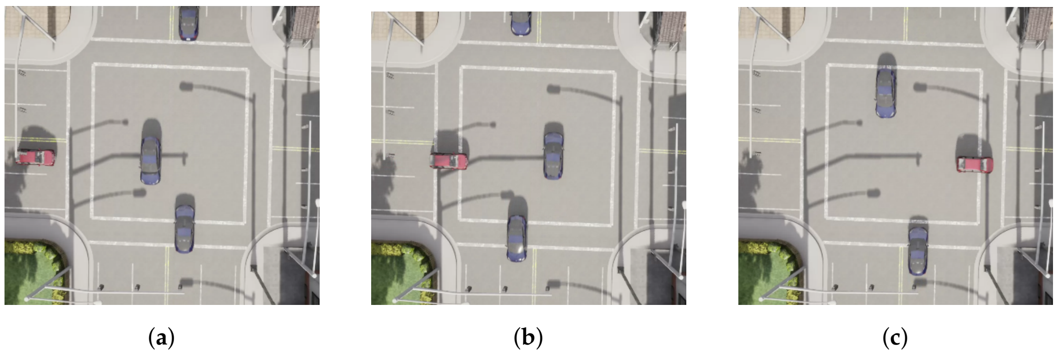

Figure 10. RL agent (red) at uncontrolled intersection. Velocity (v), action (a) and time (t) are described for each frame. (**a**) v = 3; a = stop; t = 8.5 (**b**) v = 4; a = drive; t = 12.7 (**c**) v = 5; a = drive; t = 17.2.

6. Conclusions and Future Works

The results of this paper state that a Proximal Policy Optimization algorithm can execute decisions at intersection scenarios.

The algorithm learns the policy with different types of intersections with no prior information about the type of scenario but only the relative distance and velocity of the surrounding vehicles. The ego vehicle is able to drive in complex scenarios, but for inferring traffic rules from other participants, it would be necessary to have a reward focused on this feature.

Furthermore, the application of curriculum learning techniques for the RL agent training allows improvements in the metrics obtained and faster convergence. The implementation of a features extractor module not only improves the training phase but obtains better metrics during the evaluation tests.

In the near future, we plan to extend this research by obtaining the state vector from real sensor data instead of obtaining them directly from the ground truth. The main advantage of the proposed architecture is that it is ROS based, which allows a fast and easy integration in our real vehicle [43]. We have already conducted some experiments in a real environment on our university campus, using a Petri nets-based decision-making module. Our future goal is to replace this module with the system described in this work and compare the results.

Supplementary Materials: The GitHub repositories with the implemented code are available online at https://github.com/RobeSafe-UAH/rl-intersections.

Author Contributions: Conceptualization and methodology, R.G.-M., R.B., E.L.-G. and L.M.B.; software, R.G.-M. and J.A.; validation and formal analysis, R.G.-M., R.B., E.L.-G. and L.M.B.; investigation, R.G.-M., R.B., E.L.-G. and L.M.B.; resources, R.G.-M., R.B., E.L.-G. and L.M.B.; writing—original draft preparation, R.G.-M.; writing—review and editing, all authors; supervision, E.L.-G., R.B. and L.M.B.; project administration and funding acquisition, L.M.B. and R.B. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been funded in part from the Spanish MICINN/FEDER through the Artificial Intelligence based modular Architecture Implementation and Validation for Autonomous Driving (AIVATAR) project (PID2021-126623OB-I00) and from the RoboCity2030-DIH-CM project (P2018/NMT- 4331), funded by Programas de actividades I+D (CAM), cofunded by EU Structural Funds and Scholarship for Introduction to Research activity by University of Alcalá.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Werneke, J.; Vollrath, M. How do environmental characteristics at intersections change in their relevance for drivers before entering an intersection: Analysis of drivers' gaze and driving behavior in a driving simulator study. *Cogn. Technol.* **2014**, *16*, 157–169. [CrossRef]
2. NHTSA. *Traffic Safety Facts 2019*; National Highway Traffic Safety Administration: Washington, DC, USA, 2019.
3. Li, G.; Li, S.; Li, S.; Qin, Y.; Cao, D.; Qu, X.; Cheng, B. Deep Reinforcement Learning Enabled Decision-Making for Autonomous Driving at Intersections. *Automot. Innov.* **2020**, *3*, 374–385. [CrossRef]
4. Qiao, Z.; Muelling, K.; Dolan, J.M.; Palanisamy, P.; Mudalige, P. Automatically Generated Curriculum based Reinforcement Learning for Autonomous Vehicles in Urban Environment. In Proceedings of the 2018 IEEE Intelligent Vehicles Symposium (IV), Changshu, China, 26–30 June 2018; pp. 1233–1238. [CrossRef]
5. Aoki, S.; Rajkumar, R. V2V-based Synchronous Intersection Protocols for Mixed Traffic of Human-Driven and Self-Driving Vehicles. In Proceedings of the 2019 IEEE 25th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Hangzhou, China, 18–21 August 2019; pp. 1–11. [CrossRef]
6. Duan, X.; Jiang, H.; Tian, D.; Zou, T.; Zhou, J.; Cao, Y. V2I based environment perception for autonomous vehicles at intersections. *China Commun.* **2021**, *18*, 1–12. [CrossRef]
7. Isele, D.; Rahimi, R.; Cosgun, A.; Subramanian, K.; Fujimura, K. Navigating Occluded Intersections with Autonomous Vehicles Using Deep Reinforcement Learning. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018; pp. 2034–2039. [CrossRef]
8. Cosgun, A.; Ma, L.; Chiu, J.; Huang, J.; Demir, M.; Anon, A.; Lian, T.; Tafish, H., Al-Stouhi, S. Towards full automated drive in urban environments: A demonstration in GoMentum Station, California. In Proceedings of the 2017 IEEE Intelligent Vehicles Symposium (IV 2017), Los Angeles, CA, USA, 11–14 June 2017; Zhang, W.B., de La Fortelle, A., Acarman, T.; Yang, M., Eds.; IEEE, Institute of Electrical and Electronics Engineers: Manhattan, NY, USA, 2017; pp. 1811–1818. [CrossRef]
9. Xu, H.; Gao, Y.; Yu, F.; Darrell, T. End-to-end Learning of Driving Models from Large-scale Video Datasets. *arXiv* **2016**, arXiv:1612.01079.
10. Kendall, A.; Hawke, J.; Janz, D.; Mazur, P.; Reda, D.; Allen, J.; Lam, V.; Bewley, A.; Shah, A. Learning to Drive in a Day. *arXiv* **2018**, arXiv:1807.00412.
11. Anzalone, L.; Barra, S.; Nappi, M. Reinforced Curriculum Learning For Autonomous Driving In Carla. In Proceedings of the 2021 IEEE International Conference on Image Processing (ICIP), Anchorage, AK, USA, 19–22 September 2021; pp. 3318–3322. [CrossRef]
12. Behrisch, M.; Bieker, L.; Erdmann, J.; Krajzewicz, D. SUMO—Simulation of Urban MObility: An overview. In Proceedings of the SIMUL 2011, Third International Conference on Advances in System Simulation, Barcelona, Spain, 23–28 October 2011; pp. 63–68.
13. Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; Koltun, V. CARLA: An Open Urban Driving Simulator. In Proceedings of the 1st Annual Conference on Robot Learning, Mountain View, CA, USA, 13–15 November 2017; Volume 78, pp. 1–16.
14. Wang, P.; Li, H.; Chan, C. Continuous Control for Automated Lane Change Behavior Based on Deep Deterministic Policy Gradient Algorithm. *arXiv* **2019**, arXiv:1906.02275.
15. Paden, B.; Cáp, M.; Yong, S.Z.; Yershov, D.S.; Frazzoli, E. A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles. *arXiv* **2016**, arXiv:1604.07446.
16. González, D.; Pérez, J.; Milanés, V.; Nashashibi, F. A Review of Motion Planning Techniques for Automated Vehicles. *IEEE Trans. Intell. Transp. Syst.* **2016**, *17*, 1135–1145. [CrossRef]
17. Mirchevska, B.; Pek, C.; Werling, M.; Althoff, M.; Boedecker, J. High-level Decision Making for Safe and Reasonable Autonomous Lane Changing using Reinforcement Learning. In Proceedings of the 2018 21st International Conference on Intelligent Transportation Systems (ITSC), Maui, HI, USA, 4–7 November 2018; pp. 2156–2162. [CrossRef]
18. Ye, F.; Zhang, S.; Wang, P.; Chan, C. A Survey of Deep Reinforcement Learning Algorithms for Motion Planning and Control of Autonomous Vehicles. *arXiv* **2021**, arXiv:2105.14218.

19. Bojarski, M.; Testa, D.D.; Dworakowski, D.; Firner, B.; Flepp, B.; Goyal, P.; Jackel, L.D.; Monfort, M.; Muller, U.; Zhang, J.; et al. End to End Learning for Self-Driving Cars. *arXiv* **2016**, arXiv:1604.07316.
20. Moghadam, M.; Alizadeh, A.; Tekin, E.; Elkaim, G.H. An End-to-end Deep Reinforcement Learning Approach for the Long-term Short-term Planning on the Frenet Space. *arXiv* **2020**, arXiv:2011.13098.
21. Chopra, R.; Roy, S. End-to-End Reinforcement Learning for Self-driving Car. In *Advanced Computing and Intelligent Engineering*; Springer: Singapore, 2020; pp. 53–61. [CrossRef]
22. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
23. Tram, T.; Jansson, A.; Grönberg, R.; Ali, M.; Sjöberg, J. Learning Negotiating Behavior Between Cars in Intersections using Deep Q-Learning. In Proceedings of the 2018 21st International Conference on Intelligent Transportation Systems (ITSC), Maui, HI, USA, 4–7 November 2018; pp. 3169–3174. [CrossRef]
24. Tram, T.; Batkovic, I.; Ali, M.; Sjöberg, J. Learning When to Drive in Intersections by Combining Reinforcement Learning and Model Predictive Control. In Proceedings of the 2019 IEEE Intelligent Transportation Systems Conference (ITSC), Auckland, New Zealand, 27–30 October 2019; pp. 3263–3268. [CrossRef]
25. Kamran, D.; Lopez, C.F.; Lauer, M.; Stiller, C. Risk-Aware High-level Decisions for Automated Driving at Occluded Intersections with Reinforcement Learning. *arXiv* **2020**, arXiv:2004.04450.
26. Bouton, M.; Nakhaei, A.; Fujimura, K.; Kochenderfer, M.J. Safe Reinforcement Learning with Scene Decomposition for Navigating Complex Urban Environments. In Proceedings of the 2019 IEEE Intelligent Vehicles Symposium (IV), Paris, France, 9–12 June 2019; pp. 1469–1476. [CrossRef]
27. Bouton, M.; Cosgun, A.; Kochenderfer, M.J. Belief state planning for autonomously navigating urban intersections. In Proceedings of the 2017 IEEE Intelligent Vehicles Symposium (IV), Los Angeles, CA, USA, 11–14 June 2017; pp. 825–830. [CrossRef]
28. Bouton, M.; Nakhaei, A.; Fujimura, K.; Kochenderfer, M.J. Cooperation-Aware Reinforcement Learning for Merging in Dense Traffic. In Proceedings of the 2019 IEEE Intelligent Transportation Systems Conference (ITSC), Auckland, New Zealand, 27–30 October 2019; pp. 3441–3447. [CrossRef]
29. Shu, K.; Yu, H.; Chen, X.; Chen, L.; Wang, Q.; Li, L.; Cao, D. Autonomous Driving at Intersections: A Critical-Turning-Point Approach for Left Turns. *arXiv* **2020**, arXiv:2003.02409.
30. Kurzer, K.; Schörner, P.; Albers, A.; Thomsen, H.; Daaboul, K.; Zöllner, J.M. Generalizing Decision Making for Automated Driving with an Invariant Environment Representation using Deep Reinforcement Learning. *arXiv* **2021**, arXiv:2102.06765.
31. Soviany, P.; Ionescu, R.T.; Rota, P.; Sebe, N. Curriculum Learning: A Survey. *arXiv* **2021**, arXiv:2101.10382.
32. Wang, X.; Chen, Y.; Zhu, W. A Survey on Curriculum Learning. *IEEE Trans. Pattern Anal. Mach. Intell.* **2021**, *44*, 4555–4576. [CrossRef]
33. Diaz-Diaz, A.; Ocaña, M.; Llamazares, A.; Gómez-Huélamo, C.; Revenga, P.; Bergasa, L.M. HD maps: Exploiting OpenDRIVE potential for Path Planning and Map Monitoring. In Proceedings of the 2022 IEEE Intelligent Vehicles Symposium (IV), Aachen, Germany, 5–9 June 2022.
34. Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A.Y. ROS: An open-source Robot Operating System. In Proceedings of the ICRA Workshop on Open Source Software, Kobe, Japan, 12–17 May 2009; p. 5.
35. Gutiérrez, R.; López-Guillén, E.; Bergasa, L.M.; Barea, R.; Pérez, Ó.; Gómez Huélamo, C.; Arango, J.F.; del Egido, J.; López, J. A Waypoint Tracking Controller for Autonomous Road Vehicles Using ROS Framework. *Sensors* **2020**, *20*, 4062. [CrossRef]
36. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347.
37. van Hasselt, H.; Guez, A.; Hessel, M.; Mnih, V.; Silver, D. Learning values across many orders of magnitude. *arXiv* **2016**, [CrossRef]
38. Treiber, M.; Hennecke, A.; Helbing, D. Congested traffic states in empirical observations and microscopic simulations. *Phys. Rev. E* **2000**, *62*, 1805–1824. [CrossRef] [PubMed]
39. Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; Zaremba, W. Openai gym. *arXiv* **2016**, arXiv:1606.01540.
40. Wegener, A.; Piórkowski, M.; Raya, M.; Hellbrück, H.; Fischer, S.; Hubaux, J.P. TraCI: An Interface for Coupling Road Traffic and Network Simulators. In Proceedings of the 11th Communications and Networking Simulation Symposium, Ottawa, ON, Canada, 14–17 April 2008; Association for Computing Machinery: New York, NY, USA, 2008; pp. 155–163. [CrossRef]
41. Geiger, A.; Lenz, P.; Stiller, C.; Urtasun, R. Vision meets Robotics: The KITTI Dataset. *Int. J. Robot. Res.* **2013**, *32*, 1231–1237. [CrossRef]
42. Lang, A.H.; Vora, S.; Caesar, H.; Zhou, L.; Yang, J.; Beijbom, O. PointPillars: Fast Encoders for Object Detection from Point Clouds. *arXiv* **2018**, arXiv:1812.05784.
43. Arango, J.F.; Bergasa, L.M.; Revenga, P.; Barea, R.; López-Guillén, E.; Gómez-Huélamo, C.; Araluce, J.; Gutiérrez, R. Drive-By-Wire Development Process Based on ROS for an Autonomous Electric Vehicle. *Sensors* **2020**, *20*, 6121. [CrossRef]