Learning from Oracle Demonstrations – A new approach to develop Autonomous Intersection Management control algorithms based on Multi-Agent Deep Reinforcement Learning

Antonio Guillen-Perez¹ and Maria-Dolores Cano¹, Senior Member, IEEE
¹Department of Information and Communication Technologies, Universidad Politécnica de Cartagena, 30202 Cartagena, Spain

ABSTRACT Worldwide, many companies are working towards safe and innovative control systems for Autonomous Vehicles (AVs). A key component is Autonomous Intersection Management (AIM) systems, which operate at the level of traffic intersections and manage the right-of-way of AVs, improving flow and safety. AIM traditionally uses control policies based on simple rules. However, Deep Reinforcement Learning (DRL) can provide advanced control policies, with the advantage of reacting proactively and forecasting hazardous situations. The main drawback of DRL is training time, which is fast in simple tasks, but not negligible when we address real-world problems with multiple agents. Learning from Demonstrations (LfD) emerges to solve this problem, speeding up training significantly, and reducing the exploration problem. The challenge is that LfD requires an expert to extract new demonstrations. Therefore, in this paper, we propose to use an agent, previously trained by imitation learning, to act as an expert to leverage LfD. We name this new agent Oracle, and our new approach is called Learning from Oracle Demonstrations (LfOD). We have implemented this novel method over the DRL TD3 algorithm, incorporating significant changes to TD3 that allow the use of Oracle demonstrations. The complete version is called TD3fOD. The results obtained in the AIM training scenario show that TD3fOD notably improves the learning process compared with TD3, and DDPGfD, speeding up learning to 5–6 times, while the policy found offers both significantly lower variance and better learning ability. The testing scenario also shows a relevant improvement in multiple key performance metrics compared to other vehicle control techniques on AIM, such as reducing waiting time by more than 90% and significantly decreasing fuel or electricity consumption and emissions, highlighting the benefits of LfOD.

INDEX TERMS Autonomous Intersection Management, Intelligent Transport Systems, Intersection Traffic Management, Learning from Demonstrations, Multi-Agent Deep Reinforcement Learning

I. INTRODUCTION

Deep Reinforcement Learning (DRL) has demonstrated a remarkable ability to master several complex real-world sequential decision-making problems [1]–[6]. However, this success is currently limited due to the extensive training process required by traditional DRL algorithms, which need days to months of training and tens or hundreds of graphics cards in parallel [7], [8].

A field of work where DRL has been extensively studied is the control of Autonomous Vehicles (AVs). This field achieves a perfect symbiosis combined with computer simulators, making them an excellent framework for new advanced control systems. The management of these AVs in cities should be done collectively with centralized information to be as efficient as possible. This centralized control would make it possible to obtain an intelligent system capable of controlling all AVs simultaneously, in real-time, and guaranteeing a high degree of security.

The study of Autonomous Intersection Management (AIM) [9], [10] began a few years ago, even before the large-scale
deployment of the first AVs. However, these AIM based their operations on simple rules, unable to achieve advanced control policies that could obtain truly intelligent and proactive behavior. That was the scenario until the development of RAIM [11], an AIM that bases its operation on DRL along with other advanced techniques of deep learning. However, the main problem with this approach is that it needs a lot of interactions with the environment to achieve good performance.

Recently, researchers are studying new alternatives that allow to speed up the training of the DRL algorithms, either by imitating a behavior (Imitation Learning, IL [12]), imitating some observations (Imitation from Observation, IFO [13]), or using an initial phase of supervised learning of demonstrations (tuples of {state, action, new_state, reward}) offered by an expert to progressively train the pre-trained policy using DRL (Learning from Demonstrations, LfD [14]). Several previous efforts have demonstrated that LfD can significantly accelerate the training of DRL algorithms in environments where there are expert demonstrations [14]–[16]. However, although we consider that LfD has potential for improvement, there are environments where the demonstrations offered by an expert cannot be extracted (totally or partially), such as in traffic simulators used to train new AV control systems. (e.g., SUMO [17]). In these simulators, each vehicle has its own internally modeled controller and it is not possible to extract (in a straightforward way) demonstrations from each vehicle.

Due to the above limitations, this work proposes a new approach within the field of LfD that uses an IL-trained agent that will model the hidden expert controllers. This agent allows us to extract the knowledge of a hidden expert (new experiences) and ask about each state, what action the hidden expert would have taken. In other words, the IL-trained agent can ask for each state: “What would my expert say I should do in this state?” In this way, in environments where there is no expert (or one is hidden) from which to extract the demonstrations, we can train an agent that imitates the hidden expert behavior and can be considered as such to leverage LfD to further train another agent via DRL. We call this new agent an Oracle. To enable the use of demonstrations offered by Oracle, we proposed several modifications to the DRL algorithm used, TD3: (i) A modification to the error equation that updates the TD3 actor so that the error produced by the RL action is gradually taken into account; (ii) the introduction of several parameters for a smooth and progressive transition between LfD and RL (τ₁ and τ₂); and (iii) the use of two replay buffers, one for demonstrations to train Oracle and the other for TD3, in addition to the use of Prioritized Experience Replay (PER) to accelerate learning. Following the nomenclature used in previous works, we have called this new approach Learning from Oracle Demonstrations (LfOD), and the subsequent DRL algorithm is called TD3fOD. TD3fOD is used to train an AIM algorithm previously proposed in an autonomous vehicle traffic scenario [11]. The results show a notable improvement over not using LfD, speeding up by 5–6 times and reducing the variance noticeably in the policy obtained. The results compared with DDPGfD show that the use of Oracle allows to triple the training speed and considerably reduce the variance in the control behavior during training the policies. Finally, in a test scenario, the AIM algorithm trained with TD3fOD is compared with other autonomous vehicle control algorithms, and the results show an improvement in all the studied metrics, reducing the waiting time by about 95%, among other factors.

Thanks to our proposal, it is possible to extract the hidden agent (learned by imitation in Oracle) in those simulators where it is not possible (or too complicated), in order to take advantage of the benefits offered by LfD (training acceleration and more robust policies) for the development of new complex control algorithms.

The primary motivation of this work lies in the development of a new algorithm to accelerate agent training using DRL and LfD, which can be applied to any problem where there is no agent from which to extract demonstrations. Furthermore, the development of advanced cooperative control systems for AVs and Multi-Agent DRL-based systems could be speeded up from the contributions of this work.

Therefore, the main contributions of this work are:

(i) propose a new LfD approach that can be used in environments where there are no experts from which to extract demonstrations, taking advantage of hidden agent demonstrations;

(ii) demonstrate that using an Oracle in LfOD speeds up the training of DRL algorithms, reducing training time and variance in trained policies; and

(iii) development of a new AIM trained with the proposed algorithm (RAIM with LfOD), capable of improving the performance of AIM algorithms in a cooperative autonomous vehicle control scenario.

The rest of the article is organized as follows. Section II provides the background of DRL, IL, and LfD. A review of previous related work on both AIM and LfD is discussed in Section III. Section IV details our proposal, TD3fOD. The experimental setup, simulation scenarios, and the explanation of the modifications made to TD3 and AIM are included in Section V. Section VI shows the results obtained in both the training and the testing scenarios. Finally, Section VII concludes the paper.

II. BACKGROUND

This section explains DRL, Multi-Agent DRL (MADRL), and the algorithm modified in this work, TD3. In addition, we detail how IL works, and, finally, we explain the basics of LfD.

A. DEEP REINFORCEMENT LEARNING

Reinforcement Learning (RL) is an area of machine learning in which an agent learns to complete a task in an environment where it can take an action and receive a reward for the action. The agent’s goal is to find a policy that performs actions that
maximize the rewards accumulated during the entire task, which are known as expected discounted total rewards.

The environment where the agent is located is usually modeled by a Markov Decision Process (MDP) because many RL algorithms employ dynamic programming techniques to solve these MDPs. An MDP is defined by the tuple $(S, A, R, T, γ)$, where $S$ represents a set of states of an environment, $A$ represents the set of actions that the agent can take, $T$ is the transition function $T: S \times A \times S \rightarrow [0,1]$ that determines the transition probability from any state $s \in S$ to any state $s' \in S$ when the action $a \in A$ is taken. $R$ is the reward function $R: S \times A \times S \rightarrow \mathbb{R}$ and $γ \in [0, 1]$ represents the discount factor that adjusts the trade-off between immediate and future rewards.

Resolving an MDP generates a policy $π: S \rightarrow A$, which will map the states $s \in S$ to the actions $a \in A$. An optimal policy $π^*$ maximizes the expected discounted total reward for all states. This approach to finding the optimal policy can be formulated by the state-action value function ($Q$-function): $Q^π(s, a) = E^π[\sum_{t=0}^{\infty} γ^t R(s_t, a_t)]$. This $Q$-function determines the expected reward by starting from the state $s$, taking the action $a$, and following the policy $π$.

If we focus on DRL, the introduction of Neural Networks (NN) in traditional RL algorithms has been a great revolution, considerably speeding up the learning of these algorithms and allowing them to be applied to tasks that seemed impossible before, because NNs can act as approximating functions of the policy to be learned.

Within DRL, there are several approaches, but the one that has attracted the most attention in recent years is based on actor-critics for continuous control problems. Both the actors (a policy that decides what action to take for each state) and the critic (given a state and an action, it gets what expected reward, or $Q$-value, is obtained, indicating to the actor whether the action is going to be good or not) are modeled by NNs.

B. MULTI-AGENT DEEP REINFORCEMENT LEARNING

MADRL [18] is a subset of RL problems in which multiple agents interact with each other and their environment, each of which attempts to learn a policy and learns jointly.

Within MADRL, there are two different learning approaches depending on the problem to be solved: 

- cooperative multi-agent learning [19], in which agents cooperate to maximize the total cumulative reward; and
- competitive multi-agent learning [20], in which agents compete with each other to obtain the highest possible reward individually (or from the group they belong to).

For the training and execution of these algorithms, different techniques have been developed that take advantage of the benefits offered by collective learning. One of them is centralized training and decentralized execution [21]. Here, training is performed centrally, in an environment where each agent sends its information, and the control policies of each agent are obtained. Then, each agent obtains the policy and executes it decentralized. Another approach is centralized training and centralized execution [22]. In this case, agents are trained centrally and then executed individually. Last, we have decentralized training and decentralized execution. In this case, agents are trained in a decentralized manner and execute policies individually. The benefits of centralized training are better knowledge of the entire environment, so the policy is found more quickly and robustly. However, it is not always possible to centralize knowledge for training due to communication limitations, so decentralized training is needed. In decentralized training, each agent only has local knowledge for training. Thus, each agent updates its policy individually while sharing its policy with other agents. Finally, decentralized execution means that agents are executed on a decentralized controller individually while considering all agents simultaneously.

C. TD3

Our method combines demonstrations extracted from an Oracle to exploit this knowledge and accelerate the training of a new control policy by means of the DRL TD3 algorithm [23]. TD3 is a DRL algorithm whose acronym stands for Twin Delayed Deep Deterministic Policy Gradient. By doing so, we can obtain a new way to develop AIM systems in a fast and straightforward way.

TD3 is one of the most powerful and advanced off-policy model-free RL algorithms for continuous tasks. TD3 is an evolution of the Deep Deterministic Policy Gradient (DDPG [24]) DRL algorithm where several key improvements are added, specifically, Clipped Double-Q Learning, Target Policy Smoothing, and “Delayed” Policy Updates [23], [25]. As many articles have shown, TD3 offers fast convergence for complex tasks requiring continuous control [23], [26]. For this reason, we decided to implement our proposal on TD3 and not on other algorithms such as SAC [27], PPO [28], or A3C [29].

Clipped Double-Q Learning: Instead of using only one critic network, TD3 adds a second critic network that reduces the estimation bias by selecting the smallest $Q$-value of the two critic networks, encouraging underestimation of $Q$-values. This underestimation bias is not a problem since low values will not propagate through the algorithm, unlike overestimated values. Thus, it provides a more stable approximation and improves the stability of the whole algorithm.

Target Policy Smoothing: To reduce the overfitting produced by high variance target values when updating the critics, TD3 adds a small noise to each selected action. In addition, it performs a double clipping, first to the aggregated noise and then to the noisy action. This feature reduces the variance of the selected actions and results in more stable $Q$-values.

Delayed Policy Updated: TD3 updates the policy and target networks less frequently than the $Q$-functions, providing more stable and efficient training. The original paper [23] suggests updating the policy and target networks every two updates of the $Q$-functions. In addition, the policy network $π_θ$ is updated with a gradient ascent step simply by maximizing $Q_{θ_1}$ as shown in (1).
\[ \nabla_{\theta} \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \pi_{\theta}(s)) \]  

(1)

**D. IMITATION LEARNING**

IL arises from the need to train an agent more efficiently than through RL alone. With IL, an agent learns a policy by “imitating” an expert who knows what action to take for each state. Through Supervised Learning (SL), IL is more effective than RL when an expert demonstrates the desired behavior and thus teaches a policy [30]. Besides, the problem of reward shaping is eliminated, where it was necessary to carefully select the reward received by the agent or design a hand-coded function that changes smoothly to achieve a stable and consistent policy. Within IL, there are several alternatives:

1) **BEHAVIORAL CLONING**

Behavioral Cloning (BC) [12], [31], [32] is the simplest form of IL, where a policy is trained based on expert demonstrations; that is, there is an expert agent who is capable of producing pairs of state-action demonstrations. These demonstrations are used in traditional SL and can obtain a policy that achieves a behavior that clones the expert.

BC can work excellently for some applications where the entire state-action space is explored. However, in most cases, BC can be problematic. The main concern is that SL assumes that the samples used are i.i.d. (independent and identically distributed), but that assumption cannot be guaranteed due to the nature of the samples' capturing process.

Further, when the trained agent takes control, it can make mistakes in predicting actions that can lead to states never seen before during the expert-supervised training. In these states, the agent's behavior can lead to hazardous situations known as compounding errors [12], from which the agent can never recover. An example of compounding errors is depicted in Fig. 1. Several alternatives have been proposed to obtain more samples online while testing the trained policies and resolving the problems presented by BC.

2) **DIRECT POLICY LEARNING VIA INTERACTIVE DEMONSTRATOR**

Direct Policy Learning (DPL) is an enhanced version of BC. It is an iterative process where expert feedback is collected during the training loop. The whole process starts by collecting demonstrations from the expert, which serve to train the agent. After the first training, the trained policy is rolled out, and the new states that are visited are stored. Then, the expert is asked what actions it would take in those new states, collecting new demonstrations. These new demonstrations (feedback) allow obtaining more data to train the agent using again SL. This loop continues until converging (see Fig. 2). It is important to store and use all the collected demonstrations to remember the mistakes the agent made in the past.

Within this group of IL, there are several powerful algorithms, most notably: SEARN (Search-based Structured Prediction) [33], SMILe (Stochastic Mixing Iterative Learning) [30], and DAgger (Data AGGregation) [12]. However, all these methods present a big drawback. They only use SL to obtain behaviors similar to those of the expert and do not employ any RL technique to obtain superior behavior. In addition, they need an online expert from which to obtain feedback about the states visited during rollout.

**FIGURE 1.** Compounding errors example.

**FIGURE 2.** The general Direct Policy Learning (DPL) algorithm.

**E. LEARNING FROM DEMONSTRATION**

Learning from demonstrations (LfD) was introduced to overcome the limitations of DPL. LfD appeared in DeepMind's Deep Q-Learning from Demonstrations (DQfD) work [14], elegantly unifying IL and RL.

LfD employs an expert's experiences (demonstrations, which can be potentially suboptimal) to pre-train an agent through SL, and then uses RL algorithms to improve the learned policy. However, using SL in the demonstration data and applying RL in the pre-trained policy is not ideal. What LfD does is employ the demonstration data throughout the training process. Thus, the agent finds a policy that eventually surpasses the expert. In summary, LfD allows to initialize (pre-train) an agent through expert demonstrations and then use RL to discover a better policy by interacting with the environment and using the previous expert demonstrations to not forget the base policy. What differentiates IL from LfD is that the latter only has expert demonstrations, a sequence of \((s_t, a_t, r_t, s_{t+1})\) and not an online expert to obtain feedback, as well as the training stage via RL to improve the learned policy.

LfD was presented using Deep Q-Network (DQN) algorithm. DQN is used to control agents in discrete space actions, and the transition from a discrete space action to a continuous one is not trivial. A modification of DDPG that
allowed the use of demonstrations for its training was introduced in [15], namely, DDPGfD.

DDPGfD stores both the demonstrations collected from the expert and the experiences collected by the agent in a replay buffer. In addition, it proposes a set of improvements such as the use of a mix of 1-step and n-step return losses, learning multiple times per environment step, and L2 regularization losses. However, DQfD and DDPGfD have the main drawback of misadjusting the internal parameters of the agent obtained during pre-training when the agent begins to take control, which can lead to forgetting everything that was pre-learned.

III. RELATED WORKS

This section summarizes works from the related literature that address AIM and LiD.

A. AUTONOMOUS INTERSECTION MANAGEMENT

AIM has emerged as an alternative control for AVs at traffic-light-regulated intersections. In [34], Dresner et al. proposed the first AIM that regulated the crossing of AVs at intersections using a reservation-based method following a "First Come, First Served" (FCFS) policy and eliminating traffic lights. This policy worked as follows: When a vehicle approached the intersection, it requested to reserve the space-time the vehicle needed to cross the intersection. If the reservation did not conflict with another vehicle's reservation, the intersection accepted it, and the vehicle followed the route it had requested. Otherwise, the vehicle received a reservation denial and slowed down to request another reservation later in searching for available space-time slots.

The first results obtained showed that FCFS could outperform traffic light control in terms of flow and delay. Later works [35], [36] proposed alternative control protocols that included non-autonomous vehicles (FCFS-LIGHT) and emergency vehicles (FCFS-EMERG). The authors also proposed a mechanism to switch among policies (FCFS, FCFS-LIGHT, and FCFS-EMERG) depending on intersection conditions, improving performance by using the policy that best suited each situation [37]. Their results outperformed traditional traffic light control.

A more detailed study on the FCFS protocol was proposed in [38], where it was tested against an optimized traditional traffic light signal. The results showed that FCFS reduced the delay against the traditional traffic light signal by more than 90%. An improvement of FCFS also based on reservations was proposed by Huang et al. [39]. They suggested that when the intersection sent the denial of a reservation, it also sent a recommended deceleration speed to reach the stop line when the vehicle stopped. Furthermore, this algorithm separated the vehicles into 3 groups according to their current and past status. The proposed algorithm was compared to a roundabout and a traffic light but not the original FCFS. The results showed a reduction in delay of 85% and a reduction in fuel consumption of 50%.

Because FCFS does not consider any mechanism for grouping (batching) requests that have the same direction, several enhancements were proposed [37], [40] where batching of requests was used to improve the flow of the intersection, either by having more requests to make smarter decisions or by allowing vehicles in the same flow to pass in batches. The results showed an improvement in both FCFS and traffic light control, doubling the flow and reducing the delay by 85%.

Other approaches to AVs control use mathematical optimization to obtain right-of-way [41]–[45]. The results achieved by these proposals were similar to the previously shown algorithms. However, due to the solving characteristics of these algorithms, the resolution complexity increases significantly when the number of vehicles increases. Consequently, these algorithms face a sizeable computational complexity problem, making them unfeasible for the real-time control AIM requires. Within this approach, we can find the work developed by Wu et al. in [46]. In this work, the authors allowed all movements in all lanes and developed two modules, one in charge of deciding the temporal instant at which the vehicle should enter the intersection and another one in charge of deciding in which entry lane the vehicle should be placed and which exit lane it should take. This work followed a Mixed-Integer Linear Programming (MILP) problem approach to solve the proposed set of equations and constraints. The results showed the goodness offered by AIMs; however, due to the approach followed, there are many open problems. A heuristic approach is followed in the paper [47]. In this case, the authors use this approach to resolve spatiotemporal conflicts in AVs. In this case, they follow an approach that models the conflict points within the intersection as points of interest. The SUMO tool was used to simulate the behavior of the vehicles. The results showed that the proposed system offered a shorter vehicle waiting time than other IM schemes and a traffic light-based system. Additionally, there are other novel algorithms motivated by different fields, such as those inspired by auctions [48], those that use ant colony-based optimization [49], or those that use Monte Carlo Tree Search to obtain the order of priority to be assigned to vehicles [50].

Although the previous works showed promising results, Levin et al. [51] demonstrated that further study of the proposed algorithms is necessary since, under certain situations, FCFS may present inappropriate behaviors that can lead to inappropriate results. The control policies require a detailed and in-depth study before they become operational in real control systems. Furthermore, as can be seen, all the proposed algorithms are based on simple approaches, which do not analyze the past or future behavior of the intersection or consider the consequences of actions taken in future states. One approach that may be considered for the above would be the use of RL for vehicle control.

If we focus on RL, very few works have applied this technique in AIM, although it has been widely studied to
address traffic lights control [52]–[58]. It is very interesting the work proposed by We et al. [59]. Their proposal calculates a priority order to be assigned to each vehicle using Multi-Agent RL (MARL). The results compared with FCFS and with a variant proposed by the same authors, namely, Longest-Queue-First (LQF), showed that the use of MARL could obtain a sequence of decisions that reduced the delay by more than 60%.

An approach based on DRL can be found in [60]. In this case, the authors employed an ego-centric policy trained by reinforcement and attention learning mechanisms to develop the intersection control system. The results showed that the policy outperformed other control systems under different traffic conditions. However, this approach leaves the control to each vehicle individually, and therefore, it cannot exploit all the advantages that AIMs can offer, i.e., the benefits of centralizing the knowledge in a centralized agent of all AVs. Another work applying DRL is found in [61]. In this case, the proposal models different types of AVs with different behaviors and, through a game model based on cognitive hierarchy, allows the AVs to adapt to the reactions of the other AVs. Although the results they show are promising, it is necessary to study the performance of the proposed solution in more complex environments, with more lanes and higher vehicular flow.

In our humble opinion, it makes perfect sense to use RL to control AVs in intersections. Using RL, the system can learn and acquire in-depth knowledge of AV control through trial-and-error. Additionally, we expect that RL will provide a safer and faster solution that helps overcome the limitations of existing AIM algorithms.

**B. LEARNING FROM DEMONSTRATIONS**

RL allows solving complex problems and can provide advanced control policies. Although there are many techniques for agent optimization, one technique has generated significant interest in recent years, called Learning from Demonstrations (LfD). This technique allows to pre-train a policy quickly utilizing demonstrations of an expert and later apply RL to find another policy that improves the expert’s policy as described in [14]. In that work, a DQN algorithm was adapted to incorporate expert demonstrations. The results showed a notable boost in speed up training, allowing to fulfill the tasks of Atari Games much earlier and finding a better policy than those offered by human demonstrations.

DDPG from Demonstrations (DDPGfD) was proposed in [15] to control agents in continuous action spaces, incorporating demonstrations. The demonstrations and the actions taken by the agent were stored in a replay buffer for an unlimited time. The results showed the benefits of learning from demonstrations; the obtained policy performs the tasks more efficiently than demonstrations and solves the tasks between 2-4× steps less than DDPG.

Another approach proposed over DDPG was presented by Nair et al. [16], where they used Hindsight Experience Replay (HER) as a replay buffer sampling method. They obtained an order of magnitude improvement in speed up over RL on simulated robotics tasks. Finally, Jing et al. investigated in [62] the problem of learning with imperfect expert data. The results of their proposal revealed rapid learning of a control policy that improved on other proposed methods.

All these learning techniques rely on the existence of an expert agent that can be used to obtain demonstrations (states and actions). For the design of new AVs control systems, there are numerous simulators such as SUMO [17], CARLA [63], or VISSIM [64]. However, due to the inherent design of these simulators, there is no trained expert agent from which to obtain demonstrations. For this reason, we decided to investigate the opportunity to train an agent that we can ask (Oracle) while training a controller (through TD3) to control AVs. As a result, we propose in this paper a new method called TD3 from Oracle Demonstrations (TD3fOD).

**IV. TD3 FROM ORACLE DEMONSTRATIONS – TD3fOD**

Our method combines TD3 with demonstrations extracted from an expert (Oracle). The Oracle is trained by BC to continuously obtain demonstrations, optimizing the extracted knowledge and improving and speeding up the learning process. Below, we describe our method and evaluate these insights in our experiments.

Our algorithm presents as a novelty an Oracle from which to obtain new demonstrations. This Oracle is trained by BC with the collected experiences of the expert and modifies the parameters of \( \pi_0 \) (TD3 actor) using soft_update (soft-copy of the parameters). This soft_update is inspired by the one used by Mnih et al. [65]. In this case, the weights of \( \pi_0 \) network \( (\theta_\pi^a) \) are updated as depicted in (2).

\[
\theta_\pi^a = \tau_1 \cdot \theta_\pi^{oracle} + (1 - \tau_1) \cdot \theta_\pi^a
\]

\( \text{with } 0 < \tau_1 < 1 \) (2)

By employing soft_update, we force the actor to learn more slowly than the Oracle, increasing the stability of the training. To adjust the importance that the Oracle has on the actor, we assume that the parameter \( \tau_1 \) decreases smoothly along the simulations following (3).

\[
\tau_1 = \text{sigmoid} \left( - \frac{sim - th}{th/5} \right)
\]

The \( th \) parameter adjusts the smoothness and the number of simulations from which we consider that the learning through RL has more importance than the learning done by the soft_update of the Oracle.

As it can be seen in (3), at the beginning of the training \( (sim \in 1,2,\ldots,N_{simulations}) \), the parameters of \( \pi_0 \) will be very similar to Oracle (being \( \tau_1 \) practically 1). However, the importance that Oracle has in the changes to \( \pi_0 \) is reduced progressively until the number of simulations \( (sim) >> th \), where \( \tau_1 \) is practically equivalent to 0 and cancels the first term of (3), canceling the changes in \( \pi_0 \) due to Oracle soft_update. This evolution can be verified in Fig. 3.

In addition, our algorithm improves TD3 in several aspects:
1) A modification of the error equation so that the importance of the error produced by RL actions increases progressively. More specifically, we modified (1) (which was used to update \( \pi_\theta \)), and the new form is shown in (4), where the factor \( \tau_2 \) controls the relevance of \( Q_\phi_1 \) (Q-values of critic Q1) over the update of \( \pi_\theta \). \( \tau_2 \) is shown in (5). The evolution of both \( \tau_1 \) and \( \tau_2 \) throughout the simulations can be seen in Fig. 3.

\[
\nabla_\theta 1 \sum_{s \in \mathcal{S}} \tau_2 Q_\phi_1(s, \pi_\theta(s)) \tag{4}
\]

\[
\tau_2 = 1 - \tau_1 \tag{5}
\]

2) Incorporation of two replay buffers, one for Imitation (Oracle training) and one for RL. Moreover, these buffers use PER to speed up training by using experiences from which each network can learn more information. While PER is a technique used for RL, the original PER paper [66] suggests that it can also be employed in supervised learning.

3) The Oracle has been added to provide an expert agent to obtain new demonstrations since the simulator does not have that feature. This Oracle is trained through BC from the experiences extracted from the simulator. These experiences are stored in the Imitation replay buffer (with a fixed size). The new experiences replace the older ones when the buffer is full.

4) Finally, we add an exponential increase factor (\( \beta \)) that allows \( \pi_\theta \) to control the vehicles spontaneously and in an incremental way; meaning that for each timestep, there is a probability that \( \pi_\theta \) carries out the control of the vehicles instead of the simulator (expert). This probability increases smoothly over time until a simulation where \( \pi_\theta \) always controls all vehicles. This operation offers more stability at the beginning of the training and a gradual and smooth transition from BC to RL. Furthermore, there is a small probability that the action will be taken by \( \pi_\theta \) and not by the expert (being able to consider a kind of "sticky actions" or noise actions), the proposed procedure allows the Oracle to explore a large set of states at the start of the training, with all of the benefits that this can offer and reduce the compounding errors.

The complete TD3fOD algorithm is broken down into Algorithm 1, Algorithm 2, and Algorithm 3.

![Algorithm 1: TD3fOD](image1)

**Algorithm 1: TD3fOD**

**Input:** env Environment; \( \theta^\pi_\text{actor} \) initial actor policy parameters; \( \theta^\pi_\text{target} \) initial actor policy target parameters.

**Input:** \( \phi^1_1 \) initial Q1-function parameters; \( \phi^1_2 \) initial Q1-function target parameters.

**Input:** \( \phi^2_1 \) initial Q2-function parameters; \( \phi^2_2 \) initial Q2-function target parameters.

**Input:** \( \theta^\pi_\text{oracle} \) initial Oracle policy parameters; \( \mathcal{D}^\text{imitation} \) Imitation replay buffer; \( \mathcal{D}^\text{reinforcement} \) Reinforcement replay buffer; \( p \) coeff to expert decay; \( \text{warmup}_{\text{simulations}} \) number of simulations to pre-train actor, Q-functions, and Oracle; \( \text{imitation}_{\text{learn every}} \) every timestep \( \text{run imitation}_{\text{module}} \); \( \text{reinforcement}_{\text{learn every}} \) each timestep \( \text{run reinforcement}_{\text{module}} \);

1\) for simulation \( sim \in \{1, \ldots, N_{\text{simulations}}\} \) do:

2\) # run simulation(sim):

3\) \( \beta = p_{\text{sim}} - \text{warmup}_{\text{simulations}} \)

4\) for timestep \( t \in \{1, \ldots, \text{Max}_{\text{episdoe}}\} \) do:

5\) expert_control = True if random(0, 1) < \( \beta \) else False

6\) obtain state \( s_t \) for \( \forall \) agent

7\) if expert_control then:

8\) # Expert  selects the actions \( a_t \) for \( \forall \) agent

9\) else:

10\) # Actor selects the actions \( a_t \) for \( \forall \) agent

11\) \( a_t = \pi_\theta(s_t) \)

12\) env. actions(\( a_t \))

13\) end if

14\) # Get next state and reward for \( \forall \) agent

15\) \( r_t, d_t, s_{t+1} \leftarrow \text{env. step}() \)

16\) if expert_control then:

17\) # Obtain \( a_t \) comparing \( s_t \) and \( s_{t+1} \) for \( \forall \) agent

18\) # Store (\( s_t, a_t \)) in \( \mathcal{D}^\text{imitation} \)

19\) end if

20\) # Store transition (\( s_t, a_t, r_t, s_{t+1}, d_t \)) in \( \mathcal{D}^\text{reinforcement} \)

21\) if \( t \mod \text{imitation}_{\text{learn every}} = 0 \) then:

22\) run imitation_{\text{module}}(sim)

23\) end if

24\) if \( t \mod \text{reinforcement}_{\text{learn every}} = 0 \) then:

25\) run reinforcement_{\text{module}}(sim)

26\) end if

27\) end for

28\) end for

1\) We can’t ask to an expert to obtain the actions.

![Algorithm 2: Imitation Module](image2)

**Algorithm 2: Imitation Module**

1\) # Run imitation module (sim):

2\) for epoch \( e \in \{1, \ldots, N_{\text{imitation epochs}}\} \) do:

3\) for mini batches \( \mathcal{B} \) in \( \mathcal{D}^\text{imitation} \) do:

4\) actions, states = unzip(\( \mathcal{B} \))

5\) actions_pred = \( \pi_\text{oracle}(\text{states}) \)

6\) # Update Oracle params (\( \theta_\text{oracle} \)) with one step of gradient descent.

7\) end for

8\) end for

9\) \# soft_update(\( \pi_\text{oracle}, \pi_\theta, \tau_1 \)):

10\) \( \theta^\pi_\text{actor} = \tau_1 \cdot \theta^\pi_\text{actor} + (1 - \tau_1) \cdot \theta^\pi_\text{oracle} \)

**FIGURE 3.** Evolution of \( \tau_1 \) and \( \tau_2 \) in function of \( \theta \) parameter.

---

*This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI: 10.1109/ACCESS.2022.3175493, IEEE Access.*
Algorithm 3: TD3 Reinforcement Module

# Run reinforcement module (sim):

for epoch $e \in \{1, \ldots, N_{\text{reinforcement\_epochs}}\}$ do:

1. Sample a mini_batch $B$ from $\mathcal{D}_{\text{reinforcement}}$

2. $s_t, a_t, r_t, s_{t+1}, d_t = \text{unzip}(B)$

3. $\alpha_t = (\Theta(s_t) + \epsilon).\text{clamp}(a_{\text{min}}, a_{\text{max}})$

4. Update Q-functions parameters $\phi_1^Q$ and $\phi_2^Q$ as in TD3

if $\text{mod\_update\_actor\_every} = 0$ then:

7. $\tau_2 = 1 - \tau_3$

8. Update actor policy with one step of gradient ascent with equation 4 and using $\tau_2$

9. Update target networks as in TD3 and using $\tau_3$

10. $\text{soft\_update}(\phi_1^Q, \phi_2^Q, \tau_3)$

11. $\text{soft\_update}(\phi_3^Q, \phi_4^Q)$

12. $\text{soft\_update}(\pi_0, \pi_0', \tau_2)$

end if

end for

V. EXPERIMENTAL SETUP

Our algorithm focuses on simulators or environments where there is no expert to ask or obtain feedback about past experiences (although it is possible to obtain new demonstrations).

However, because the controller is internally modeled and is not possible to access due to the nature of the simulator (lack of an API and/or closed-source software licensing), our algorithm exploits new demonstrations to build an expert to ask (Oracle) and then train a new control system by RL via LfD. By taking advantage of the benefits offered by LfD, training is greatly accelerated and improved. For example, a simulator that has these characteristics is SUMO [17]. SUMO is a microscopic simulator where each vehicle is explicitly simulated and is widely used by the scientific community and urban planners to obtain better traffic controllers or optimize the existing ones. For this reason, it was decided to use this simulator for this study. TD3fOD was programmed in Python 3.7 and Pytorch 1.5.0. A 16-core processor was used as a CPU, together with an Nvidia 2080TI GPU.

A. RAIM OVER TD3fOD (RAIMfOD)

TD3fOD was used to train RAIM [11], an algorithm developed for AIM systems. RAIM can control the speed of AVs in the surroundings of intersections so that the flow and safety of these vehicles can be notably increased, significantly reducing waiting time, pollutant emissions, and consumption of both fuel and power electricity. RAIM leverages the advantages provided by MADRL, to find a policy to control vehicles intelligently, collectively, and collaboratively. RAIM belongs to MADRL’s Centralized Trained and Centralized Execution Cooperative approach, where vehicles send their states to the AIM. It is the AIM itself that obtains the action for each AV (centralized training and execution in the AIM) with a common goal [19].

In the original article, RAIM was trained through curriculum-based learning, increasing simulated vehicle flow when some stability in the results was achieved. The proposed solution optimized the system but required a large number of simulations and a considerable amount of training time. By using TD3fOD, we aim to accelerate learning and reduce the number of simulations without using curriculum-based learning. The new approach is called RAIM from Oracle Demonstrations (RAIMfOD). The network architecture, for both the actor and the critics, consisted of four fully connected layers (448, 128, 50, and 1 neurons for each layer). The input of the network included the characteristics of the vehicles to be controlled (e.g., position, speed, route, lane, etc.) and the output indicated the speed at which the vehicle should drive during the next time interval. These features, as well as the inner workings of RAIM, can be seen in more detail in the original RAIM article [67].

B. TRAINING SCENARIO

The training scenario was used to optimize TD3fOD/RAIMfOD. This scenario consisted of an intersection of four branches and three lanes for each branch, where it was allowed to turn left, turn right, and go straight, one movement for each lane. A representation of the simulated intersection can be seen in Fig. 4.

As a reward signal, the following sparse reward was designed. Each agent (vehicle) received each timestep: $+10$ (strong positive reward) when the vehicle crossed the intersection, $-10$ (strong negative reward) when the vehicle collided with another vehicle, and $-\text{timestep}$ otherwise to promote crossing the intersection as fast as possible. Table I includes the values of the hyperparameters. These hyperparameters and the reward values were empirically selected, offering notable performance and a stable training process.

![Representation of simulated intersection with 4 approaches and 3 lanes/approach, where the movements go straight, turn right, and turn left are allowed.](image)

**TABLE I**
C. TESTING SCENARIO

A test scenario was incorporated to test the ability of our algorithm to face never-before-seen scenarios. For this purpose, a scenario with a traffic distribution that presents multiple variations was proposed, with low flows (500 veh/h), medium flows (1000 veh/h), and high flows (2000 veh/h). In addition, it presented asymmetric and symmetric traffic regarding the branches of origin, North/South (N/S) and West/East (W/E). The intersection was the same as in the previous scenario, with three lanes, where left, right, and straight turns were allowed. The time distribution of the simulated flow can be seen in Fig. 5. The algorithms used to compare the performance of RAIMfOD were: no control, a stop signal, a fixed-time traffic light algorithm with total cycles of 30, 60, and 90 s, iREDVD [68] (a traffic light algorithm based on queueing theory), the previous RAIM algorithm and the direct implementation of RAIM with DDPGfD [15].

The following key performance metrics were used to compare the different algorithms: travel time, waiting time, time loss due to congestion, and pollution and consumption metrics (CO, CO2, HC, PMx, NOx, and fuel and electric). The vehicle distribution used was: 35% of diesel cars, 35% of gasoline cars, and 30% of electric cars with zero emissions.

VI. RESULTS

This section shows the results obtained in both the training and test scenarios.

A. TRAINING SCENARIO

Fig. 6 depicts the reward of each episode throughout the simulations and the average time loss by each vehicle in the training scenario. The time loss is due to driving below the ideal speed and is defined as follows: total_duration * (1 - speed/ideal_speed). The results show a significant improvement in several aspects.

The main improvement is the increase in training speed, reducing by 5 to 6 times the number of simulations needed to achieve even better performance than the original DRL TD3 (RAIM), with reduced variance of the results and a much more stable and robust policy. Compared to DDPGfD, it can be seen that the use of an Oracle from which to extract feedback and the improvements proposed in TD3 allow to increase the speedup threefold and to reduce considerably the variance in the control behavior of the trained RAIM policy. If we look at Fig. 6b, we can see that RAIM with TD3fOD (RAIMfOD) is able to reduce the time loss to below 20 seconds over the 200 simulations. Comparatively, the original RAIM must train more than 1500 simulations to obtain a policy that reduces 20 seconds of time loss. Fig. 6b depicts the reward metric. As it can be seen, it reduces the number of simulations by more than x6, accelerating the training of new advanced control systems. Finally, comparing the training results with those obtained with traditional control techniques shows how the metrics are much better with RAIM and RAIMfOD.

In Fig. 6a and Fig. 6b, there are three different phases of the RAIMfOD. The pre-training begins between simulations 0 and 100, filling the Imitation and RL replay buffers. The pre-training ends from simulation 100 to simulation 250, and RAIMfOD starts. This situation can be seen in the shift in the metrics' trend over iteration 100. In this range of simulations, the transition between learning by "soft-copying" the Oracle and the RL of TD3 begins, with the simulator taking more actions and the actor in TD3 acting as a "sticky action." From simulation 250 onwards, the \( \tau_1 \) and \( \tau_2 \) curves cross each other, and most of the actions are carried out by the TD3 actor, allowing to find a better control policy to optimize the results further. This highlights the notable performance of LfOD, allowing to achieve a policy that outperforms the one offered by the expert through a smoothed step from a pre-trained policy made by an Oracle to the policy learned by RL.

B. TESTING SCENARIO

We demonstrate our algorithm’s ability to generalize and adapt to new situations in the test scenario. This is how we illustrate the benefits offered by MADRL and LfOD. The
results obtained are shown in Table II. In this table, it can be seen that RAIM with TD3fOD and original RAIM obtain very similar results, demonstrating that both algorithms find solutions with very notable performance, but RAIMfOD finds a policy much earlier and with a much lower variance because of LfOD, as can be seen in Fig. 6.

From the results included in Table II, we confirm a reduction of time loss between 95% and 86% and a reduction of waiting time between 97% and 93%. This represents a reduction in travel time of between 72% and 53%. Regarding the emission of pollutant gases, we see that a significant improvement has been achieved in all the metrics, decreasing all studied variables by up to over 50%. Finally, in terms of vehicles’ fuel and electricity consumption, a reduction of between 29% and 5% is achieved for combustion vehicles, and between 34% and 24% for electric vehicles. These results conclude the potential of LfOD and MADRL systems for the control of AVs in a centralized approach.

C. DISCUSSION
The results obtained in this work demonstrate the benefits of using MADRL with TD3fOD to solve different real problems. This solution has the potential to solve these problems in real-time, decreasing the time needed to train new systems and improving the performance of existing ones. The TD3fOD method is able to find a policy that significantly improves the results obtained by traditional training techniques, resulting in more stable and robust policies.

When the performance of RAIM with TD3fOD (RAIMfOD) is compared to the performance of the original RAIM, we notice that the learning performance of RAIMfOD is between ×5 and ×6 faster, and the variance of the results is substantially smaller. Furthermore, when compared to RAIM with DDPGfD, we can see that, thanks to the advantages offered by TD3, as well as the use of Oracle, we can reduce the number of simulations by up to ×3, also significantly reducing the variance in the policy while training. Finally, if we look at the test scenario, we see that the analyzed metrics outperform the original RAIM, showing the robustness of the proposed algorithm in scenarios never seen before, obtaining an improvement of between 72% and 53% in travel time and a reduction in waiting time of between 97% and 93%. Furthermore, most cases reduce pollutant gas emissions by more than 50% and energy or fuel consumption by nearly 30%.

VII. CONCLUSIONS
The success of AVs depends on advances in various driving and control systems components and the understanding and handling of unpredictable situations that can arise in complex driving environments. The application of MADRL allows the development of dynamic systems capable of adapting to many conditions and acting collectively and proactively, anticipating dangerous situations and ultimately preventing accidents and increasing flow. LfD can provide a simple way to find adaptive control policies capable of solving highly complex tasks in different fields of work, such as teaching robots how to walk, navigate through dangerous traffic, manage the presence of obstacles, avoid collisions with other road users, perform safe and efficient maneuvers at intersections, and more.

To enable the use of an expert’s demonstrations in environments where the expert is not accessible, we have proposed in this work the use of an Oracle in LfD, obtaining LfOD.

The Oracle is trained by Imitation Learning; thus, it can be employed to teach from demonstration an agent using RL. This original approach facilitates using LfD in environments with no expert from whom to obtain feedback. In this way, an agent can be trained much more quickly, achieving a better policy than the expert could offer and presenting a lower variance in the results.

![Image](image_url)

**FIGURE 6.** Training results. (a) Episode reward evolution (b) Time Loss evolution. RAIM with TD3fOD was able to learn a robust policy faster than the original RAIM. We plot the smoothed mean with an exponential moving average and a 90% confidence interval across 3 seeds.
TD3 was the algorithm modified to use the demonstrations offered by Oracle. Following the nomenclature used in the algorithms developed for LiD, we called the new proposal TD3 from Oracle Demonstrations (TD3fOD). The modifications made to TD3 were: i) incorporation of an Oracle trained by Imitation Learning from the states extracted from the simulator; ii) include several parameters for a smooth and progressive transition between LiOD and RL; and iii) use of two replay buffers, one for demonstrations to train Oracle and the other for RL, in addition to the use of PER to speed up learning. TD3fOD was applied in the SUMO traffic simulator for a complete network of traffic intersections and progressive transition between LiOD and RL; and improved those obtained by RAIM in all studied scenarios, improving those obtained by RAIM in all studied metrics and with a lower variance. These results highlight the benefits offered by LiOD. RAIM over TD3fOD reduces waiting time between 97% and 93%, resulting in a reduction of up to 50% in the emission of contaminating gases compared to other traditional vehicle control techniques such as traffic lights and other advanced techniques such as iREDVD. Finally, if we look at consumption, combustion vehicles reduce their fuel consumption by up to 29% and electric vehicles by 34%. Furthermore, if we compare TD3fOD with another LiD algorithm such as DDPGD we show that the use of the proposed LiDO approach allows to speed up the training, reducing the number of interactions with the simulator by up to x3.

Thanks to our proposal, it is possible to extract the hidden agent (learned by imitation with an Oracle) in those simulations where it is not possible (or too complicated), in order to take advantage of the benefits offered by LiD (training acceleration and more robust policies) for the development of new complex control algorithms. The proposed approach for LiD is applicable to different RL algorithms, being one of the main contributions of this work the development of an expert agent (Oracle) in environments where it does not exist to take advantage of LiD, as well as the incorporation of this approach in the TD3 DRL algorithm, making severe modifications in TD3 to adapt the training process to the presence of Oracle.

As future work, we plan to include multiple hierarchical systems that enable level-based control of the different actors in a complete network of traffic intersections and explore the development of new algorithms that can use LiD in other domains.

### TABLE II

| Algorithm | Travel Time (s) | Waiting Time (s) | Time Loss (s) | CO Emiss. (g) | CO2 Emiss. (g) | HC Emiss. (mg) | PMx (mg) | NOx Emiss. (mg) | Fuel Cons. (ml) | Elect. Cons. (W) |
|-----------|----------------|-----------------|--------------|--------------|--------------|--------------|----------|----------------|---------------|----------------|
| No control | 121.42 ± 25.17 | 83.32 ± 4.94  | 91.44 ± 1.23 | 2.47 ± 0.86  | 111.78 ± 2.45| 13.97 ± 0.36 | 5.21 ± 0.15| 375.87 ± 12.6 | 41.24 ± 3.3 | 33.97 ± 2.88 |
| Stop      | 107.53 ± 13.45 | 72.65 ± 5.23  | 78.15 ± 3.21 | 1.74 ± 0.45  | 81.70 ± 1.65 | 11.65 ± 0.32| 3.52 ± 0.12| 272.65 ± 13.4 | 35.33 ± 3.44| 35.44 ± 2.88 |
| Signal   | 8.94 ± 0.94    | 12.42 ± 1.23  | 19.85 ± 2.74 | 0.99 ± 0.15  | 32.44 ± 2.74 | 6.01 ± 0.23 | 0.99 ± 0.09| 37.95 ± 2.74 | 8.99 ± 2.88 | 2.88 ± 0.27 |
| TL30     | 86.85 ± 10.85 | 50.07 ± 4.32  | 55.46 ± 3.21 | 1.08 ± 0.15  | 69.45 ± 2.65 | 7.32 ± 0.23 | 2.61 ± 0.15| 194.99 ± 13.4 | 30.55 ± 3.44| 32.39 ± 2.74 |
| TL60   | 4.43 ± 0.94    | 15.21 ± 1.23  | 17.83 ± 2.74 | 0.64 ± 0.15  | 23.64 ± 2.74 | 3.23 ± 0.23 | 0.86 ± 0.09| 31.43 ± 2.74 | 7.66 ± 2.97 | 2.97 ± 0.27 |
| TL90  | 8.97 ± 1.02   | 16.98 ± 2.32  | 19.26 ± 3.21 | 0.83 ± 0.15  | 38.52 ± 2.78 | 2.78 ± 0.92 | 0.92 ± 0.09| 37.45 ± 2.78 | 8.31 ± 3.04 | 3.04 ± 0.27 |
| iREDVD | 7.26 ± 0.94    | 27.12 ± 3.25  | 32.25 ± 4.32 | 1.49 ± 0.15  | 81.34 ± 2.74 | 9.00 ± 3.18 | 3.18 ± 0.15| 221.94 ± 3.74 | 35.60 ± 3.55 | 36.50 ± 2.87 |
| RAIM  | 6.75 ± 0.94    | 9.98 ± 1.23   | 15.23 ± 2.74 | 0.43 ± 0.15  | 23.42 ± 2.74 | 3.05 ± 0.67 | 0.67 ± 0.09| 33.95 ± 2.74 | 9.85 ± 2.88 | 2.87 ± 0.27 |
| RAIM over DDPGD | 3.41 ± 0.94 | 1.92 ± 2.32 | 2.22 ± 3.21 | 1.08 ± 0.15 | 33.88 ± 2.74 | 2.03 ± 0.55 | 0.55 ± 0.09 | 24.92 ± 2.74 | 4.32 ± 2.66 |
| RAIM over TD3fOD | 3.32 ± 0.94 | 1.86 ± 2.32 | 2.12 ± 3.21 | 1.02 ± 0.15 | 52.34 ± 2.74 | 6.79 ± 3.18 | 2.47 ± 0.15 | 136.85 ± 3.74 | 28.88 ± 2.43 | 24.03 ± 2.88 |

**Improvement of RAIM over TD3fOD**

| Abs Value | [-88.19, -81.46, -87.23, -1.45, -58.24, -7.18, -2.74, -239.02, -12.36, -12.61] |
|-----------|--------------------------------------|
| [%]       | [72.62, 97.77, 95.40, 58.70, 53.01, 51.40, 52.39, 63.59, 29.97, 34.42] |

The improvement shows the range between [best-case, worst-case]. It is shown as an absolute value (abs) in the corresponding units and percentage value (%).

TLXX means Traffic Light with a fixed total cycle duration of XX segs.
subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2006, pp. 1–46.

G. Chen, “A new framework for multi-agent reinforcement learning - centralized training and exploration with decentralized execution via policy distillation,” in Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS, 2020, pp. 1–10.

P. K. Sharma, E. G. Zaroukian, R. Fernandez, A. Basak, and D. E. Asher, “Survey of recent multi-agent reinforcement learning algorithms utilizing centralized training,” in Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III, 2021, p. 84.

S. Fujimoto, H. van Hoof, and D. Meger, “Addressing Function Approximation Error in Actor-Critic Methods,” in 35th International Conference on Machine Learning, ICML 2018, 2018, vol. 4, pp. 2587–2601.

T. P. Lillicrap et al., “Continuous control with deep reinforcement learning,” in 4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings, 2016.

A. Raffin, J. Kober, and F. Stulp, “Smooth Exploration for Robotic Reinforcement Learning,” in 5th Conference on Robot Learning (CoRL 2021), 2020, pp. 1–15.

A. Stooke and P. Abbeel, “rlpyt: A Research Code Base for Deep Reinforcement Learning in PyTorch,” Sep. 2019.

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in 35th International Conference on Machine Learning, ICML 2018, 2018.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” Jul. 2017.

V. Mnih et al., “Asynchronous methods for deep reinforcement learning,” in 33rd International Conference on Machine Learning, ICML 2016, 2016, pp. 1928–1937.

S. Ross and J. A. Bagnell, “Efficient reductions for imitation learning,” J. Mach. Learn. Res., vol. 9, pp. 661–668, 2010.

M. Bain and C. Sammut, “A Framework for Behavioural Cloning,” Mach. Intell., vol. 15, no. 1, pp. 103–129, 2001.

S. Daftry, J. A. Bagnell, and M. Hebert, “Learning Transferable Policies for Monocular Reactive MAV Control,” in Springer Proceedings in Advanced Robotics, vol. 1, 2017, pp. 3–11.

H. Daume, J. Langford, and D. Maru, “Search-based structured prediction,” Mach. Learn., vol. 75, no. 3, pp. 297–325, Jun. 2009.

K. Dresner and P. Stone, “A Multiagent Approach to Autonomous Intersection Management,” J. Artif. Intell. Res., vol. 31, pp. 591–656, Mar. 2008.

Peter Stone and Kurt Dresner, “Human-usable and emergency vehicle–aware control policies for autonomous intersection management,” in Fourth International Workshop on Agents in Traffic and Transportation (ATT), Hakodate, Japan, 2016, no. May, pp. 17–25.

K. Dresner and P. Stone, “Sharing the road: Autonomous vehicles meet human drivers,” in The Twentieth International Joint Conference on Artificial Intelligence (IJCAI 07), 2007, vol. 1, no. January, pp. 1263–1268.

R. Tachet et al., “Revisiting Street Intersections Using Slot-Based Systems,” PLoS One, vol. 11, no. 3, p. e0149607, Mar. 2016.

D. Fujardo, T.-C. Au, S. T. Waller, P. Stone, and D. Yang, “Automated Intersection Control,” Transp. Res. Rec. J. Transp. Res. Board, vol. 2259, no. 1, pp. 223–232, Jan. 2011.

S. Huang, A. W. Sadek, and Y. Zhao, “Assessing the Mobility and Environmental Benefits of Reservation-Based Intelligent Intersections Using an Integrated Simulator,” IEEE Trans. Intell. Transp. Syst., vol. 13, no. 3, pp. 1201–1214, Sep. 2012.

X. Wei, G. Tan, and N. Ding, “Batch-Light: An adaptive intelligent intersection control policy for autonomous vehicles,” in 2014 IEEE International Conference on Progress in Informatics and Computing, 2014, pp. 98–103.

J. Lee and B. Park, “Development and Evaluation of a Cooperative Vehicle Intersection Control Algorithm Under the
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3175493, IEEE Access