ECAI 2023
K. Gal et al. (Eds.)

doi:10.3233/FAIA230304

Deep Reinforcement Learning with Implicit Imitation for Lane-Free Autonomous Driving

Iason Chrysomallis[a;*], Dimitrios Troullinos[a], Georgios Chalkiadakis[a], Ioannis Papamichail[a] and Markos Papageorgiou[a]

[a]Technical University of Crete, Chania, Greece

Abstract. *Implicit imitation* assumes that learning agents observe only the state transitions of an agent they use as a mentor, and try to recreate them based on their own abilities and knowledge of their environment. In this paper, we put forward a deep implicit imitation Q-network (DIIQN) model, which incorporates ideas from three well-known Deep Q-Network (DQN) variants. As such, we enable a novel implicit imitation method for *online*, *model-free* deep reinforcement learning. Our thorough experimentation in the complex environment of the emerging *lane-free traffic* paradigm, verifies the benefits of our approach. Specifically, we show that *deep implicit imitation RL* dramatically accelerates the learning process when compared to a "vanilla" DQN method; and, unlike *explicit* imitation reinforcement learning, it is able to outperform mentor performance without resorting to additional information, such as the mentor's actions.

1 Introduction

Deep Reinforcement Learning (DRL) [4], combining as it does Reinforcement Learning (RL) [43] with neural networks, has gained popularity in several problem domains and games, with DRL-based agents that can even outperform human experts in the field. In environments with substantially large or continuous state spaces where tabular reinforcement learning fails to provide competent results, DRL provides a methodology with the ability to tackle such complex problems that were out of reach otherwise. Still, problems with high complexity usually require substantially more training steps for DRL algorithms.

As such, the need for accelerated training arises. Provided with a highly trained agent, hastened improvement can be achieved through the use of *imitation* [36, 1] methods. The most common applications of imitation in machine learning take advantage of *explicit imitation* [15]. According to explicit imitation, the skilled agent—*the mentor*—transfers most, if not all, information available to it (commonly states, actions taken, and rewards received) to the *trainee* observer agent, granting the latter the opportunity to explicitly imitate its behaviour [3]. However, this comes with the price of having to communicate information, which may not be available. Depending on the environment, the agent, or data privacy restrictions, observability of the expert's actions or policy-related information is not always possible. For example, observing a human driver in a vehicle, we may not be in a position to monitor their throttle press values. Since complete transparency is not always available, an *implicit imitation* [36] paradigm can be used instead. With just the state transition observations of the mentor, the trainee can decrease its training time and reach the skill level of its mentor in fewer episodes, by filling the blanks of the information it needs, using its own heuristic approaches. Moreover, implicit imitation, even in tabular RL, has demonstrated an ability to outperform the mentor [36] without requiring access to further information.

In this work we put forward a framework and algorithms for model-free *deep implicit imitation reinforcement learning (DIIRL)*. Specifically, we describe an implicit imitation model and its integration into the arguably most widely used model-free DRL technique, the *Deep Q-Network (DQN)* [29] algorithm. Using neural networks for RL along with mentor observations, complex continuous state space can be solved in significantly less time. We detect state similarities between the observer and the mentor using the mentor's state transition demonstrations, so that the observer can then infer the mentor's action and use it to calculate TD error values based on *augmented loss functions* we introduce. Since the observer is not explicitly restricted to imitate the mentor's policy, the possibility for the observer to actually surpass a potentially sub-optimal mentor emerges.

A highly challenging domain for evaluating our approach lies within the area of lane-free autonomous driving. Autonomous driving is a complex field when taking into consideration the plethora of dynamic variables involved [34]. The lane-free traffic paradigm [33] further increases its complexity by not restricting the vehicles to a standard predetermined number of lanes for them to abide, but providing free lateral movement. This novel paradigm calls for the development of novel approaches in (optimal) control [33, 54, 26], multiagent coordination [49], simulator environments [48], and machine learning methods [17]. In particular, the opportunity emerges for the use of existing, or the formulation of novel (deep) reinforcement learning algorithms that take into account the position and behaviour of nearby vehicles, and aim to adjust the (both longitudinal and lateral) acceleration of the vehicle under control [17]; and for testing DIIRL in this interesting and challenging domain.

Summarizing, our contributions are the following: we provide DIIQN, a novel DQN-based method for DRL *implicit imitation*; and enhance it via incorporating the key ideas of three well-known DQN variants as algorithmic components of our model. Central in our approach is the creation of novel augmented loss functions to update our network's weights in accordance with implicit imitation. Moreover, our approach is an *online*, *model-free* RL one: it does not include the offline supervised pre-training phase common in DRL imitation techniques; and makes use of mentor-provided information while interacting with the environment. Our experimental results in

* Corresponding Author. Email: ichrysomallis@tuc.gr.

the challenging lane-free traffic autonomous driving environment, demonstrate that deep implicit imitation can both accelerate DRL agent training, and improve its performance over the mentor agent.

The remainder of the paper is structured as follows. Section 2 provides background on DRL and imitation learning, as well as related work on imitation and lane-free traffic. Section 3 presents in detail our deep implicit imitation RL model. Section 4 describes our extensive experimental evaluation process and our results. Finally, Section 5 concludes this paper and outlines future work.

2 Background and Related Work

Here we provide background and discuss related work.

2.1 Deep Q-Networks

Deep Q-Networks (DQN) [29], a distinguished method of deep reinforcement learning, combines Q-learning [53] with neural networks. A deep Q-network is a multi-layered neural network with weights θ_t, that approximates the Q action-value function of the standard Q-learning algorithm utilizing a neural network representation $Q(s, \alpha; \theta_t)$. As such, given a transition tuple (s, a, r, s', γ),[1] the current Q-network's weights θ_t are optimized w.r.t. the loss function:

$$L(\theta_t) = E_{s,a,r,s'}[(y_t^{DQN} - Q(s, a; \theta_t))^2] \quad (1)$$

where $y_t^{DQN} = r + \gamma \max_{a'} Q(s', a'; \theta_t^-)$ is the target value corresponding to the updated evaluation of the Q-function, at current time-step t. The notion of a *target network* is used in DQN to provide stability in training. This is achieved using a snapshot of the network's weights θ_t^- from a specific prior time-step for the y_t^{DQN} estimate. Weights θ_t^- are periodically synchronized with the current weights θ_t. Another important DQN component is *experience replay* [22, 29], a data structure that stores past experiences. In each episode, the network is updated based on a small minibatch uniformly sampled from the replay memory, breaking consecutive sample state chains that can misguide the training process with over-fitting data.

Now, the use of *Double DQN (DDQN)* [11] tackles value overestimation issues: Since DQN is based on a single estimator method, numerous occasions of value overestimation can be observed during training [41, 10]. To diminish this overestimation, the DDQN target value has the form $y_t^{DDQN} = r + \gamma Q(s', \text{argmax}_a Q(s', a; \theta_t); \theta_t^-)$.

Moreover, prioritizing high TD-error samples via a *Prioritized Experience Replay (PER)* [37] has led to more efficient training, using *importance sampling* [25] to balance the magnitude under which the samples affect the network's weights.

Finally, *Dueling Network Architecture (DNA)* [52], decouples the Q-network architecture so that it estimates a value function $V(s)$ and a state-dependent action advantage function $A(s, a)$ separately, improving performance in the face of redundant or similar actions.

2.2 Imitation Learning

The concept of transferring knowledge among agents for the benefit of constant improvement is not new [5, 27]. The core idea behind imitation reinforcement learning includes two agents, the mentor and the observer. The observer is the agent under training, with unknown starting values to all predictions, trying to learn from scratch. The mentor agent has already undergone training up to a satisfactory level, rendering it capable of guiding the observer through different proposed methods [36, 42]. The goal of imitation learning is to accelerate the learning process of the observer, by feeding information provided by the mentor. Traditionally, this can be achieved either explicitly, implicitly [36] or by other means, such as third person imitation with unsupervised learning [42].

Explicit imitation requires a set of first-person point of view information. For RL algorithms, these mainly include the state transitions, the actions taken, and the rewards received [36]. When combined with DRL and other methods, explicit imitation can contribute to outperforming mentor behavior [40, 31, 51, 12, 16]. The state transitions information is essential for imitation RL algorithms (both explicit and implicit). However, the remaining information, primarily the actions taken, may be private or hard to observe and acquire.

Our focus thus is on *implicit imitation*: i.e., the observer can note only the changes of the mentor's experience environment, leaving to fill in the blanks for actions taken. *Implicit Imitation* attempts to approximate the mentor's behavior by observing the state transitions, but estimating (or, "predicting"[2]) the actions resulting to them. It should be highlighted that in implicit imitation the observer does not blindly follow the mentor's directions. This results to a more flexible and stable observer agent that aims to adapt and even outperform the mentor in, especially, non-visited states. To define correctly the mentor's shared knowledge, some assumptions are required [36]: *(i)* The observer has access to the mentor's state transitions, but not its action taken in each step; *(ii)*: Both mentor and observer use the same local state spaces, otherwise the observations would be irrelevant and utterly unusable by the learner; *(iii)*: There is homomorphism between the abilities of the mentor and the observer agent: both action spaces are assumed to coincide, and state transitions follow the same MDP for each $a \in A$; *(iv)*: In our work, as in [36], both mentor and observer have matching reward functions.

2.3 Related Work

The primary influence of our approach is that of [36, 35] on model-based (non-deep) reinforcement learning through implicit imitation. Our work is a *model-free*, *deep* RL variant of their approach, inspired by a conceptual extension for model-free RL, proposed as part of future work in [36].

Several other methods on imitation learning exist, making use of either explicit or implicit imitation models. Regarding explicit imitation algorithms, they typically either introduce a pre-training step applying explicit imitation with the use of supervised learning [40, 12], or inject mentor samples directly into the replay memory of the observer [31, 51, 16]. Other interesting explicit imitation approaches include taking advantage of human experience combined with a checkpoint system [14] and introducing a two-level hierarchical technique to the imitation learning application [19]. Finally, [42] use a *third-person imitation* approach, which is a form of explicit imitation in which the mentor's demonstrations do not comply with the perspective of the observer.

Now, as far as implicit imitation applications are concerned, they are mainly categorized into two groups [47]; model-based and model-free. Model-based approaches make use of dynamics models, mainly in the form of inverse dynamics models [30, 23, 45], using mentor demonstrations to infer an action policy that is later exploited in a supervised or model-based manner. In this model-based

[1] Transition tuple with state s, action a, reward r, next state s' and discount factor γ.

[2] We use "predict" in lack of a better term; we do not choose to use the word "infer" as this term points to approaches that construct a mentor model.

Algorithm 1 High-level overview of the DIIQN algorithm

Input: Mentor extracted $experiences$

1: Initialize environment, DQN model and import $experiences$;
2: Enable pre-demonstration phase for initially associating actions to $experiences$ (as discussed in Section 3.1.2)
3: **while** environment not solved **do**
4: Select observer action a_o based on policy π and ϵ-greedy exploration;
5: Predict mentor action a_m (as explained in Section 3.1.2) via exploiting a state similarity (*findSimilarStates()* assessment process, presented in Section 3.1.2);
6: Create a new "augmented" observer experience tuple, which includes the predicted a_m, and store it in the experience replay;
7: Sample minibatch from experience replay;
8: Compute values of augmented loss functions L_o, L_m (defined in Section 3.1.3) with respect to the sampled minibatch;
9: Select suitable augmented loss function using Q-value divergence D_o, D_m (as explained in Section 3.1.3);
10: Optimize network parameters θ_t with respect to the selected augmented loss function;
11: **end while**

paradigm, forward dynamics models [6] also tackle a similar problem of missing information, predicting the next state s' provided a state s and an action a. Model-free approaches comprise adversarial methods [28, 46], using as discriminator a model trained by the mentor's data.

Finally, many imitation learning approaches involve the use of *inverse* RL (IRL) [32, 1], i.e., inferring a (human or non-human) mentor's reward function. For instance, [13, 7] explore IRL to extract an expert mentor policy using adversarial networks. Then, authors in [2] design an elaborate imitation learning model that employs IRL and behavioural cloning in an interactive training setup.

It is important to highlight that our approach does not include the supervised pre-training or explicit sample injection phases common in DRL imitation techniques. Even though we have a pre-demonstration phase which initializes mentor action estimates information (see Sec. 3.1.2), we do not build an inverse dynamics mentor model, but simply utilize the collected information from the mentor in a model-free way while interacting with the environment at each step. Consequently, ours is a model-free online RL algorithm.

Regarding the emerging lane-free traffic paradigm, a plethora of works have introduced novel ideas for vehicle movement strategies [54, 33, 49, 50]. In terms of RL, the work of [17] proposed a DRL setting for lane-free autonomous driving. Their approach defined MDP-related variables such as the state space and, more importantly, the reward function. Even though their work heavily focuses on continuous action space with the use of the Deep Deterministic Policy Gradient algorithm [21], the integration of our algorithm into the lane-free domain was based on their MDP formulation.

3 Deep Implicit Imitation RL

We now describe our framework for enabling implicit imitation for DRL. We first describe how to incorporate implicit imitation in DQN, giving rise to DIIQN, our novel basic implicit imitation-equipped DQN algorithm. We then further improve our deep imitation model by incorporating additional DQN variants into our framework.

3.1 Implicit Imitation for DQN

DIIQN can be viewed as a modular extension of DQN, i.e., DQN with the additional capability to exploit a mentor's dataset in order to accelerate training. The mentor provides a dataset of state transitions $\langle s_m, s'_m \rangle$, namely *experiences*, which serves as input to the observer. Training the observer while utilizing mentor's information requires an estimate of the mentor's action a_m for the proper calculation of the augmented loss functions L_o, L_m we introduce in this section—these augmented loss functions are responsible for optimizing the observer's network. To do so in an implicit manner, we associate the mentor's states s_m with an estimate for the action, which is iteratively updated while the observer interacts with the environment. These mentor action estimates are gradually improved by the observer. Essentially, the observer's state information is juxtaposed with the state information of the mentor, and, with KL-divergence, we can have an error metric in order to measure similarity of states.

To provide structure to the different DIIQN processes, we divide it into separate components. In Alg. 1 we summarize the steps taken in our DIIQN model. We now proceed to explain these steps in detail.

3.1.1 Experience Extraction

Before initiating the process of DIIQN, we need access to a dataset with the mentor's extracted *experiences*. For our purposes, the mentor can be a standard DQN model, or any type of algorithmic controller. In the former case, we first train the DQN agent until convergence, and view the trained model as a mentor. Regardless of the mentor's form, we monitor its interactions with the environment until N observations are collected. Each observation contains *only* the state transition, i.e., a tuple of $\langle s_m, s'_m \rangle$, where s_m is the mentor state before it takes an unknown action a_m, while s'_m is the resulting state after the action execution. Upon initialization of DIIQN, the observer agent has access to all recorded observation tuples, indicated as mentor extracted *experiences* in Alg. 1. Likewise, multiple mentors can be directly integrated by combining their observations in a single dataset. We highlight that only state information is being tracked, and neither the mentor action a_m, nor the received reward r_m is known to the observer.

3.1.2 Mentor Action Prediction

The observer can make use of mentor experiences by correlating its encountered states with those of the mentor, in order to "fill in the blanks" regarding the actions actually taken by the mentor (which are not observed). We now proceed to describe a process (Alg. 2) that predicts the mentor action a_m for an observer's transition $\langle s_o, a_o, s'_o \rangle$—without relying on model-based techniques commonly found in the literature (see Sec. 2.3). This information is required for DIIQN, specifically when we calculate the augmented loss function from the mentor's perspective, as shown in Sec. 3.1.3.

Associating actions to mentor states We begin by associating actions to mentor states—that is, the process associates each state s_m in the mentor's extracted experiences, with a corresponding error and action estimate (l.$1-7$ in Alg. 2).

Specifically, for each observer transition tuple $\langle s_o, a_o, s'_o \rangle$, we compare the transition (sub-)tuple $\langle s_o, s'_o \rangle$ with each transition tuple $\langle s_m, s'_m \rangle$ in the mentor's experience, using as an error metric the Kullback–Leibler (KL) divergence for multivariate normal distribution over the state features. If the error computed is lower than the

error already assigned to this mentor state, we update the estimate of the mentor's action a_m at s_m to the a_o taken by the observer at s_o, while also updating its error value accordingly. This ensures that we have associated to each mentor state transition the most accurate—given our knowledge so far—action.

However, utilizing the above-mentioned technique with action estimates in a cold start fashion can reduce the mentor's influence on the observer's training. To tackle this problem, we proceed as follows. We have a dedicated *pre-demonstration phase* of random movement before training, devoted to initializing the action estimates with beneficial information. Subsequently, this information is continuously being updated with observer transition tuples $\langle s_o, a_o, s'_o \rangle$, while training online.

Finding similar states Thus, Alg. 2 has so far updated the actions and errors associated with each mentor state. Therefore, one could simply choose the s_m state most similar (in terms of KL-divergence) to s_o, and pick the a_m action associated with s_m in order to fill in an "augmented" observer experience tuple with the "missing" mentor action. However, such an approach could favor specific states, resulting to early overfitting problems.

This calls for a heuristic that will provide a *set* S_m (with $|S_m| \geq 1$) of mentor states that are similar to the s_o observer state. The search for similar mentor states to s_o (denoted as *findSimilarStates(s_o)* in Alg. 1 and Alg. 2) is performed as follows. For each observer state s_o, we use *Nearest Neighbour* (KNN) [9] search on the mentor's dataset to find a set of the K-nearest neighbour mentor states. We then identify, among all neighbour candidates, the "most similar" to s_o (in terms of KL-divergence) mentor state s_m^{KL}. Additionally, we introduce an error threshold e_{th}^{KL}. For each remaining neighbor, if its KL-divergence from s_m^{KL} does not exceed e_{th}^{KL}, it is deemed similar and kept alongside the chosen mentor state s_m^{KL}. As such, *findSimilarStates(s_o)* returns a set of similar states S_m, which includes s_m^{KL} together with similar neighbors states. The state similarity problem we face has conceptual connections with work related to Exploration in Deep Reinforcement Learning [44]. There, authors introduce hashing to discretize large state spaces as a way to identify similar states, in order to enhance exploration in the environment.

Final mentor action selection After obtaining a set of similar mentor states S_m through the state similarity process (l.8 in Alg. 2), we can conclude to a final mentor action a_m. Via $getAssociatedActions(S_m)$ (l.9 in Alg. 2), we retrieve the associated mentor's actions estimates, which constructs the action set A_m, and then randomly choose a single action $a_m \in A_m$.

Due to the expertise of the mentor, it is expected during the early training stages that the inexperienced observer will tend to take different actions from the mentor action estimate ($a_o \neq a_m$) and therefore result in different next states (s'_o and s'_m could potentially be quite different even though the current states are 'similar'). We note that a_m is only perceived as the correct mentor action from the perspective of the observer, and does not necessarily correspond to the actual mentor action taken for this state transition. This is caused by the action association process definition, since action estimates may have high error values (i.e., when correlated with high KL-divergence mentor states).

Based on this a_m selection, we can directly extract the related state transition $\langle s_m, s'_m \rangle$ from the mentor's dataset. We now have complete information for an "augmented" observer experience tuple, that contains additional information regarding the estimated transition of the mentor: $\langle s_o, a_o, s'_o, r_o, s_m, a_m, s'_m \rangle$. This tuple is stored in the replay memory and will be used for computing values of the "augmented" loss functions below, to appropriately optimize the network parameters.

Algorithm 2 Mentor action prediction

Input: Observer transition $\langle s_o, a_o, s'_o \rangle$, Mentor extracted $experiences$
Output: Predicted mentor action a_m

1: **for** $\langle s_m, s'_m \rangle$ in $experiences$ **do**
2: err = KLdivergence($\langle s_o, s'_o \rangle, \langle s_m, s'_m \rangle$)
3: **if** $err <$ getAssociatedError(s_m) **then**
4: setAssociatedError(s_m, err)
5: setAssociatedAction(s_m, a_o)
6: **end if**
7: **end for**
8: S_m = findSimilarStates(s_o)
9: A_m = getAssociatedActions(S_m)
10: a_m = random(A_m)
11: **return** a_m

3.1.3 Augmented Loss Functions

By predicting the mentor's action, the observer now has all components of its "augmented" experience tuple in place. Thus, it is able in principle to estimate the Q action-value function from both perspectives, the mentor's and its own. Of course, to estimate these in a DRL fashion, one needs to equip the Q-approximating network with loss functions that augment the default one to incorporate both perspectives. We create such *augmented* loss functions for both the mentor and observer, from the observer's perspective. We use a common Q network, where the observer augmented loss function is constructed as:

$$L(\theta_t)_o = E_{s,a,r,s'}[(r_o + \gamma Q(s'_o, \underset{a'}{\operatorname{argmax}}\, Q(s'_o, a'; \theta_t^-); \theta_t^-) - Q(s_o, a_o; \theta_t))^2] \quad (2)$$

while the mentor augmented loss function as:

$$L(\theta_t)_m = E_{s,a,r,s'}[(r_o + \gamma Q(s'_m, \underset{a'}{\operatorname{argmax}}\, Q(s'_m, a'; \theta_t^-); \theta_t^-) - Q(s_m, a_m; \theta_t))^2] \quad (3)$$

Having two different loss functions creates the need to select the best one for the model. Intuitively, small deviations for Q values on subsequent states translates into a more stable model. We thus use Q-values divergence as the deciding factor on which augmented loss function we use. For each time step we store the value $Q_{past} = Q(s, a; \theta_t)$, where $s = s_o, a = a_o$ if $L(\theta_t)_o$ was chosen, and $s = s_m, a = a_m$ otherwise. Now, we can compare the immediately previous Q-function values with present ones. Specifically, we calculate two square distance terms at each step:

$$D_o = ||Q(s_o, a_o; \theta_t) - Q_{past}||^2 \quad (4)$$

$$D_m = ||Q(s_m, a_m; \theta_t) - Q_{past}||^2 \quad (5)$$

We select the augmented loss function associated with the minimum distance: the observer uses $L(\theta_t)_o$ if $D_o < D_m$, and $L(\theta_t)_m$ otherwise. The selected augmented loss function will be used to optimize the network.

3.2 Improving the DIIQN model

We further optimize our algorithm by integrating three well-known DQN extensions in our model, specifically *DDQN*, *PER* and *DNA* (see Sec. 2) and view these as algorithmic components that potentially enhance our DIIQN model. The DNA and PER algorithms do not require any changes for their implementation as our algorithm does not directly affect the network's architecture or experience replay sampling. However, for the remaining one, specific modifications of the original algorithm are required.

3.2.1 DDQN on Deep Implicit Imitation

As with non-augmented DQN-based loss functions, overestimation error can be improved by DDQN in our setting also. Specifically, we can update the estimate of the target value according to DDQN for both mentor and observer respectively. As such, the augmented loss functions with DDQN are the following:

$$L(\theta_t)_o = E_{s,a,r,s'}[(r_o + \gamma Q(s'_o, \underset{\boldsymbol{a'}}{\mathbf{argmax}}\, \boldsymbol{Q(s'_o, a'; \theta_t)}; \theta_t^-) - Q(s_o, a_o; \theta_t))^2] \quad (6)$$

$$L(\theta_t)_m = E_{s,a,r,s'}[(r_o + \gamma Q(s'_m, \underset{\boldsymbol{a'}}{\mathbf{argmax}}\, \boldsymbol{Q(s'_m, a'; \theta_t)}; \theta_t^-) - Q(s_m, a_m; \theta_t))^2] \quad (7)$$

3.2.2 Algorithmic Components' Selection

We performed extensive testing of various combinations of these algorithmic components (DDQN, PER, DNA) and our initial DIIQN model, evaluating their performance in well-known DRL testbed domains—specifically, *2D Maze* and *Cart-pole*. This systematic initial experimentation led us to the conclusion that the most beneficial choice is using all those components in our final ("extended") DIIQN model. In material supplementary to this paper[3] we include relevant results on these DRL testbed domains, where we exhibit the benefit of including all these algorithmic components on the DIIQN model. The algorithmic steps for training our final DIIQN model are summarized in Fig. 1. We now proceed to present the experimental evaluation of our final DIIQN model (henceforth referred to simply as "DIIQN model" or "DIIQN agent").

4 Experimental Evaluation

We now present in detail a systematic experimental evaluation of our deep implicit imitation approach within a highly complex and challenging experimental setting, namely the autonomous driving in a lane-free traffic simulation environment. In the supplementary material, we first provide a full list of the hyperparameters used in the lane-free traffic domain, paired with their corresponding values for these experiments. Additionally, common DRL testbed environments are also included, which showcase the performance of our algorithm in low complexity environments.

[3] The supplementary material, which also includes a link to a remote git repository with our DIIQN implementation, can be found at: https://bit.ly/3Kco5jZ

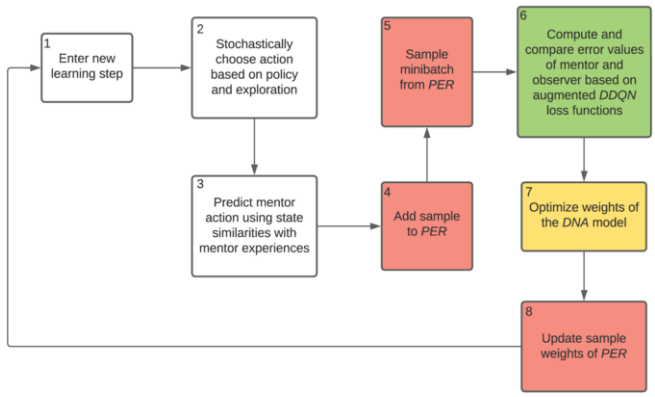

Figure 1. DIIQN training step flowchart. Step(s) labeled as: **6** is part of *DDQN*, **4-5-8** are part of *PER*, **7** is part of *DNA*

4.1 Experimental Setup

The lane-free traffic paradigm implies that no inner lanes -within the same direction- restrict the movement of the vehicles. The vehicles are thus able to utilize the full width of the road, thus this paradigm is inherently able to resolve recurrent traffic congestion problems. Our simulations utilize the lane-free traffic simulator TrafficFluidSim [48] which is an extension of the well-known Simulation of Urban MObility (SUMO) tool [24]. In our scenario, we have m vehicles that populate a highway, each with its own target cruising speed—namely, a *desired speed* v_d corresponding to a random value between $[v_d^{low}, v_d^{high}]$. The parameter choices[4] constitute a demanding learning environment, where our agent needs to consistently and cautiously operate alongside nearby traffic, and learn to overtake when needed. This highway emulates ring-road behaviour, in the sense that vehicles at the end of the road reappear at the beginning. This is a discrete-time domain, with the time-step set to $dt = 0.25\,sec$; while each episode consists of 1000 time-steps.

The state space should provide sufficient information to properly describe the agent's position on the road, as well as its interaction with neighbour vehicles. Thus, ours is a continuous state space, allowing for the following observed state features. Regarding the agent, at each time step we observe: its *lateral position* on the road y, its *speed* as a two dimension vector containing both its longitudinal and lateral speed v_x, v_y, and its longitudinal *desired speed* v_d. Regarding the neighbouring vehicles, for each neighbour we store information concerning: the longitudinal and lateral distances between our agent and the neighbour dx, dy, the longitudinal and lateral speed divergence between the aforementioned vehicles dv_x, dv_y. We observe the n closest neighbours with respect to a threshold distance parameter, regulating our longitudinal field of vision up to a certain length. Our state has a fixed vector size, therefore we always have n neighbours in the state information. In case our agent actually observes fewer neighbours within its field of view, we augment our state with virtual neighbours at the furthest possible distance with respect to our agent. Figure 2 shows an example of our state representation.

Each simulated vehicle is controlled by two continuous acceleration values, the longitudinal and the lateral acceleration a_x, a_y. We follow the action space implementation of [49], increasing the action

[4] Specifically, we populate a $2km$ highway with 100 vehicles, with desired speed range $[25, 35]m/s = [90, 126]km/hr$.

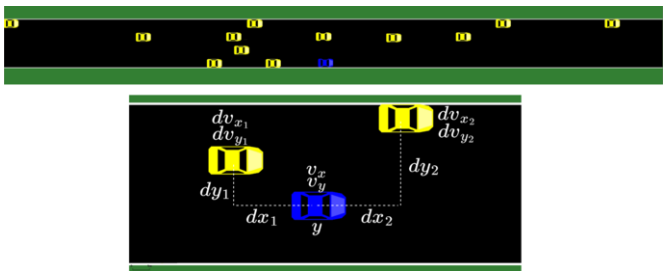

Figure 2. The Lane-Free traffic environment. Top: snapshot of the environment in TrafficFluid-Sim. Bottom: state information visualization.

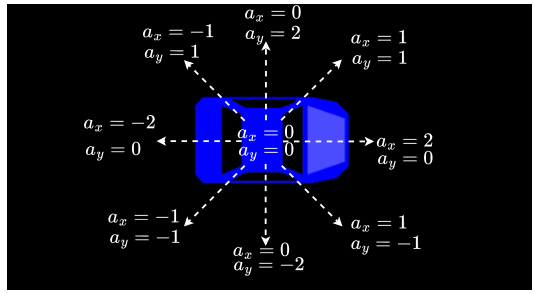

Figure 3. Lane-Free traffic: Action space, measurement unit $\mathbf{m/s^2}$

space from 5 to 9 discrete actions (see Figure 3). These (joint longitudinal and lateral acceleration) actions cover all main orthogonal and diagonal acceleration and deceleration directions, as well as the zero acceleration one.

Finally, we adopt the reward function put forward in [17]. The reward function introduced in their work is constructed for the lane-free traffic paradigm and it covers multiple objectives, namely, desired speed pursuit and collision avoidance. Starting with the maintenance of the desired speed v_d, using the current longitudinal speed v_x, we can calculate a normalized speed divergence:

$$c_x = \frac{|v_x - v_d|}{v_d} \tag{8}$$

aiming for zero values. Then, in order to integrate collision prevention into the reward function at each time step we provide a negative value of $c_{cl} = w_c \cdot l$, where l is the total number of collisions occurred during a time step and w_c is a weight magnifying the impact of each collision in the reward function. To provide a reward signal before a collision happens, we employ *potential fields* [18], calculating ellipsoid field utilities that can measure the threat of collision. Using longitudinal and lateral distances and speed divergences, two ellipsoid functions are calculated for each neighbour j, covering the critical and broad region. The critical region (E_c) is based on the distance between the vehicles, while the broad region (E_b) also considers speed deviations to predict potential danger. Higher returns are correlated with a higher chance of approaching collision. As such, a value $u_j = E_c(dx_j, dy_j) + E_b(dx_j, dy_j, dvx_j, dvy_j)$ is calculated for each neighbor vehicle in our state, and a combined value $c_f = \min\left\{\sum_j u_j, 1\right\}$ to be minimized is computed, and bounded to 1. For a detailed analysis, we refer to [17, 49]. Our final reward function is calculated as:

$$r_t = \frac{\epsilon}{\epsilon + w_x \cdot c_x + w_f \cdot c_f} + c_{cl} \tag{9}$$

where r_t is the reward on step t, w_x and w_f are weights empirically set to control the influence of c_x and c_f respectively, ϵ is a parameter that maximizes the returns when both c_x and c_f tend to 0.

Given the parameters, we can empirically set a lower bound to define a near optimal policy. In our case, a near optimal policy is characterized with no collisions ($c_{cl} = 0$) and a close pursuit of the vehicle's desired speed v_d ($c_x = 0$). Since the potential fields function c_f will always return a non-zero value as long as a neighbour vehicle is nearby, we assume a low value on average ($c_f = 0.1$) according to empirical investigation, and based on the hyperparameters ($\epsilon = 0.1$, $w_f = 0.65$), we empirically set the lower bound to 605. Thus, we consider the problem environment solved when the agent reaches a point when it is consistently collecting a reward of at least 605 per episode (i.e., a total of 1000 time-steps).

4.2 Experiments and Results

In our experiments within the lane-free traffic domain, the *other* vehicles follow the controller of a highly optimized, state-of-the-art movement control strategy for lane-free traffic, introduced in [33]. That method relies on the use of forces, which adjust the vehicles' behavior for collision avoidance. Additionally, due to its efficiency, we utilize the same controller as an "expert" mentor in our experiments. Note that convergence is reached at a reward of 605 per episode as explained above.

Fig. 4 depicts the performance of three different agents in terms of reward collected per episode over a 400 episode horizon (with all results averaged over 30 runs). First, we show the performance of our ("final") DIIQN approach, with its mentor being the aforementioned "expert" movement control strategy. Thus, we term this trainee agent as $DIIQN_{wExpM}$ (i.e., "DIIQN with an Expert Mentor"). Second, we examine the performance of a "baseline" simple DQN learning agent (enhanced with DDQN, PER, DNA), labeled as DQN_{noImit}. We also indicate in the figure the performance of our DIIQN approach when using as a mentor a "non-expert" *DQN agent*. This "non-expert" mentor shares the same model with the baseline DQN agent, but is only trained until its episodic reward is 500. We thus term this observer agent as $DIIQN_{wNonExpM}$ (i.e., "DIIQN with a Non-Expert Mentor"). This additional mentee agent is included so as to examine the behaviour of our approach when a sub-optimal mentor agent is used, aiming to highlight the capability of DIIQN to outperform such mentors. The highest reward achieved by the "non-expert" mentor is visible in Fig. 4 with a green dotted line, serving as a threshold for our $DIIQN_{wNonExpM}$ agent to reach and expectedly exceed. Given the importance of safety in this problem domain, we additionally report on the corresponding collision measurements in Fig. 5.

We first focus on the behaviour of the baseline DQN_{noImit} agent. Its early curve is steep, improving substantially during the first steps, and slowly converges to our empirically-set lower bound (605) for the reward, at 370 episodes. Studying the collisions' curve (Fig. 5), we can attribute the steepness to the corresponding decrease in collisions. The agent has a consistently small number of collision occurrences approximately from episode 70, and then attempts to improve upon the speed strategy, i.e., reach the desired speed without being involved in collisions. It is obvious that a combination of these two goals is a considerably more difficult objective, however convergence

is eventually achieved (in all cases). In intense traffic, when multiple slower vehicles in front of our agent, the desired speed objective can be hindered when overtaking is difficult or even impossible. Situations where the agent is reluctant to overtake (due to its surroundings) are quite common, thus fluctuations in reward can occur, even when near convergence.

We now discuss the learning curve of $DIIQN_{wExpM}$, the DIIQN agent using observations extracted from the optimized movement control strategy mentor. The same remarks regarding the curve steepness and fluctuations apply to this agent as well; however, it exceeds the lower bound of reward value 605 much earlier, at episode 120. Compared to the reward curve of the DQN_{noImit}, the learning acceleration improvement is immediately clear, requiring only a third of the DQN_{noImit}'s episodes. We also observe that DQN_{noImit} does not surpass a certain reward threshold (606), which is notably lower than the reward consistently gained by $DIIQN_{wExpM}$ (627) in the last episode.

Finally, we examine $DIIQN_{wNonExpM}$, which corresponds to an agent imitating the non-optimized mentor. As mentioned, the mentor was averaging rewards of 500 when its experiences were extracted. The learning curve converges at episode 180. It is apparent that the $DIIQN_{wNonExpM}$ outperforms its mentor, achieving performance comparable to the previously presented $DIIQN_{wExpM}$. This illustrates experimentally that implicit imitation is not upper-bounded by the mentor's performance. Naturally, since the mentor is of lower quality, observed learning speed is slower than that of $DIIQN_{wExpM}$, but still substantially faster than that of DQN_{noImit}.

By contrast, Fig. 5, which presents the total number of collisions per episode, indicates similar behaviour across all agents. This reflects the urgency of collisions avoidance that is embedded in our reward function. In the autonomous driving domain, safety of the drivers is imperative, and consequently we use reward functions that satisfy this requirement. Fluctuations in the collision graphs are to be expected, since our exploration parameter ϵ is non-zero and instances with collisions can still occur. However, in the last 50 episodes, during which exploration is disabled, we observe no collisions, demonstrating that all final models are collision-free upon convergence of the agent policy.

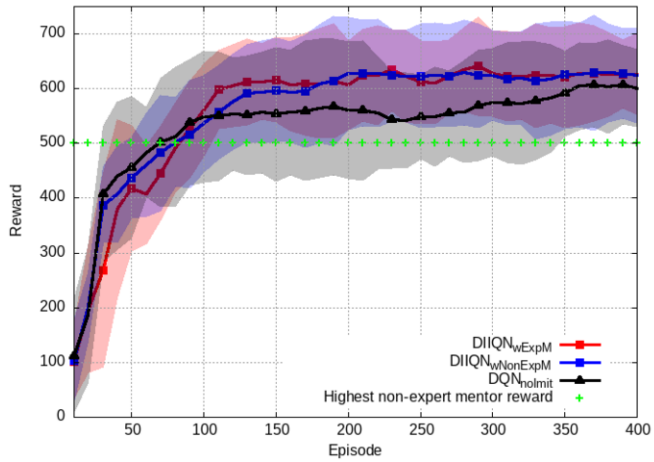

Figure 4. Lane-Free traffic: *DIIQN* reward (avg. over 30 runs)

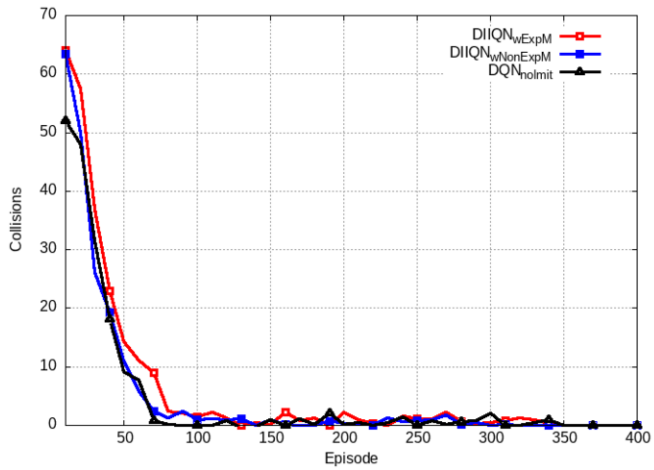

Figure 5. Lane-Free traffic: *DIIQN* collisions (avg. over 30 runs)

5 Conclusions and Future Work

This paper puts forward a novel model that enables, the incorporation of implicit imitation into *deep reinforcement learning*, thus allowing DRL methods to reap the benefits of implicit imitation. Our thorough experimentation on the novel and challenging lane-free autonomous driving domain (and also in common RL testbed environments, as reported in the supplementary material) clearly demonstrate the benefits of our approach. Specifically, deep implicit imitation was shown to accelerate significantly the DRL process, and to enable the "mentee" to outperform a non-optimal mentor.

Our novel deep implicit imitation model is so far built on the well-known DQN algorithm, however it can and should be extended to allow the integration of implicit imitation in alternative DRL methods. We intend to begin such extensions with the DDPG and PPO algorithms [21, 38], allowing for the integration of a continuous action space model in our approach. Specifically, for an on-policy method such as PPO, the adoption of our current approach would not be straightforward. PPO requires a trajectory of on-policy, consecutive state-action-reward pairs to estimate an advantage function (as the critic) at each training iteration. Moreover, we intend to study the performance differences of explicit imitation [45, 12] or inverse RL [7] techniques with our implicit imitation algorithm; and examine the influence of action information on performance and training acceleration when compared to an implicit imitation problem.

Future work focusing on the use of deep implicit imitation in the lane-free autonomous driving domain, includes the implementation of a DRL safety model. Theoretical approaches on safety boundaries for *lane-based* autonomous DRL [20, 39], could be extended and tested in lane-free traffic as well. Specifically, one could filter the actions taken by the agent based on safety boundaries, in order to achieve accelerated, imitation-based learning, while avoiding collisions altogether during training. More generally, it would be interesting to incorporate safety constraints in our implicit imitation methods. This can be key for enabling *safe reinforcement learning* [8] *with implicit imitation* in non-simulation, real-life environments.

Acknowledgements

The research leading to these results has received funding from the European Research Council under the European Union's Horizon 2020 Research and Innovation programme/ERC Grant Agreement n.[833915], project TrafficFluid.

References

[1] P. Abbeel and A. Y. Ng, 'Apprenticeship learning via inverse reinforcement learning', in *ICML*, (2004).
[2] J. Abramson et al. Imitating interactive intelligence, 2020.
[3] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, 'A survey of robot learning from demonstration', *Robotics and Autonomous Systems*, (2009).
[4] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, 'Deep reinforcement learning: A brief survey', *IEEE Signal Processing Magazine*, (2017).
[5] C. G. Atkeson and S. Schaal, 'Robot learning from demonstration', in *ICML*, (1997).
[6] A. D. Edwards, H. Sahni, Y. Schroecker, and C. L. Isbell, 'Imitating latent policies from observation', in *ICML '19*, (2019).
[7] J. Fu, K. Luo, and S. Levine, 'Learning robust rewards with adversarial inverse reinforcement learning', in *ICLR '17*, (2017).
[8] J. García, Fern, and o Fernández, 'A comprehensive survey on safe reinforcement learning', *Journal of ML Research*, (2015).
[9] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov, 'Neighbourhood components analysis', in *NIPS'04*, (2004).
[10] H. van Hasselt, 'Double q-learning', in *NIPS'10*, (2010).
[11] H. van Hasselt, A. Guez, and D. Silver, 'Deep reinforcement learning with double q-learning', in *AAAI'16*, (2016).
[12] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, G. Dulac-Arnold, J. Agapiou, J. Leibo, and A. Gruslys, 'Deep q-learning from demonstrations', in *AAAI*, (2018).
[13] J. Ho and S. Ermon, 'Generative adversarial imitation learning', in *NeurIPS*, (2016).
[14] I.-A. Hosu and T. Rebedea, 'Playing atari games with deep reinforcement learning and human checkpoint replay', in *Evaluating General Purpose AI (EGPAI 2016)*, (2016).
[15] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, 'Imitation learning: A survey of learning methods', *ACM Comput. Surv.*, **50**(2), (2017).
[16] B. Kang, Z. Jie, and J. Feng, 'Policy optimization with demonstrations', in *PMLR'18*, (2018).
[17] A. Karalakou, D. Troullinos, G. Chalkiadakis, and M. Papageorgiou, 'Deep reinforcement learning reward function design for autonomous driving in lane-free traffic', *Systems*, **11**(3), (2023).
[18] O. Khatib, *Real-Time Obstacle Avoidance for Manipulators and Mobile Robots*, 396–404, Springer New York, New York, NY, 1990.
[19] H. M. Le, N. Jiang, A. Agarwal, M. Dudík, Y. Yue, and H. Daumé III, 'Hierarchical imitation and reinforcement learning', in *ICML'18*, (2018).
[20] N. Li, A. Girard, and I. Kolmanovsky, 'Chance-constrained controller state and reference governor', *Automatica*, (2021).
[21] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, 'Continuous control with deep reinforcement learning', in *ICLR*, (2016).
[22] L.-J. Lin, 'Self-improving reactive agents based on reinforcement learning, planning and teaching', *Mach. Learn.*, **8**(3–4), (1992).
[23] F. Liu, Z. Ling, T. Mu, and H. Su. State alignment-based imitation learning, 2019.
[24] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wiessner, 'Microscopic traffic simulation using sumo', in *ITSC'18*, (2018).
[25] A. R. Mahmood, H. P van Hasselt, and R. S Sutton, 'Weighted importance sampling for off-policy learning with linear function approximation', in *NIPS'16*, (2016).
[26] M. Malekzadeh, I. Papamichail, M. Papageorgiou, and K. Bogenberger, 'Optimal internal boundary control of lane-free automated vehicle traffic', *Transportation Research Part C: Emerging Technologies*, (2021).
[27] M. J. Mataric, 'Using communication to reduce locality in distributed multiagent learning', *Journal of Experimental & Theoretical Artificial Intelligence*, **10**(3), 357–369, (1998).
[28] J. Merel, Y. Tassa, D. TB, S. Srinivasan, J. Lemmon, Z. Wang, G. Wayne, and N. Heess. Learning human behaviors from motion capture by adversarial imitation, 2017.
[29] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, 'Human-level control through deep reinforcement learning', *Nature*, **518**(7540), 529–533, (2015).
[30] A. Nair, D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, and S. Levine. Combining self-supervised learning and imitation for vision-based rope manipulation, 2017.
[31] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel. Overcoming exploration in reinforcement learning with demonstrations, 2017.
[32] A. Y. Ng and S. J. Russell, 'Algorithms for inverse reinforcement learning', in *ICML '00*, (2000).
[33] M. Papageorgiou, K.-S. Mountakis, I. Karafyllis, I. Papamichail, and Y. Wang, 'Lane-free artificial-fluid concept for vehicular traffic', *Proc. of the IEEE*, **109**(2), 114–121, (2021).
[34] S. D. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. H. Eng, D. Rus, and M. H. Ang, 'Perception, planning, control, and coordination for autonomous vehicles', *Machines*, **5**(1), (2017).
[35] B. Price and C. Boutilier, 'Implicit imitation in multiagent reinforcement learning', in *ICML*, pp. 325–334, (1999).
[36] B. Price and C. Boutilier, 'Accelerating reinforcement learning through implicit imitation', *Journal of Artificial Intelligence Research*, (2003).
[37] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, 'Prioritized experience replay', in *ICLR '04*, (2004).
[38] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.
[39] S. Shalev-Shwartz, S. Shammah, and A. Shashua. On a formal model of safe and scalable self-driving cars, 2017.
[40] D. Silver et al., 'Mastering the game of go with deep neural networks and tree search', *Nature*, (2016).
[41] J. E. Smith and R. L. Winkler, 'The optimizer's curse: Skepticism and postdecision surprise in decision analysis', *Management Science*, **52**, 311–322, (2006).
[42] B. C. Stadie, P. Abbeel, and I. Sutskever, 'Third-person imitation learning', in *International Conference on Learning Representations (ICLR 2017)*, (2017).
[43] R. S Sutton and A. G Barto, *Reinforcement learning: An introduction*, MIT press, 2018.
[44] H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. #exploration: A study of count-based exploration for deep reinforcement learning, 2016.
[45] F. Torabi, G. Warnell, and P. Stone. Behavioral cloning from observation, 2018.
[46] F. Torabi, G. Warnell, and P. Stone, 'Adversarial imitation learning from state-only demonstrations', in *AAMAS '19*, (2019).
[47] F. Torabi, G. Warnell, and P. Stone, 'Recent advances in imitation learning from observation', in *IJCAI'19*, (2019).
[48] D. Troullinos, G. Chalkiadakis, D. Manolis, I. Papamichail, and M. Papageorgiou, 'Lane-free microscopic simulation for connected and automated vehicles', in *ITSC'21*, pp. 3292–3299, (2021).
[49] D. Troullinos, G. Chalkiadakis, I. Papamichail, and M. Papageorgiou, 'Collaborative multiagent decision making for lane-free autonomous driving', in *AAMAS '21*, (2021).
[50] D. Troullinos, G. Chalkiadakis, V. Samoladas, and M. Papageorgiou, 'Max-sum with quadtrees for decentralized coordination in continuous domains', in *IJCAI '22*, (2022).
[51] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards, 2017.
[52] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, 'Dueling network architectures for deep reinforcement learning', in *ICML'16*, (2016).
[53] C. J. C. H. Watkins and P. Dayan, 'Q-learning', *Machine Learning*, **8**(3), 279–292, (May 1992).
[54] V. K. Yanumula, P. Typaldos, D. Troullinos, M. Malekzadeh, I. Papamichail, and M. Papageorgiou, 'Optimal path planning for connected and automated vehicles in lane-free traffic', in *ITSC'21*, (2021).