Robotics and Autonomous Systems 131 (2020) 103568

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Combining reinforcement learning with rule-based controllers for transparent and general decision-making in autonomous driving

Amarildo Likmeta [a,*], Alberto Maria Metelli [a], Andrea Tirinzoni [a], Riccardo Giol [a], Marcello Restelli [a], Danilo Romano [b]

[a] DEIB, Politecnico di Milano, Milan, Italy
[b] Marelli Europe S.p.A., Italy

ARTICLE INFO

Article history:
Available online 23 June 2020

Keywords:
Autonomous driving
Decision making
Interpretability
Reinforcement learning
Parameter-based exploration

ABSTRACT

The design of high-level decision-making systems is a topical problem in the field of autonomous driving. In this paper, we combine traditional rule-based strategies and reinforcement learning (RL) with the goal of achieving transparency and robustness. On the one hand, the use of handcrafted rule-based controllers allows for transparency, i.e., it is always possible to determine why a given decision was made, but they struggle to scale to complex driving scenarios, in which several objectives need to be considered. On the other hand, black-box RL approaches enable us to deal with more complex scenarios, but they are usually hardly interpretable. In this paper, we combine the best properties of these two worlds by designing parametric rule-based controllers, in which interpretable rules can be provided by domain experts and their parameters are learned via RL. After illustrating how to apply parameter-based RL methods (PGPE) to this setting, we present extensive numerical simulations in the highway and in two urban scenarios: intersection and roundabout. For each scenario, we show the formalization as an RL problem and we discuss the results of our approach in comparison with handcrafted rule-based controllers and black-box RL techniques.

1. Introduction

The development of the *autonomous vehicle* (AV) over the last decades has experienced a progressive increase in the automation level [1–5]. It is now established that the achievement of extensive *decision-making* capabilities is a fundamental element to accomplish full automation of vehicles. In the *autonomous driving* (AD) field, with decision maker, we refer to the software module that, taking as input a representation of the environment around the vehicle, selects a discrete high-level action (or *maneuver*) leading to an appropriate driving behavior. The decision-making module communicates its choice to the *motion planning* module that, given the environment representation and the current vehicle state, computes a spatial trajectory and a speed profile, satisfying the constraints of the dynamical model of the vehicle, in order to actuate the selected maneuver. Thus, decision-making in AD concerns with abstract and intentional actions, delegating their physical actuation to the lower-level modules. Typical examples of maneuver that the decision maker can select are: "follow the current path", "stop at a certain location", "change lane to right or left", and "yield to incoming vehicles".

Clearly, the decision maker has to be designed in order to output an "appropriate maneuver" in every situation. However, the notion of appropriateness is only intuitive, hard to formalize and, for this reason, it is typically obtained as an emerging property coming from the ensemble of different, possibly conflicting, objectives:

i. *Safety*. The maneuver selected by the decision maker has to satisfy certain safety requirements, avoiding dangerous situations or causing any collision/accident.
ii. *Comfort*. The selected maneuver has to take into account some performance indexes regarding the comfort perceived by passengers of the AV.

Moreover, the decision process itself should satisfy a set of properties such as:

iii. *Transparency*. The decision making execution process has to be traceable and reversible. This has to be assured in order to reconstruct the complete decision tree flow in investigating the motivations behind the selection of a given action.
iv. *Robustness*. Decision making has to be robust in terms of ability of handling all possible driving situations, avoiding corner cases or misbehavior in unexpected scenarios.

* Corresponding author.
E-mail address: amarildo.likmeta@polimi.it (A. Likmeta).

https://doi.org/10.1016/j.robot.2020.103568
0921-8890/

One of the main challenges of decision making in AD is to fulfill simultaneously these four objectives. The driving domain is highly unstructured, due to the almost arbitrary topology of the roads, and unpredictable in presence of other vehicles that might act in an adversarial way. Furthermore, given a driving situation there could exist multiple appropriate actions that satisfy the objectives stated above. Therefore, it is hard to find a unique rigorous solution to the AD problem. Another challenge consists in the sequential nature of the problem. The decision maker operates in closed-loop, since actions chosen in a given state affect next states in the near future. Therefore, the actions display a long term effect that the decision module has to take into account. For this reason, a greedy approach is not sufficiently far-sighted for capturing the sequential nature of the problem and could lead to heavily sub-optimal behaviors. Additionally, the nature of the driving environment requires to perform negotiation and to demonstrate intention awareness w.r.t. other driving participants [6]. In every situation in which different agents interact with each other, such as an intersection or a pedestrian crossing, conflicting goals appear and the decision making has to deal with them.

Early popular approaches to AD attempted to manually define the action to perform in specific situations by using a finite state machine or behavior trees [e.g., 7–10]. These solutions have been successfully applied by the vehicle BOSS, that won the DARPA Urban Challenge [DUC,7]. A similar technique can be found in *Junior*, the Stanford's proposal for the DUC [8]. Though pioneering, these approaches are unsuited for scaling to more realistic testing scenarios, whereby the major drawback is a substantial inability to understand the surrounding environment. Indeed, in a *rule-based decision maker* it is hard to design universal hand-crafted rules, able to deal with the complexity of all possible cases, accounting for uncertainty and other vehicles' intentions. This results in a lack of robustness and generality. Nevertheless, the rule-based approach can easily match two of the objectives stated above. Safety can be achieved since the answers of the system reacting to predefined events are hand-coded and they can be designed to match the desired level of safety. Furthermore, transparency is granted since it is immediate to traverse the decision tree and appreciate how variables are evaluated by the algorithm to output the prescribed action.

In recent years, a vast amount of work has been developed for addressing the decision-making problem in the framework of *machine learning* [11], also for overcoming the limitations of hand-crafted controllers [e.g., 12–14]. Learning-based methods represent a promising direction thanks to their ability to generalize on unseen situations, possibly producing more robust controllers [15]. Furthermore, they rely less on hand-coded features and more on interaction with the environment. However, the majority of these approaches disregard transparency and explainability. *End-to-end driving* [16], in which a neural network maps observations into low-level control actions (steering, throttle and braking), is probably the most emblematic example. Clearly, given the complexity of the network, no insights or considerations on the motivations behind the choices made by the algorithm are possible. These controllers, mapping observations to actions, can be trained by means of *reinforcement learning* [RL,17]. In RL the optimal control problem is framed into a learning problem in which an agent has to devise a mapping from observations to actions, called *policy*, interacting with the environment in a trial and error fashion. The learning process is guided by a *reward function* that, intuitively, specifies how well the agent is performing locally. Controllers learned via RL are more robust since their reasoning mechanism is automatically built and, therefore, can generalize to unexpected situations. Nevertheless, the interpretability of the resulting controller heavily depends on the model employed to map observations to actions (e.g., neural networks).

The goal of this paper is to make a step towards the simultaneous achievement of the four objective listed above. We propose an approach that combines rule-based controllers and reinforcement learning, incorporating the strength points of both methods. Specifically, we aim at preserving the safety and transparency properties of the hand-crafted rule-based controllers, while enhancing them with the generalization capabilities of RL. For this purpose, we design a *parametric rule-based policy*, i.e., a rule-based controller in which the rules are defined in terms of a set of *parameters*, whose values are not manually set, but learned by interacting with the environment using an RL algorithm. Our framework is general and the execution process can be employed in a variety of AD scenarios.

The paper is organized as follows. We start by providing a basic background about sequential decision-making problems and RL (Section 2). Then, we present an overview of the state of art of RL for AD (Section 3). In Section 4, we discuss how to employ RL to learn the parameters of the rule-based policy, illustrating a specific class of algorithms employed for this purpose. The subsequent sections are meant to present the AD scenarios we face (Section 5), how to model them as sequential decision-making problem (Section 6), and how to design a suitable rule-based controller (Section 7). Finally, Section 8 is devoted to the presentation of the numerical simulation results.

2. Preliminaries

In this section, we provide the basics of reinforcement learning [RL,17] (Section 2.1) and briefly review the state-of-the-art RL algorithms which are commonly employed in AD applications (Section 2.2), with particular attention on the deep RL methods (Section 2.3). For a thorough overview of RL we refer the reader to Sutton and Barto's book [17].

2.1. Markov decision processes

The interaction between the agent and the environment is typically modeled under the *Markov decision process* [MDP,18] framework. An MDP can be formalized as tuple $\mathcal{M} = \langle \mathcal{X}, \mathcal{U}, p, r, \mu, \gamma \rangle$, where $\mathcal{X}$ is the state space, $\mathcal{U}$ is the action space, $p(\cdot|\boldsymbol{x}, \boldsymbol{u})$ is the transition model providing the distribution of the next state given that action $\boldsymbol{u}$ is taken in state $\boldsymbol{x}$, $r : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \to \mathbb{R}$ is the reward function, μ is the initial-state distribution, and $\gamma \in [0, 1]$ is the discount factor. A deterministic *control policy* $\pi : \mathcal{X} \to \mathcal{U}$ is a mapping from states to actions. At the beginning of each episode of interaction, the initial state $\boldsymbol{x}_0$ is drawn from μ. Then, the agent chooses an action $\boldsymbol{u}_0 = \pi(\boldsymbol{x}_0)$ according to its policy π, it transitions to the next state $\boldsymbol{x}_1$, which is chosen by the environment according to $p(\cdot|\boldsymbol{x}_0, \boldsymbol{u}_0)$, and it receives a reward $r(\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{x}_1)$. The process is then repeated for H steps, where H is called *horizon* of the task. The result of this repeated interaction is called *episode* and denoted as $\tau = \langle \boldsymbol{x}_0, \pi(\boldsymbol{x}_0), \ldots, \boldsymbol{x}_{H-1}, \pi(\boldsymbol{x}_{H-1}), \boldsymbol{x}_H \rangle$. The goal of the agent is to find the policy π^* maximizing the *discounted expected return*:

$$J^{\pi} = \mathop{\mathbb{E}}_{\substack{\boldsymbol{x}_0 \sim \mu \\ \boldsymbol{x}_{t+1} \sim p(\cdot|\boldsymbol{x}_t, \pi(\boldsymbol{x}_t))}} \left[\sum_{t=0}^{H-1} \gamma^t r(\boldsymbol{x}_t, \pi(\boldsymbol{x}_t), \boldsymbol{x}_{t+1}) \right]. \tag{1}$$

To this end, we define the state–action value function $Q^{\pi}(\boldsymbol{x}, \boldsymbol{u})$ as the expected return obtained by taking action $\boldsymbol{u}$ in state $\boldsymbol{x}$ and following the policy π thereafter. Similarly, we define the optimal state–action value function $Q^*(\boldsymbol{x}, \boldsymbol{u})$ as the expected return obtained by taking action $\boldsymbol{u}$ in state $\boldsymbol{x}$ and following an optimal policy thereafter. Then, an *optimal policy* π^* is a policy that is greedy w.r.t. the optimal value function, i.e., $\pi^*(\boldsymbol{x}) \in \arg\max_{\boldsymbol{u} \in \mathcal{U}} Q^*(\boldsymbol{x}, \boldsymbol{u})$ for all states $\boldsymbol{x}$.

2.2. Reinforcement learning

Many algorithms have been proposed in the literature to find the optimal policy when the MDP model is known (i.e., the agent has access to the reward and transition models). Dynamic programming approaches such as policy iteration and value iteration [e.g., 18] are well-known examples. Unfortunately, the knowledge of the environment dynamics is hardly available in practice. For instance, in AD applications one does not perfectly know the effects of all actions unless a very precise model of the car and of its surrounding environment is given. RL addresses this issue by providing algorithms to learn solely by *experience*, i.e., by interacting with the environment, without requiring any model of the underlying MDP. More precisely, the agent is allowed to try different actions and receives local feedback from the environment (the immediate reward and next state). As the learning process goes on, the agent figures out which actions are beneficial to maximize the expected return and starts choosing those more frequently, until convergence to an optimal behavior.

According to the general taxonomy [17], RL algorithms can be *model-based*, if they use experience to build an approximate model of the environment from which an optimal policy can be computed, or *model-free*, if they directly compute the optimal policy from the collected samples. Furthermore, RL algorithms are classified into *value-based*, *policy-based*, and *actor–critic*. Value-based algorithms focus on estimating the optimal value function, from which the optimal policy can be easily derived. Policy-based algorithms, on the other hand, directly search in the space of possible policies for the one maximizing the expected return. Finally, actor–critic methods combine the evaluation functionalities of value-based algorithms with the search strategies of policy-based methods to provide more general approaches.

2.3. Deep reinforcement learning

Since real-world problems often involve high-dimensional and continuous state–action spaces, state-of-the-art RL algorithms consider parametrized policies and/or value functions and directly learn in the space of possible parameters. Deep neural networks represent the most common approximator, which has led to the recent field of *deep reinforcement learning* [19,20]. Deep RL algorithms have obtained impressive results in a variety of domains, including video-games [21,22], robotics [23–26], and, as we shall discuss later, autonomous driving. Deep Q networks [DQNs,22] constitute one of the most common approaches in this context. A DQN is a value-based method that approximates the optimal state–action value function Q^* using a deep neural network. This state–action value function is learned by minimizing an approximation loss via gradient descent, by relying solely on the experience collected online using a policy in charge of enforcing exploration (e.g., ϵ-greedy policy or Boltzmann distribution). For instance, in the case of ϵ-greedy exploration, given the current Q function, the policy chooses the corresponding optimal action with probability $1 - \epsilon$ and a random action with probability ϵ. When the exploration rate ϵ is decayed at a suitable rate, the algorithm eventually converges to optimal behavior. Compared to standard Q-learning with function approximation, DQN introduces new techniques to stabilize and speed up the learning process when the function approximator is a neural network, such as experience replay memory [27] and the concept of target network [22]. A variety of modifications to further improve the learning process of the DQN have been proposed, including the Double DQN [28], the dueling architecture [29], and several others [e.g., 30–32]. DQNs are typically well-suited for finite action spaces. For continuous control problems, policy-based or actor–critic algorithms are more common [e.g., 24,33–36].

3. Reinforcement learning for autonomous driving

In the recent years, there has been a growing interest in applying RL techniques to AD and several methodologies have been proposed that show promising results. *End-to-end* approaches are a well-known demonstration of the capabilities of RL algorithms. Here the goal is to learn a direct mapping between raw inputs (e.g., sensors or cameras) and controls. For instance, several works have shown that it is possible to learn driving policies from pixels in a variety of AD scenarios, including car racing [24,37,38], lane following [39], and the most general case of urban driving [40,41]. Standard sensor data, such as distances and velocities, have been used as an alternative to images [14,42–44]. Since end-to-end algorithms typically require an enormous amount of experience when learning from scratch, several works apply imitation learning to speed up the training process by leveraging expert demonstrations [e.g., 40,45–50]. Due to the adoption of deep neural networks, one of the key advantages of end-to-end approaches is the ability to extract relevant features from the raw inputs, which removes the burden from the system designer. On the negative side, this makes the resulting policies poorly interpretable and non-safe. Thus, these algorithms are typically tested on simulated domains, though there have been some attempts to deploy the resulting policies to the real world [41].

In order to achieve the desired objectives for an AD system (the ones mentioned in the Introduction) and facilitate deployment to real vehicles, a large body of literature applies RL to high-level decision-making problems. In this setting, the decision maker has access to a high-level representation of the world (e.g., features or occupancy grids rather than raw sensor data), which is built by other modules in the AD stack. Due to the simplicity in modeling the environment, highway driving is perhaps the most common task in these settings and has been widely adopted as a testbed for RL and imitation algorithms [51,52]. For such problems, in [12] a method for learning lane change decisions is proposed. A DQN is used to choose among five discrete actions given as input a high-level occupancy grid combined with some sensor readings. In [53] several components are studied to make RL algorithms more suitable to tactical decision making. In [13], the authors model the behavior of all agents involved and propose a method that reacts to the actions of the other drivers. The latter is tested on both simulated and real highway data. Similarly, in [54] relevant issues in AD settings are addressed, such as safety and dealing with multiple agents. Other works consider the more complicated problem of crossing intersections. Here the agent faces several complications, such has handling partial observability and negotiations. In [55] the former complication is addressed by designing a suitable set of actions. Finally, in [56] RL is combined with tree search to learn complex task plans.

Despite the impressive results, the application of RL algorithms to real-world autonomous driving problems still suffers several complications. Ensuring safety of the learned policies is perhaps the most relevant and well-known difficulty. Some recent solutions have been proposed to overcome this limitation, typically involving the inclusion of constraints [57] or prior knowledge [58] into the learning process. Another relevant problem is the high sample complexity required by RL agents to learn complicated tasks (such as most of the AD ones) from scratch. Similarly to the safety issue, the most common solution is to leverage prior knowledge. The latter can be either from human experts, e.g., driving demonstrations, or "artificial" from previously-learned tasks. Both cases fall under the transfer RL setting, where the goal is to build algorithms capable of reusing past knowledge to speed-up the learning process of new tasks. This is perhaps one of the most promising directions to scale RL methods to real-world AD applications and several works

have been recently proposed in this direction [e.g., 41,59–63]. In addition to these limitations, policies learned by (deep) RL approaches are typically very hard to interpret, which further complicates their deployment to real vehicles [64,65].

4. Reinforcement learning with parametric rule-based policies

In this section, we illustrate how to employ RL techniques to learn the parameters of rule-based policies. We start by framing the problem of learning a controller of an AD system as an RL problem (Section 4.1). Then, we present the *Policy Gradients with Parameter-based Exploration* [PGPE,66], a class of policy-based RL algorithms suited for our setting (Section 4.2).

4.1. Problem definition

We denote a parametric rule based policy as $\pi_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{U}$, i.e., a function taking a state of the environment $\boldsymbol{x}$ as input, producing a control action $\boldsymbol{u} = \pi_{\boldsymbol{\theta}}(\boldsymbol{x})$ and parametrized by $\boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^d$. Thus, $\boldsymbol{\theta}$ is a d-dimensional vector belonging to a subset of $\mathbb{R}^d$, denoted by Θ, the parameter space. Recall from Section 2 that the goal of an RL agent consist in finding the *optimal policy*, i.e., the policy maximizing the expected return, as defined in Eq. (1). When we deal with parametric policies $\pi_{\boldsymbol{\theta}}$, denoting for conciseness $J(\boldsymbol{\theta}) = J^{\pi_{\boldsymbol{\theta}}}$, the goal becomes to find the best parameter in the parameter space Θ, maximizing the expected return:

$$\max_{\boldsymbol{\theta} \in \Theta} J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\substack{\boldsymbol{x}_0 \sim \mu \\ \boldsymbol{x}_{t+1} \sim p(\cdot|\boldsymbol{x}_t, \pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t))}} \left[\sum_{t=0}^{H-1} \gamma^t r(\boldsymbol{x}_t, \pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t), \boldsymbol{x}_{t+1}) \right]. \tag{2}$$

As mentioned in Section 2.2, having access to the full vehicle model and exploiting the physics of the vehicle-environment interaction we could, in principle, address this problem using dynamic programming [18]. However, this approach would require either to exploit complex models with notable computational effort or make use of simplified versions, getting likely sub-optimal solutions. In the RL framework, no environment model is requested as the expectation is estimated through Monte Carlo simulation. In other words, we interact with the environment executing policy $\pi_{\boldsymbol{\theta}}$, collecting N independent episodes $\{\tau_i\}_{i=1}^N$ and replacing the expectation with the corresponding sample mean, leading to the estimator of the expected return:

$$\widehat{J}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H-1} \gamma^t r(\boldsymbol{x}_{i,t}, \pi_{\boldsymbol{\theta}}(\boldsymbol{x}_{i,t}), \boldsymbol{x}_{i,t+1}). \tag{3}$$

It is worth noting that the optimization of such objective requires to test multiple values of the parameters $\boldsymbol{\theta}$ in order to figure out which is the best one. In other words, each evaluation of the objective implies an interaction with the environment with a different and possibly bad parametrization that might make the vehicle behave in an undesired, possibly dangerous, way. For this reason, the standard RL approach can be freely applied in simulation, while requires more care in the deployment in the real-world scenario [67]. This represents an instance of the well-known *exploration–exploitation* dilemma [17].

4.2. Parameter-based policy optimization

The optimization of the objective function at Eq. (3) can be carried out using essentially two class of policy-based methods: *action-based* policy optimization and *parameter-based* policy optimization. Action-based methods, also named *policy gradient methods*, update the parameters $\boldsymbol{\theta}$ by following the improving direction of the gradient [68,69]:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \widehat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \tag{4}$$

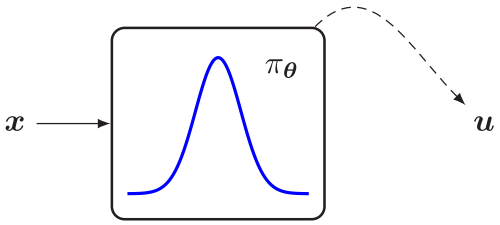


Fig. 1. Graphical representation of action-based methods.

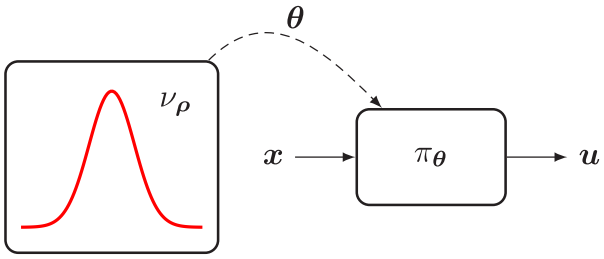


Fig. 2. Graphical representation of parameter-based methods.

where $\alpha > 0$ is a learning rate and $\widehat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is a Monte Carlo estimator of the policy gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, i.e., the gradient of the expected return $J(\boldsymbol{\theta})$, obtained with a set of episodes collected in the environment $\{\tau_i\}_{i=1}^N$ (Fig. 1). These methods require that the policy $\pi_{\boldsymbol{\theta}}$ is differentiable w.r.t. to the parameters $\boldsymbol{\theta}$, which is not the typical case when considering rule-based policies. Furthermore, they require the policy to be stochastic in order to enforce a sufficient degree of exploration. Stochastic policies are highly undesirable in the AD scenario since they prevent enforcing traceability of the decision making process.[1] For a broader view of the policy gradient methods refer to the survey [70].

For these reasons, we resort to parameter-based policy optimization methods, also named *policy gradient with parameter-based exploration* [PGPE,66]. These approaches allow for non-differentiable and deterministic policies, as they move the exploration problem to a higher level. Instead of having a stochastic policy at the action level, exploration is moved to the parameter level. The parameters $\boldsymbol{\theta}$ are sampled from a *hyperpolicy* $\nu_{\boldsymbol{\rho}}$ depending itself on a parameter vector $\boldsymbol{\rho} \in \mathcal{P} \subseteq \mathbb{R}^p$. It is required that $\nu_{\boldsymbol{\rho}}$ is stochastic and differentiable w.r.t. $\boldsymbol{\rho}$. Thus, exploration is carried out by testing different parameters sampled from $\nu_{\boldsymbol{\rho}}$. Consequently, the policy $\pi_{\boldsymbol{\theta}}$ can be chosen to be deterministic and non-differentiable.

Example. Consider learning a tree-based policy π_θ with a single parameter $\theta \in \mathbb{R}$. For instance, $\pi_\theta(\boldsymbol{x})$ might be a rule of the form "if $f(\boldsymbol{x}) > \theta$ then take action $\boldsymbol{u}_1$, else take action $\boldsymbol{u}_2$", for some function $f : \mathcal{X} \to \mathbb{R}$. Clearly, the mapping between states and actions is not necessarily differentiable with respect to θ and direct optimization of the expected return as a function of θ is difficult. However, we could instead learn the parameters $\boldsymbol{\rho}$ of a distribution $\nu_{\boldsymbol{\rho}}$ over θ. For instance, we could choose $\boldsymbol{\rho} = (\mu, \sigma^2)$ and $\nu_{\boldsymbol{\rho}}(\cdot) = \mathcal{N}(\mu, \sigma^2)$, a Gaussian distribution with mean μ and variance σ^2. As we formally explain shortly, it turns out that the expected return is indeed differentiable as a function of μ and σ^2, and standard gradient-based techniques could be applied to learn these parameters. At the end of the learning process, the resulting distribution (i.e., its mean μ) should concentrate around the most performing parameter, and thus one can take π_μ as the final decision-tree.

[1] As we shall see in Section 7, rule-based policies can be seen as parametrized decision trees. Thus, they are not in general differentiable and surely not stochastic.

Table 1
Gradient for a d-variate Gaussian hyperpolicy with diagonal covariance $\nu_{\boldsymbol{\mu},\boldsymbol{\sigma}} = \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}))$, $i \in \{1, \dots, d\}$.

$\nabla_{\mu_i} \log \nu_{\boldsymbol{\mu},\boldsymbol{\sigma}}(\boldsymbol{\theta})$	$\nabla_{\sigma_i} \log \nu_{\boldsymbol{\mu},\boldsymbol{\sigma}}(\boldsymbol{\theta})$
$\frac{\theta_i - \mu_i}{\sigma_i^2}$	$\frac{(\theta_i - \mu_i)^2 - \sigma_i^2}{\sigma_i^3}$

Algorithm. In this setting, it is convenient to redefine the expected return as a function of $\boldsymbol{\rho}$:

$$J(\boldsymbol{\rho}) = \mathop{\mathbb{E}}_{\boldsymbol{\theta}\sim\nu_{\boldsymbol{\rho}}}[J(\boldsymbol{\theta})] = \mathop{\mathbb{E}}_{\boldsymbol{\theta}\sim\nu_{\boldsymbol{\rho}}}\Bigg[\underbrace{\mathop{\mathbb{E}}_{\substack{\boldsymbol{x}_0\sim\mu \\ \boldsymbol{x}_{t+1}\sim p(\cdot|\boldsymbol{x}_t,\pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t))}}\Bigg[\sum_{t=0}^{H-1}\gamma^t r(\boldsymbol{x}_t, \pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t), \boldsymbol{x}_{t+1})\Bigg]}_{J(\boldsymbol{\theta})}\Bigg]. \tag{5}$$

The learning process is described in Algorithm 1 and proceeds as follows. A set of N parameters $\{\boldsymbol{\theta}_i\}_{i=1}^N$ are sampled independently from the hyperpolicy $\nu_{\boldsymbol{\rho}}$. For $i \in \{1, \dots, N\}$, we run the rule-based policy $\pi_{\boldsymbol{\theta}_i}$ in the environment, collecting M independent episodes $\{\tau_{ij}\}_{j=1}^M$ (Fig. 2). Finally, we employ the NM episodes to estimate the objective:

$$\widehat{J}(\boldsymbol{\rho}) = \frac{1}{N}\sum_{i=1}^N \widehat{J}(\boldsymbol{\theta}_i) = \frac{1}{N}\sum_{i=1}^N \underbrace{\frac{1}{M}\sum_{j=1}^M\sum_{t=0}^{H-1}\gamma^t r(\boldsymbol{x}_{ij,t}, \pi_{\boldsymbol{\theta}}(\boldsymbol{x}_{ij,t}), \boldsymbol{x}_{ij,t+1})}_{\widehat{J}(\boldsymbol{\theta}_i)}. \tag{6}$$

Now, since the hyperpolicy is stochastic and differentiable we can easily compute the gradient, exploiting the log-trick [66]: $\nabla_{\boldsymbol{\rho}} J(\boldsymbol{\rho}) = \mathbb{E}\left[\nabla_{\boldsymbol{\rho}} \log \nu_{\boldsymbol{\rho}}(\boldsymbol{\theta}) J(\boldsymbol{\theta})\right]$. Thus, at each iteration the hyperpolicy parameters are updated with a single step of gradient ascent:

$$\boldsymbol{\rho} \leftarrow \boldsymbol{\rho} + \alpha \widehat{\nabla}_{\boldsymbol{\rho}} J(\boldsymbol{\rho}), \qquad \widehat{\nabla}_{\boldsymbol{\rho}} J(\boldsymbol{\rho}) = \frac{1}{N}\sum_{i=1}^N \nabla_{\boldsymbol{\rho}} \log \nu_{\boldsymbol{\rho}}(\boldsymbol{\theta}_i)\widehat{J}(\boldsymbol{\theta}_i), \tag{7}$$

where $\alpha > 0$ is the learning rate and $\widehat{\nabla}_{\boldsymbol{\rho}} J(\boldsymbol{\rho})$ is the estimator of the gradient $\nabla_{\boldsymbol{\rho}} J(\boldsymbol{\rho})$ obtained with the collected episodes $\{\{\tau_{ij}\}_{j=1}^M\}_{i=1}^N$. It is common to select a Gaussian hyperpolicy with a diagonal covariance matrix $\nu_{\boldsymbol{\mu},\boldsymbol{\sigma}} = \mathcal{N}(\boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}))$, having, consequently, $\boldsymbol{\rho} = (\boldsymbol{\mu}, \boldsymbol{\sigma})$ as parameters. Refer to Table 1 for the expression of the gradients in such case.

One of the main drawbacks of PGPE is that we cannot directly reuse the same episode more than once to perform multiple gradient steps since the hyperpolicy parameters change as an effect of the update. Nevertheless, it is possible to employ *importance sampling* [71] techniques to estimate the gradient of the expected return w.r.t. to the hyperpolicy parametrization $\boldsymbol{\rho}'$ having samples collected with a different hyperpolicy parametrization $\boldsymbol{\rho}$. This allows performing multiple updates with the same collected episodes [72], possibly improving the sample complexity of the PGPE algorithm.

Algorithm 1 PGPE.

Input: N number of sampled policy parameters
M number of episodes per policy parameter
Ite number of iterations
$(\alpha^h)_{h=1}^{\mathrm{Ite}-1}$ learning rate schedule
Initialize the hyperpolicy parameters $\boldsymbol{\rho}^0$ arbitrarily
for $h = 0, 1, \dots, \mathrm{Ite} - 1$ **do**
Sample N policy parameters $\{\boldsymbol{\theta}_i^h\}_{i=1}^N$ independently from $\nu_{\boldsymbol{\rho}^h}$
Collect M trajectories $\{\tau_{i,j}^h\}_{j=1}^M$ independently for each $\{\pi_{\boldsymbol{\theta}_i^h}\}_{i=1}^N$
Update the hyperpolicy parameters $\boldsymbol{\rho}^{h+1} = \boldsymbol{\rho}^h + \alpha^h \widehat{\nabla}_{\boldsymbol{\rho}} J(\boldsymbol{\rho}^h)$
end for

5. Autonomous driving scenarios

In this work, we consider three autonomous driving scenarios that fall into the following areas: *highway* and *urban*. For the former area, we consider the *lane change* scenario along the highway where the goal is to drive the ego vehicle displaying a "natural" behavior, respecting the driving rules and safety constraints, while maximizing the speed. For the urban area, the scenarios faced are the *crossroads* scenario and the *roundabout* scenario (Fig. 3). In the crossroads scenario the goal is to drive the ego vehicle through the intersection, fulfilling the right of way, whereas in the roundabout the ego vehicle has to give way to the vehicles inside and then follow the roundabout until the exit. While the highway is a *continuing* task, i.e., the agent has to learn how to trade-off multiple conflicting objective (e.g., keeping a high speed, staying on the rightmost lane), the crossroads and the roundabout are *goal-based* task, i.e., the agent has to complete the intersection in the minimum amount of time possible.

Similarly to what happens in real autonomous vehicles, in our framework we assume the presence of a *low-level controller*, which assures the fulfillment of the safety distance with the other vehicles, managing the speed accordingly and maintaining the center of the lane. This allows us to focus on high-level decisions. Informally, in the lane change scenario, the decision consists in finding the right timing at which performing a lane-change on the right or on the left. Instead, in the crossroads case the decision consists in adjusting the velocity so that the intersection is completed in the minimum time possible, accounting for the presence of other vehicles.

In this section, we introduce the simulation tool we employed in our experimental campaign (Section 5.1) and we provide a more detailed description of the three scenarios we address (Sections 5.2–5.4). In the following, we will refer to the vehicle we control as *ego vehicle*.

5.1. The SUMO simulator

We employ SUMO simulator, an open source, highly portable, microscopic and continuous road traffic simulation package designed to handle large road networks [73]. SUMO focuses on the high-level control of the car, integrating an internal system that controls the vehicle dynamic. For all the driving tasks considered, we model in the simulator different scenarios of road topology and traffic intensities, randomizing the flow of vehicles, to ensure producing general driving policies, and avoid overfitting their behavior to a specific driving case. One parameter of interest of the simulator is the frequency of control, i.e., the frequency at which the control signal is issued. We set the control frequency to 10 Hz for all our experiments, meaning that we choose an action to perform every 100ms. Fig. 4 shows an instance of the graphical interface of the simulator. During the simulation, SUMO provides information about the other vehicles around the ego vehicle. More specifically, we can query SUMO for positions and velocities of all the cars in the simulation. This information is also available to the decision-making module in a real car, being provided by the sensing module.[2]

[2] Modeling the noise in the sensors is out of the scope of this paper, although we believe it is an important future extension of the work.

(a) Lane change scenario. (b) Crossroads. (c) Roundabout.

Fig. 3. An example of (a) lane change, (b) crossroads, and (c) roundabout scenarios.

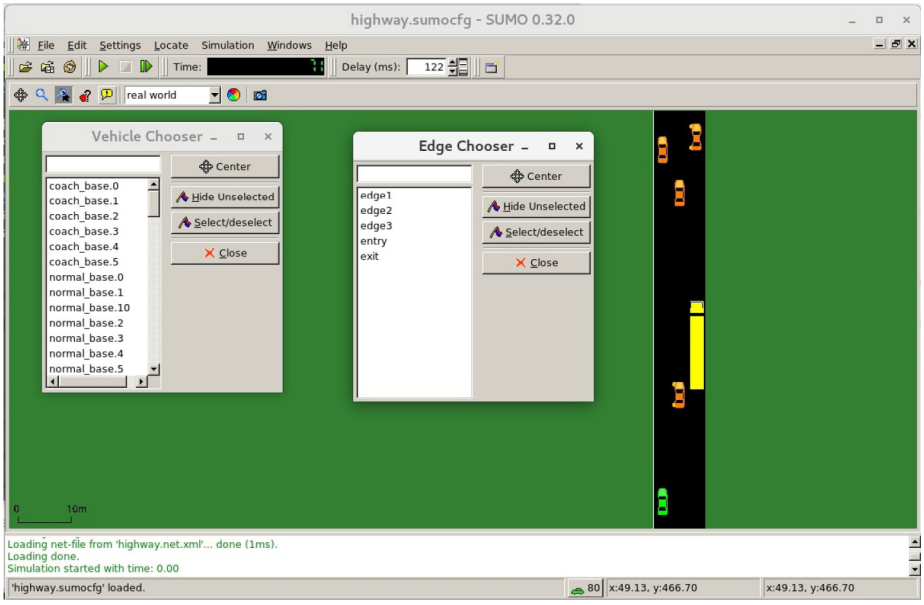


Fig. 4. SUMO graphical interface.

5.2. Lane change scenario

The lane change problem consists in controlling the ego vehicle in the highway and deciding when it is convenient to perform a lane-change on the left in order to overtake, or a lane-change on the right, to occupy the rightmost free lane. This is one of the most tackled problems in the practical application of RL to AD, thanks to the potentially simple representation of the environment and the "few-constrained" possibility in choosing the action. The goal of this problem is to control the ego vehicle along the highway, exposing a natural behavior, going as fast as possible and avoiding *dangerous maneuvers*. We qualify as dangerous a lane-change maneuver that would violate the safety distance with other vehicles in the target lane. The safety distance is defined as a function of the stopping distance of two vehicles, a follower and a leader. More formally, the safety constraint between a follower vehicle and a leading vehicle is defined via the condition:

$$\begin{aligned} &\text{dist}_{\text{stop\_follower}} + \text{min\_gap} < \text{dist}_{\text{stop\_leader}} + \text{vehicle\_gap}, \\ &\text{dist}_{\text{stop\_follower}} = \text{dist}_{\text{reaction}} + \text{dist}_{\text{brake}} \\ &\qquad = \tau \frac{v_f + (v_f + \tau a)}{2b} + \frac{(v_f + \tau a)^2}{2b}, \\ &\text{dist}_{\text{stop\_leader}} = \text{dist}_{\text{brake}} = \frac{v_l^2}{2b}, \end{aligned} \tag{8}$$

where min_gap is the minimum safety distance between two vehicles, vehicle_gap is the distance between the leading and following vehicle, τ is the reaction time, v_f is the speed of the follower, a is the acceleration of the follower, b is the emergency deceleration and v_l is the speed of the leader. So the safety distance is composed of three parts, the distance traveled during the reaction time τ, for which we assume maximum acceleration a, the braking distance, assuming maximum deceleration b, and the min_gap safety gap between two vehicles. Table 2 shows the values of the safety distance parameters used in our experiments. Fig. 5 shows an example where the safety constraints are satisfied. The following vehicle's stopping distance is shown in red. It is composed of three parts, the reaction time distance, braking distance and *min_gap*. This stopping distance is lower than the sum of the distance between the two vehicles (shown in blue) and the stopping distance of the leader vehicle (shown in yellow).

Table 2
Parameters for the highway scenario.

min_gap	τ	a	b
2.5 m	1s	$3\frac{\text{m}}{s^2}$	$3\frac{\text{m}}{s^2}$

The low-level controller prevents dangerous maneuvers from being executed, considering the vehicle inside the sensors range only. To ensure that the resulting policy avoids requesting unsafe actions (even if blocked in lower control levels), these maneuvers are recorded and are considered to compute the reward signal. With the low-level controller maintaining the safety distance

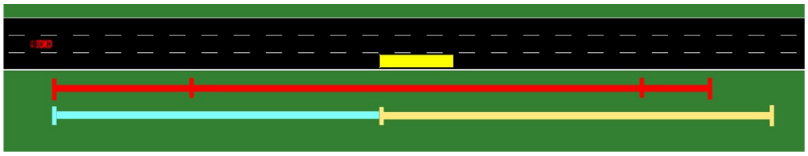


Fig. 5. Safety distance in a lane change scenario.

Fig. 6. Safety distance in a crossroads scenario.

Table 3
Parameters for the crossroads scenario.

min_gap	τ	$a_{\max}$	b
2.5 m	1s	$0\,\frac{\mathrm{m}}{\mathrm{s}^2}$	$3\,\frac{\mathrm{m}}{\mathrm{s}^2}$

with the other vehicles, the task is reduced to finding the right timing to perform the lane-changes on the left and on the right, in order to maintain a high-speed profile and a general respectful behavior.

5.3. Crossroads scenario

The crossroads scenario consists in an intersection with an arbitrary number of roads. The ego vehicle coming from the source road has to reach a target road, with a higher priority. We consider a priority intersection, thus, the ego vehicle has to give way to the other vehicles, but not necessarily stop before entering the intersection.[3] The goal of the agent is to drive the ego car and enter the target road, avoiding dangerous maneuvers. The agent can control the car by accelerating and decelerating. The route of the car is managed by the low-level controller, that also maintains the safety distance with the car in front of the same road. If the ego vehicle performs a *dangerous entrance*, the task is considered failed. A dangerous entrance is defined as an entrance where there is a car "too close" to the junction on one of the other roads entering the junction, where by "too close" we mean the case in which if the ego vehicle stops in the junction, the other vehicle cannot stop before entering the junction, generating an unsafe situation. In this manner, a dangerous entrance is defined as a function of the stopping distance of the other cars in the road. We can compute the stopping distance $\mathrm{dist}_{\mathrm{stop}}$ of a vehicle as follows:

$$\begin{aligned}\mathrm{dist}_{\mathrm{stop}} &= \mathrm{dist}_{\mathrm{react}} + \mathrm{dist}_{\mathrm{brake}} + \mathrm{min\_gap} \\ &= \tau\frac{v + (v + \tau a)}{2} + \frac{v + \tau a}{2b} + \mathrm{min\_gap},\end{aligned} \tag{9}$$

where v is the distance of the car from the junction, τ is the reaction time, a is the acceleration during the reaction time, b is the maximum deceleration and min_gap is the minimum gap allowed between two vehicles. The values of these parameters are reported in Table 3. Fig. 6 shows an example where the ego vehicle respects the safety distances, while entering the junction to reach the destination road on the left. The segments in red show the components of the safety distance. It is important to note that this definition of dangerous entrance is very conservative, and makes the task harder to learn. This choice is justified since it ensures that no crashes can occur.

[3] A scenario with the stop signal at the intersection is trivial from a learning standpoint: the ego car slows down comfortably before the intersection to stop right before it, and then it enters the junction as soon as possible, when the other vehicles are at safe distance.

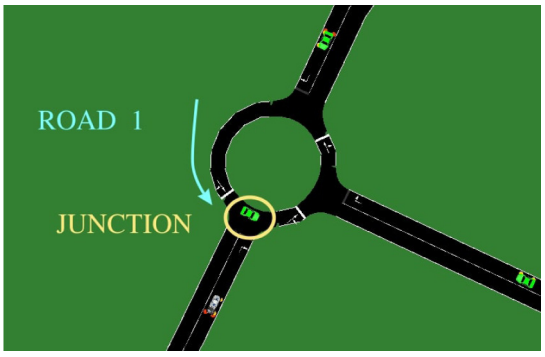


Fig. 7. Junction and road considered for a roundabout scenario.

5.4. Roundabout scenario

The roundabout scenario can be interpreted, and consequently modeled, as a particular instance of a crossroad. This modeling choice allows reusing the state representations and policies employed in the crossroads. We limit out treatment to single lane roundabouts. The ego vehicle has to enter the roundabout in a safe way and stay in the roundabout until its exit. When entering, the ego vehicle has to give way to the cars already in the roundabout. This scenario can be reduced to an intersection of the portion of the roundabout where the road of the ego vehicle merges. The only road to consider to chose the action, is the portion of the roundabout that enters in this intersection from the left, as seen in Fig. 7. The entrance in the roundabout is considered dangerous if there is a car on the roundabout coming from the left of the junction too close. The safety distance rules considered for the roundabout are the same as the crossroads scenario. The only difference, in our experiments between a roundabout and a normal crossroads, is the maximum speed of the road which was set to $8.5\,\frac{\mathrm{m}}{\mathrm{s}}$ instead of $14\,\frac{\mathrm{m}}{\mathrm{s}}$.

6. Design of the Markov decision problem

In the previous section, we introduced the three driving scenarios we will address with RL. We now discuss how to design

a suitable MDP. Specifically, we will illustrate the state space representation (Section 6.1), the action spaces (Section 6.2), and the reward function design (Section 6.3). In our experiments, we employed a value for the discount factor $\gamma = 0.999$.

6.1. State space representation

The design of the state space $\mathcal{X}$ is a crucial choice for the high-level controller, as the information made available to the decision module must be, at the same time, exhaustive and concise, in the sense that it must contain all the information available needed to make an optimal decision, possibly preprocessed in a set of features on which we can effectively build a rule-based policy. Classical state representation approaches used in the autonomous driving literature, such as raw sensors, images, or occupancy grids [74], are inappropriate for our purposes as any decision made directly on these input spaces would hardly be interpretable.

In our framework, the state is composed of a vector of *features* related to the ego vehicle and the other *vehicles of interest*, that are located within the sensor range. The definition of these vehicles depends on the specific AD scenario faced. For instance, in the lane change scenario, we include in the state the information about the vehicles in the lanes around the ego vehicle, whereas, in the crossroads case, we consider the vehicles in the roads entering the junction. For each vehicle of interest and for the ego vehicle, we record the speed and position, along with some additional features depending on the scenario (see Appendix A for details). It is important to stress that the features extracted from the environment to construct the state of the decision maker are easily obtainable from the sensing module of the real-life car.

Modeling the state in this way produces a *factored* representation, in which each factor corresponds to a single vehicle (the ego vehicle or a vehicle of interest). As a consequence, the full state turns to have variable length, according to the number of vehicles placed within the sensor range. More formally, the state of the environment $\boldsymbol{x}$ is composed of a vector of k variables regarding the ego vehicle's state and the concatenation of vectors of h variables for each of the n vehicles of interest in the sensor range:

$$\boldsymbol{x} = \langle \underbrace{\boldsymbol{x}_0}_{\in \mathbb{R}^k} , \underbrace{\boldsymbol{x}_1}_{\in \mathbb{R}^h}, \ldots, \underbrace{\boldsymbol{x}_n}_{\in \mathbb{R}^h} \rangle \in \mathbb{R}^{k+nh}.$$

ego vehicle; vehicles of interest

Remarkably, this makes our state general w.r.t. the road topology, meaning that we can represent a variety of scenarios.[4] For instance, in the crossroads scenario, a state coming from an intersection with five incoming roads is indistinguishable from a state coming from a three roads intersections. As a consequence, the same policy can be used in both of this scenarios. Similarly, this state representation allows handling highways with different number of lane.

Partial observability. An important issue to face in the autonomous driving task is *partial observability* (PO). Typically, the state representation does not capture all the relevant information of the environment needed to make the optimal decisions. Obvious sources of PO include, but are not restricted to, the limited sensor range and the partial occlusions of objects of interest (e.g., vehicles in the lanes around the ego vehicle or entering the junction) from objects in the road or objects outside the road (e.g., buildings in the side of the road). In this work, we address these two sources of PO in a conservative way, by means of a preprocessing of the state. Thus, we add, at the limits of the visible range (which could be the end of the sensor ranges in the roads of interest or the closest portion of visible road not occluded from obstacles) *fictitious* vehicles, i.e., extra vehicles at the limits of visibility, with worst-case speeds. More specifically, in the highway scenario, we add to the state extra vehicles, positioning them at the limits of the sensor ranges, in front and behind our vehicle, in all the lanes. In the crossroads scenario, we consider also occlusion due to outside objects as in Fig. 8(a). The building is represented from the gray rectangle in the bottom-right corner. The effect of the building on the visibility range of the ego vehicle is easy to calculate through standard triangle similarities. The yellow vehicle, shown in the border of the visible portion of the road is a fictitious vehicle added to the state of the controller. These vehicles are not really part of the simulation. Their distance from the junction is set to the limit of the visibility and their speed v_{fict} is set such that the fictitious vehicle will enter the junction at the same time with the ego vehicle, assuming constant velocity. Calling tte_{ego} the time to enter of the ego vehicle, with the assumption of constant velocity, s_{vis} the visibility range, the speed of this fictitious vehicle is given by:

$$v_{fict} = \frac{s_{vis}}{tte_{ego}} = \frac{s_{vis} v_{ego}}{s_{ego}},$$

where v_{ego} and s_{ego} are the speed and distance from the junction of the ego vehicle. By setting the fictitious vehicles in the state, for each road, at the limit of the visibility, with the speed discussed above, we effectively formalize the worst-case scenario, producing conservative behaviors. The case of vehicles occluding other vehicles is handled in the same way as in Fig. 8(b), setting a fictitious vehicle in the portion not occluded from a vehicle.

This is a very conservative handling of PO as we do not consider earlier versions of the state in which the vehicle was not occluded. As soon as a vehicle becomes occluded, we "jump" its speed and distance from the junction to the worst case. As an alternative, a tracking model could be employed to give more realistic predictions of the speed and distance of the occluded vehicles when they are not visible. This would produce less conservative behaviors. Nevertheless, it is important to stress, that the same policies, trained with these conservative values for the fictitious vehicles, could be employed with less conservative state pre-processing such as the vehicle tracking, as it would only change the state given in input to the decision module.

6.2. Action spaces

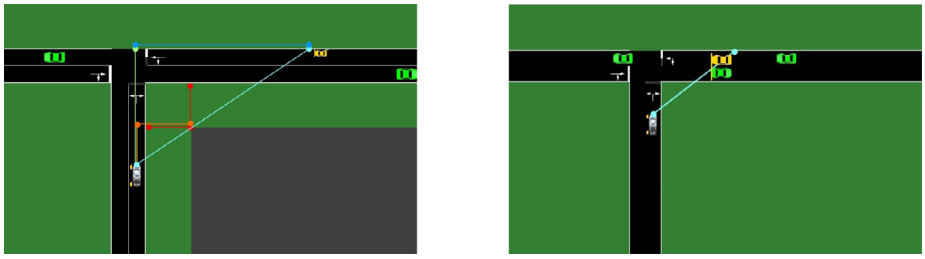
We now introduce the action spaces $\mathcal{U}$ we will employ in the different scenarios. For both, we introduce a *total order relationship* $\prec$, that we will employ in the execution of our rule-based controller (Section 7). The idea is to rank the available actions according to their degree of "conservativeness" (e.g., a deceleration is more conservative compared to an acceleration), in order to select, at the decision stage, the most restrictive one.

Lane change action space. The action space in the lane change highway scenario is composed of three actions, having the following ordering:

car_following $\prec$ lane_change_right $\prec$ lane_change_left

The *car_following* action leaves the control of the car to the low-level controller which follows the route planned, curving when it is necessary, but does not perform lane changes. Furthermore, it controls the speed of the vehicle in order to avoid collisions and maintains a safe distance with the vehicle in front. The controller sets the safety speed considering the vehicles that are in

[4] It is worth noting that no information about the topology of the road is embedded in the state. Indeed, the rooting is in charge of the lower level controller, allowing a totally *topology-agnostic* modeling at the high-level decision-making stage.



(a) PO induced by a building. (b) PO induced by vehicles in the road.

Fig. 8. Models of partial observability.

sight of the sensors only and adjusts it respecting the maximum deceleration and acceleration viable. The remaining two actions are the *lane changes*, left or right. These maneuvers are non-interruptible, once they are issued, they cannot be reverted. In our simulations, they last 3.1s and are performed with a maximum lateral acceleration of $2\frac{\text{m}}{\text{s}^2}$.

Crossroads action space. The decision module of the crossroad scenario is in charge of controlling the acceleration of the ego vehicle. At each decision step, this modules takes in input a state of the environment and outputs an acceleration value in the range $[u_{\min}, u_{\max}]$. In our experiments, we employed the values $u_{\min} = -3\frac{\text{m}}{\text{s}^2}$ and $u_{\max} = 1\frac{\text{m}}{\text{s}^2}$. One of the main concerns in the crossroads scenario consists in ensuring a sufficient degree of comfort, which requires avoiding too many deceleration–acceleration switches.[5] For this reason, we evaluated two different action space: a discrete action space and a continuous one.

- *Discrete Action Space.* The action space is composed of six deceleration levels $\{-3, -2.5, -2, -1.5, -1, -0.5\}\frac{\text{m}}{\text{s}^2}$, an acceleration action corresponding to the value $1\frac{\text{m}}{\text{s}^2}$ that is kept until the maximum allowed speed is reached, and a no-acceleration action. Overall, the action space is composed of eight actions, having the following ordering:

$$\underbrace{\text{dec\_3} \prec \text{dec\_2.5} \prec \text{dec\_2} \prec \text{dec\_1.5} \prec \text{dec\_1} \prec \text{dec\_0.5}}_{\text{decelerations}} \prec \underbrace{\text{zero}}_{\text{no-acceleration}} \prec \underbrace{\text{acc\_1}}_{\text{acceleration}}.$$

- *Continuous Action Space.* The action is chosen in the continuous interval $\mathcal{U} = [-3, 1]$ and the ordering is simply given by the natural ordering of real numbers. This case, employing a superset of actions compared to the first one, offers the vehicle greater control, but, on the other hand, it might make the problem harder to optimize.

6.3. Reward function

We turn now to the design of the reward functions used in our work. A good reward function needs to encode all aspects of the desired behavior. For example, a good driving controller in the crossroads scenario should enter the junction as fast as possible, while respecting the safety constraints, and producing a smooth trajectory to ensure the comfort of the passengers. The lane change scenario, on the other hand, has different goals, which implies a different reward function. We consider linear reward functions only, for both driving tasks considered. Thus, the reward is a linear function of a vector of p features, defined over the tuple $(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')$, where $\boldsymbol{x}$ is the current state, $\boldsymbol{u}$ is the action executed in $\boldsymbol{x}$ and $\boldsymbol{x}'$ is the next state:

$$r(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}') = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}'), \tag{10}$$

where $\boldsymbol{\phi} : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \to \mathbb{R}^p$ is a feature function mapping the experience tuples to a vector of features and $\boldsymbol{\phi}$ and $\boldsymbol{w} \in \mathbb{R}^p$ is a weight vector. The design of the reward function includes two design choices: i. engineering the set of feature functions $\{\phi_i\}_{i=1}^p$ and, afterwards, ii. choosing an appropriate weight vector $\boldsymbol{w}$. The weight vectors we employed were a product of a grid search, meant do identify a suitable weighting vector allowing for a behavior compatible with the intuitive notion of "driving well". Automatic ways to derive the weighting vector starting from a set of expert demonstrations, i.e., episodes collected from an agent playing the optimal policy (e.g., a human) exist and they are based on *inverse reinforcement learning* [75].

Table 4
Reward features and corresponding weights for the highway scenario.

Feature $\boldsymbol{\phi}$	*Domain*	*Objective*	*Weight* $\boldsymbol{w}$
target_speed_distance	[0, 1]	High speed profile	−3.3
free_right	{0, 1}	Good driving behavior	−1
useless_lane_change	{0, 1}	Comfort	−16
safety_violation	{0, 1}	Safety	−1

Lane change reward function. The lane change scenario is the classical example of a multi-objective problem. The agent has to maximize different conflicting objectives. The agent has to maintain a high speed profile, while staying in the rightmost lane as much as possible, and avoiding useless lane changes to ensure the comfort of the passengers. Clearly, the agent's behavior must respect the safety constraints. Table 4 shows the features and the corresponding weights used in this task. To ensure a high speed profile, we give a negative reward proportional with the difference between the ego speed and the maximum speed of the road (*target_speed_distance*). This encourages the agent to maintain high speeds to avoid the punishment. Additionally, we consider three binary features to model unwanted situations. First, the feature *free_right* is activated when the lane on the right is free, while the *useless_lane_changes* is activated when a lane change on the left is performed without overtaking. Finally, the *safety_violation* feature is activated when the agent requests an action that would violate the safety constraints.

Crossroads reward function. In the crossroads scenario we consider a constant feature at each decision step, weighted with −1 (*constant_time*). This is a standard reward feature in goal-based problem as it incentivizes the agent to finish the episode as fast as possible. Moreover, we add a binary feature, *harsh_slow_down*, which activates whenever $\boldsymbol{u}_t$ is lower than a threshold (the

[5] We will formalized this requirement with the notion of *jerk* in Section 6.3.

Table 5
Reward features and corresponding weights for the Crossroads scenario.

Feature $\boldsymbol{\phi}$	*Range of Values*	*Objective*	*Weight* $\boldsymbol{w}$
constant_time	{1}	fast completion	-1
harsh_slow_down	{0, 1}	Comfort	-2
jerk	[0, 1]	Comfort	-0.25
dangerous_entrance	{0, 1}	Safety	$H \cdot (-1-2-0.25)$

threshold used in our experiments was -1.5). This feature is related to the comfort objective, as it pushes the agent to avoid decelerating too harshly. The third feature is the absolute value of the instantaneous *jerk*, i.e., the finite-difference derivative of the acceleration. This feature also encodes the comfort objective and is normalized by its maximum value. More formally the jerk feature at decision step $t+1$ is:

$$\text{jerk}_{t+1} = \frac{1}{\text{max\_jerk}} \cdot \frac{|a_{t+1} - a_t|}{\Delta t},$$

where a_{t+1} and a_t are the acceleration at the two consecutive steps, Δt is the decision frequency, 0.1 s and max_jerk is the maximum possible value of the jerk determined by the range of possible acceleration. Finally, we used the binary feature of *dangerous_entrance*, which activates when the ego vehicle violates the safety constraints or performs a crash. The weight of this feature is dependent on the weights given to the other features as this is considered the main objective of the task. Violating the safety constraints or crashing into another vehicle is the worst-case performance, so the corresponding weight is set to a value for which every other possible behavior dominates performing a crash. Specifically, the weight of the *dangerous_entrance* is set to the sum of the weights of the other features, multiplied by the horizon H. In this way, performing a dangerous entrance is never convenient. Table 5 shows the features and the relative weights for the crossroads scenario.

7. Design of the parametric rule-based controller

In this section, we present the structure of the rule-based policies, employed in this work. As discussed in Section 4, the policy is modeled as a parametric set of rules whose parameters are optimized using RL algorithms. To make the policy general w.r.t. the type of scenario addressed (especially to the number of vehicles of interest involved), we model the policy $\pi_{\boldsymbol{\theta}}$ as a module that takes as input the information about the ego vehicle and a *single* vehicle of interest, i.e., any vehicle that is present around the ego vehicle (which is the only one we control). The rule-based policy outputs an action that takes into account only the presence of the currently considered vehicle of interest. By running the rule-based policy we obtain a set of actions (one for each vehicle of interest). The action to be executed by the ego vehicle is the "conservative" action, according to the total order of actions $\prec$ previously defined. In this way, we get a controller that is robust w.r.t. the choices of the surrounding vehicles.

The general structure of the decision module is given in Fig. 9. The features of each vehicle of interest $\boldsymbol{x}_i$ in the state representation, together with the features relative to the ego vehicle $\boldsymbol{x}_0$, are passed through the same parametric policy $\pi_{\boldsymbol{\theta}}$, producing an action relative to each vehicle $\boldsymbol{u}_i$. These actions are then filtered to determine the most conservative action, according to $\prec$. It is worth noting that the features in input to the parametric policy will also be relative to the fictitious vehicles added to the state to cope with the partial observability of the problem.

Each $\pi_{\boldsymbol{\theta}}$ is built as a decision tree, with a structure that is problem-dependent, and whose nodes are parametrized with the parameter vector $\boldsymbol{\theta}$. The goal of the learning agent is to find the best parameters for the nodes of the decision tree. A straightforward approach is to leave the design of the tree structure to experts of the field and then parametrize it. In this case, the learning algorithm would just find the best set of parameters for a specific policy structure, which is also the approach adopted in this work. Another approach consist in using expert demonstrations to fit a decision tree to the actions taken, to derive a structure of the tree. After having this structure, we optimize the parameters of the node using RL.

To give an example of the approach, Fig. 10 shows a general structure the rule-based policies used in the crossroads scenario with the discrete action space. The first level of decision is meant to discriminate between safe and unsafe situations. For this purpose, the rule-based policy has three parameters θ_1, θ_2 and θ_3, to be optimized, which represent ranges of the features of the state. Specifically, this notion of safety is encoded by means of two conditions on the *time to enter* (tte_{ego}), which represents the time needed to reach the junction assuming a constant speed, and on the *safety feature* (sf), which is an aggregated feature used to discriminate whether entering the junction is currently safe (its precise definition is given in Appendix B.2). The learning algorithm is tasked to find "safe" regions of the features of the state, for which the autonomous vehicle should accelerate. In the case of an unsafe situation, the rule prescribes to brake and the tree has one or more leaves for each discrete level of deceleration. The path to these leaves is parametrized. In our experiments, we used simple thresholds on the *time to enter* of the ego vehicle to decide the level of acceleration. The values of these thresholds are optimized by the learning algorithm. In the default case the vehicle decides to maintain the same speed. The threshold are all learned separately, which means that if the ordering between adjacent thresholds are not kept, the learning algorithm can effectively "cut-off" some levels of deceleration, reducing the number of deceleration actually employed. Instead, the acceleration action is fixed to *acc_1*, which means give control to the low level controller, which accelerates if possible, without braking the speed limits. For details on the policies used for the highway and crossroads scenarios refer to Appendix B.

8. Experiments and results

In this section, we provide an experimental campaign of the proposed method in the highway (Section 8.1) and urban (Section 8.2) scenarios discussed earlier.

8.1. Lane change scenario

We evaluate our method in six different highway networks, all modeling highways with three lanes, with different levels of randomized traffic. To validate our method we devise a simple set of rules with seven parameters, and trained our model using PGPE, discussed in Section 4. Details about this rule-based policy are given in Appendix B. We compare our methods with DQN [22], described in Section 2.3. After tuning, we used a two layer network, multi-layer perceptron, with 64 hidden units per layer. This network has around 2000 parameters to be learned, a significantly larger amount compared to our method. On the other hand, DQN does not constrain the policy space as much as our rule-based policy, possibly leading to more performing controllers. We train each algorithm for 25M steps of experience. After 500 training episodes, we stop the training process and evaluate the best policies found so far with each method. Fig. 11 shows the training curves of the algorithm. First, we notice that

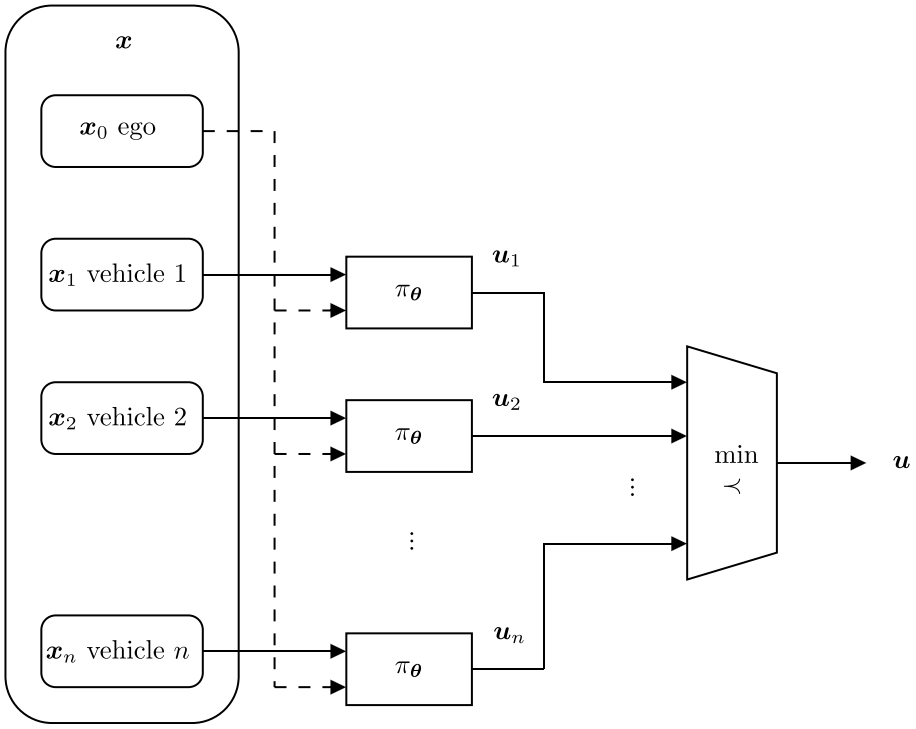


Fig. 9. Graphical representation of the decision module.

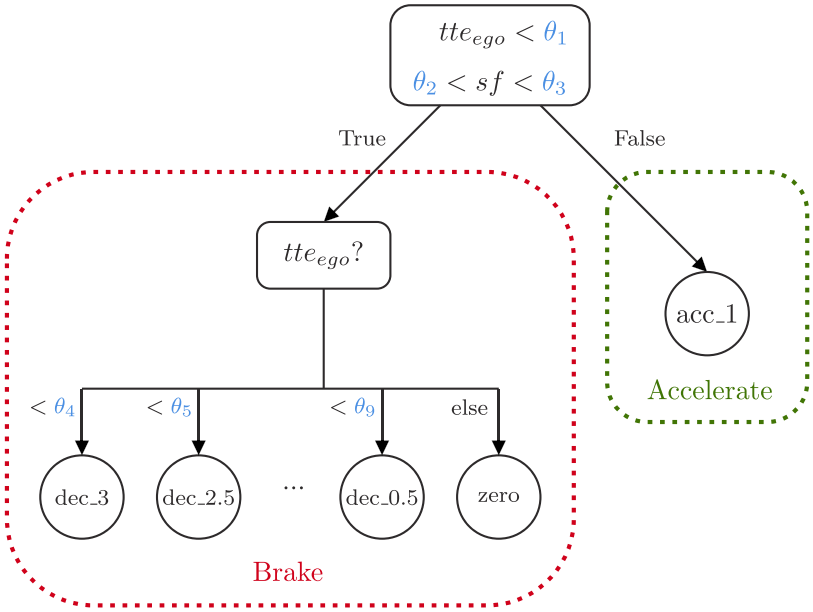


Fig. 10. Illustrative representation of the rule-based policy for the crossroad scenario with the discrete action space.

(on average) PGPE was able to perform better than DQN in the experiments we conducted. Indeed, already at the starting point, PGPE outperforms DQN. This is also reasonable, since at the beginning of the training schedule DQN is a completely random policy, whereas the rule-based policy optimized by PGPE is hand-coded with a set of hand-picked initial parameters. This is another advantage of our method, since starting from a good initial policy is not so straightforward in DQN. Furthermore, DQN, in our experiments, was unable to catch up with the performance of PGPE. Secondly, the training process of DQN is more unstable than the one of PGPE, with sudden oscillations in the performance of the policy. Also, as expected with a Deep RL method, DQN has a higher variance of the return during training. Finally, although it may seem that PGPE has converged early in the training schedule, this is because the improvement on PGPE is small w.r.t. the oscillations of DQN. For more information on the training process of PGPE, Appendix C.1 shows details on different KPIs during

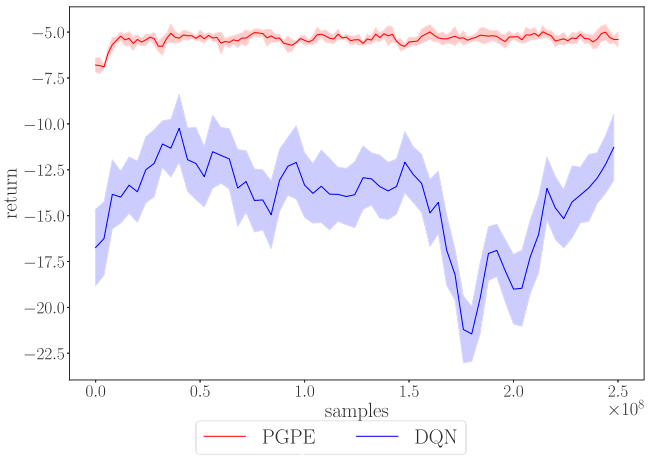


Fig. 11. Training curves of PGPE and DQN. Average of 5 runs with 95% c.i.

the training process and how they continue being optimized throughout the whole training process.

Next, we compare the best policies that we were able to derive from each algorithm. This because, although DQN has a high variance and unstable training process we were able to derive a "good enough" policy from it (even though early in the training procedure). Fig. 12 shows the comparison between our method (PGPE), DQN and the performances of a human controlling the ego-vehicle by hand (HUMAN). The plots show the average value of the recorded KPIs during an evaluation of 100 episodes together with the 95% confidence intervals, represented by the black lines. Each evaluation episode is 40 s long (400 decision steps). First we notice that a Deep RL method like DQN is able to adequately solve this task, maintaining high speed and traveled distance while having a low number of actions blocked by the lower level controller. Nonetheless, by applying the rule-based approach, even though we constrain the policy space to a specific predefined structure, we are able to further optimize the distance traveled, average speed and blocked maneuvers KPIs. At the same time PGPE performs, on average, less lane changes (LC), which improves on the comfort objective (as can be seen in the Left LC, Right LC, and Useless LC KPIs). Our method is also able outperform DQN on the Could Overtake KPI, which denotes the average number of decision steps in which the vehicle could perform a safe lane change, but due to a vehicle in front entering its safety distance, it is decelerating. Effectively, our trained rule-based policy brings the number of these steps close to zero compared to an average of 7 steps of DQN, meaning we are better at finding the right moment at which to perform a lane change, compared to DQN. We also evaluated the performance of a human controlling the vehicle through a graphical representation of the simulation (HUMAN). A human controller can still drive further and with a higher speed compared to DQN and PGPE, while suffering in other KPIs, like the steps with the free right and blocked maneuvers. This is also due to the fact, that it is harder for a human to judge the safety distances with the surrounding vehicles from a graphical representation. The policy PGPE-INIT, represents the performances of the starting point, of the rule-based policy, with hand-picked parameters. The value of these parameters has been selected by manual adjustment. A series of preliminary experiments have been conducted to achieve a reasonable behavior that avoids too extreme maneuvers. This can be seen as an instance of a handcrafted rule-based controller. We can see, that even though this policy avoids dangerous maneuvers, it is not able to perform well in the other KPIs, showing the importance of optimizing the parameters of a rule-based policies. For completeness, we also show the performances of a random policy uniform over the three available actions, which is also the starting point of the DQN algorithm. While in our approach it is straightforward to initialize the learning process from an *adequate* policy, in the DQN case, given the complexity of the policy model, it is very difficult. For details on the hyperparameters of the learning algorithms refer to Appendix C.

8.2. Urban scenarios

We use four different road networks to evaluate our method, an intersection with three roads like in Fig. 8, two intersections with five roads with different traffic levels, and one roundabout network. We perform the same tests as in the lane change scenario, comparing our rule based policies, with DQN and a human driver. The same training schedule and network architecture has been used for DQN as in the lane change scenario.

We employed PGPE with two different rule-based policies, one for a discrete action case and one for the continuous version, as described in Section 6.2. Fig. 13 shows the learning curve of PGPE with both action spaces and DQN. Here the advantage of using PGPE with a rule-based policy is evident. DQN completely fails to optimize this task and shows highly unstable behavior during training. This is mainly due to the nature of the task. First, it is a difficult exploration problem. The task has 8 discrete actions, of which 6 are decelerations, so it is easy to end in a sub-optimal policy that stops before reaching the junction. Furthermore, the reward function itself is hard to optimize since extremely high punishments are received whenever the agent performs dangerous maneuvers. The exploration problem is less problematic in the case in which PGPE is employed since the policy is constrained in a specific form, defined by the structure of the rules. In fact we can see that since the beginning we have a better performance than DQN (better than the random initial policy) and we continue to optimize it. Appendix C.1 shows details of the training curves of PGPE for some KPIs of interest in this scenario.

Next, we compare the results of the best policies in terms percentage of intersection they are able to successfully complete, how fast they complete them and the comfort of the resulting policies, by recording the average jerk of an episode and the number of comfortable or harsh slow downs (SD) performed. Recall that we label as harsh slow downs the slow downs with deceleration greater than $1.5\frac{\mathrm{m}}{s^2}$. An episode is interrupted after 60 s (600 timesteps), if the agent was not able to complete the intersection. Fig. 14 shows the average values for some KPIs of interest, over an evaluation of 100 episodes. We compare the best policies achieved after a training period of 20M samples of experience, with DQN and PGPE. First we notice a further confirmations, that in this scenario, an end to end method like DQN fails to learn the task. Learning a good policy in the crossroads scenario is harder than in the lane change, as the agent needs to perform more effective exploration to find well-performing policies able to complete the intersection. Exploration is still an open problem in RL literature, and is especially problematic in this scenario when performed in the action space, as it is done in the DQN algorithm. Since most of the actions are decelerations, the learning process has a bias towards policies that do not complete the intersection, but simply stop. Instead, by moving exploration to the parameter level, like in PGPE, we partially mitigate this problem. Next, we compare the results of the starting point of the policy trained with PGPE (denoted PGPE-INIT in the plots) with the resulting policy from the learning process of DQN and PGPE. We can see that, differently from DQN, with the rule-based approach proposed, we are able to start the learning process from a policy that has quite a good performance, and

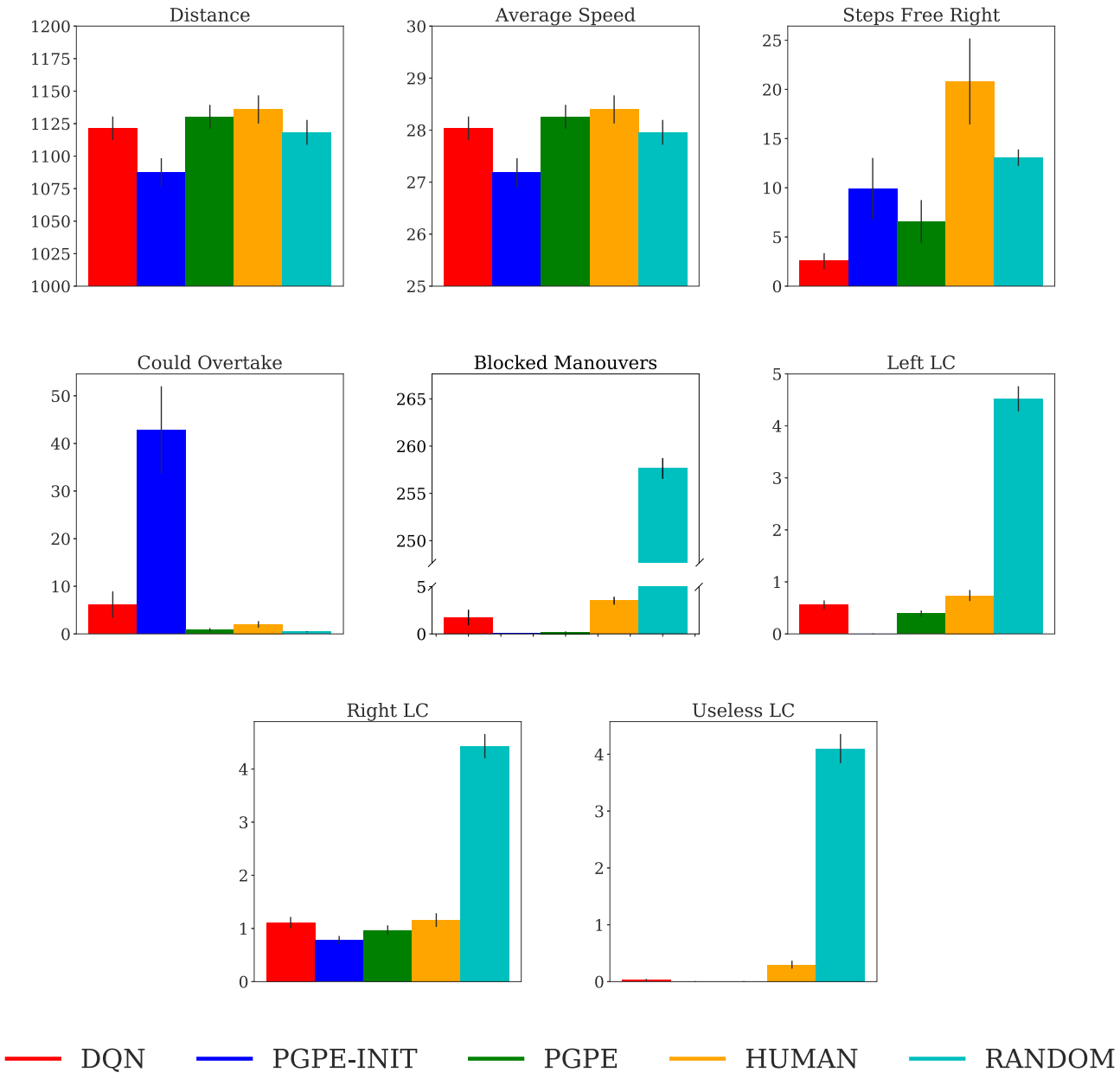


Fig. 12. Experimental evaluation in the Highway Scenarios. Average over 100 episodes together with 95% c.i.

we are able to refine this hand picked parameters. Initializing the neural network to a good, predictable policy, on the other hand, is not that straightforward. We see that PGPE is able to optimize the parameters of the rule based policy until reaching a near-optimal policy while reducing to a minimum the dangerous entrances and the harsh slow down actions. We still have 1% of dangerous entrances, but it is worth noting that the definition of the dangerous entrances used is highly conservative, as described in Section 5. This can be seen also from the KPIs of the human demonstrations, in which a human controlled the vehicle from the graphical interface of the simulation. We see that it is hard, even for a human, to judge the dangerous entrances, being very conservative, so as to remove the chance of crashes happening. Finally, from a comparison of PGPE and a human driver, we see that a human is still better at controlling the jerk of the vehicle (comfort objective). For this purpose, we employed also a policy with a continuous action space as described in Section 5, to investigate the effect of the action space on the comfort objective. As we can see, using PGPE with a policy that outputs directly the desired level of acceleration, we are able to further optimize the jerk objective to nearly the same of a human, while losing little to nothing with respect to the other objectives.

9. Conclusions and future work

In this work, we showed how reinforcement learning and rule-based approaches can be combined, to design a decision-making controller for an autonomous vehicle. The resulting parametric rule-based policy is able to achieve both transparency (thanks to the usage of interpretable rule) and robustness (thanks to the generalization capabilities of RL). The experimental campaign highlights two main points. First, we empirically demon-

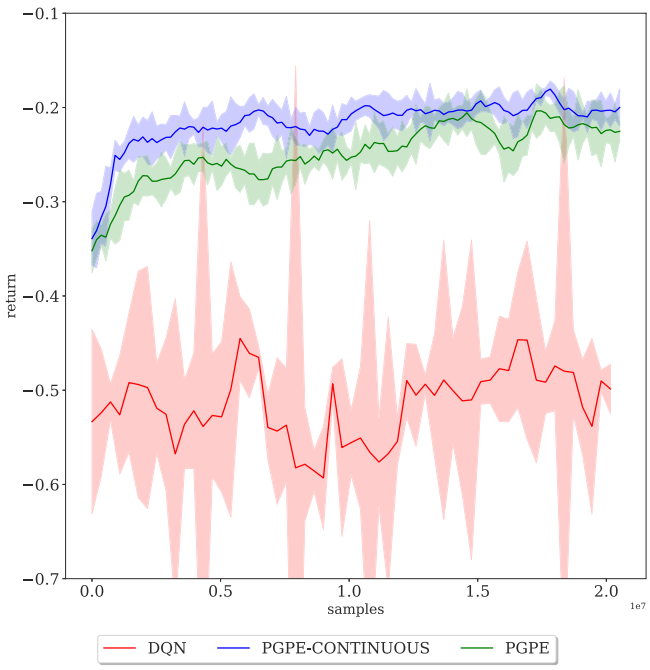


Fig. 13. Training curves of PGPE and DQN. Average of 5 runs with 95% c.i.

strate that RL, especially parameter-based policy optimization (PGPE) turns out to be an effective learning tool to improve the performance of hand-crafted policies. Second, we illustrate how black-box approaches based on RL, like DQN, not only produce controllers that are hard to interpret, but given the complexity of their policy space, often fail to learn effectively the task.

We are convinced that the combination of RL with rule-based controllers represents a promising path in the development of a real-world autonomous vehicle. On the one hand, we are able to achieve transparency and, thus, have precise control over what happens within the policy, possibly avoiding undesirable corner cases. On the other hand, thanks to the parametric nature of the considered rules, we have access to a larger policy space compared to handcrafted rule-based approaches. This enforces robustness since the interactive learning process at the basis of RL allows generalizing over previously unseen situations.

Nevertheless, the generality of the learned controller is tightly connected to the structure of the parametric rules. Indeed, we restrict the class of rule-based controllers to those that consider the ego vehicle together with a single vehicle of interest at a time. In this way, among the actions suggested for each vehicle of interest, we pick the most conservative one. In principle we could consider a structure for the rule-based controller that takes as input the ego vehicle and a set of vehicles of interest. In this way, we could

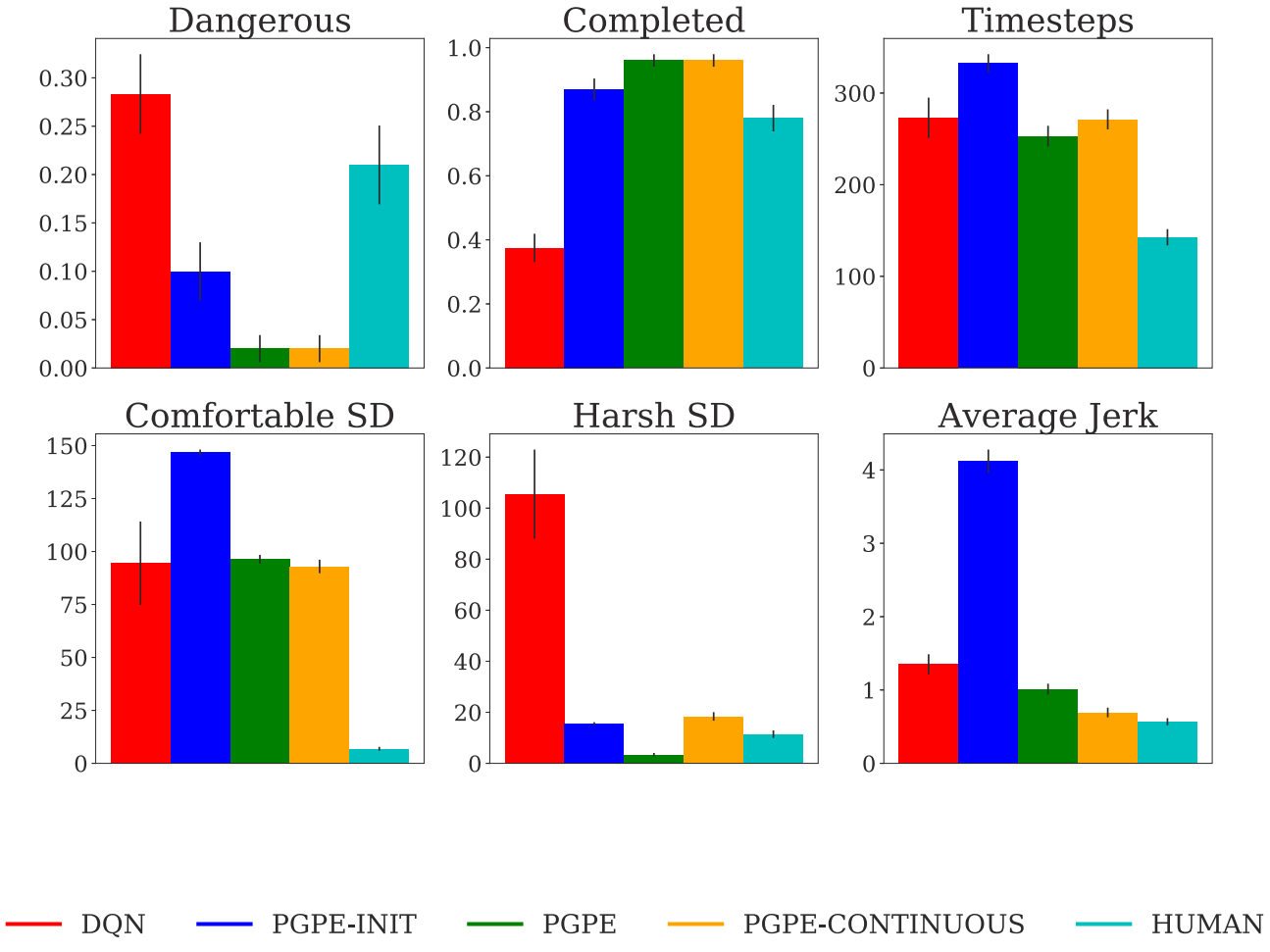


Fig. 14. Experimental evaluation in the urban scenarios. Average over 100 episodes together with 95% c.i.

make a decision based on relative information (e.g., relative speed or relative distance) between a pair of vehicles of interest. Clearly, this approach would increase the generality, at the price of a more complex controller, with possibly a larger number of parameters and an increase in the cost of execution. More generally, the choice of suitable complexity for the controller is a trade-off between sample complexity and representation power, which is worth investigating.

Several future extensions of this work are possible; we discuss two of them in the following. First, we mainly focused on robustness and transparency, partially neglecting comfort. Although we believe that our reward functions implicitly prescribe a "comfortable" behavior (e.g., continuously switching between acceleration and brake is for sure suboptimal), we could include in the learning process some explicit terms encoding the comfort requirements. Second, our evaluation is restricted to simulation environment. This is essential to employ an RL-based approach in which exploration, possibly leading to dangerous behaviors, is necessary. However, once the policy is learned it can be implemented in the real vehicle and evaluated there, opening new challenges concerning transfer learning and domain adaptation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.robot.2020.103568.

References

[1] D. Krajzewicz, J. Erdmann, M. Behrisch, L. Bieker, Recent development and applications of SUMO-simulation of urban mobility, Int. J. Adv. Syst. Meas. 5 (3&4) (2012).

[2] S. International, Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems, SAE International Warrendale, PA, 2016.

[3] L. Claussmann, M. Revilloud, S. Glaser, D. Gruyer, A study on al-based approaches for high-level decision making in highway autonomous driving, in: Systems, Man, and Cybernetics (SMC), 2017 IEEE International Conference on, IEEE, 2017, pp. 3671–3676.

[4] E. Yurtsever, J. Lambert, A. Carballo, K. Takeda, A survey of autonomous driving: Common practices and emerging technologies, 2019, arXiv preprint arXiv:1906.05113.

[5] C. Badue, R. Guidolini, R.V. Carneiro, P. Azevedo, V.B. Cardoso, A. Forechi, L.F.R. Jesus, R.F. Berriel, T.M. Paixão, F. Mutz, et al., Self-driving cars: A survey, 2019, arXiv preprint arXiv:1901.04407.

[6] T. Bandyopadhyay, K.S. Won, E. Frazzoli, D. Hsu, W.S. Lee, D. Rus, Intention-aware motion planning, in: E. Frazzoli, T. Lozano-Perez, N. Roy, D. Rus (Eds.), Algorithmic Foundations of Robotics X, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 475–491.

[7] C. Urmson, J. Anhalt, H. Bae, J.A.D. Bagnell, C.R. Baker, R.E. Bittner, T. Brown, M.N. Clark, M. Darms, D. Demitrish, J.M. Dolan, D. Duggins, D. Ferguson, T. Galatali, C.M. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. Howard, S. Kolski, M. Likhachev, B. Litkouhi, A. Kelly, M. McNaughton, N. Miller, J. Nickolaou, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, V. Sadekar, B. Salesky, Y.-W. Seo, S. Singh, J.M. Snider, J.C. Struble, A.T. Stentz, M. Taylor, W.R.L. Whittaker, Z. Wolkowicki, W. Zhang, J. Ziglar, Autonomous driving in urban environments: Boss and the urban challenge, J. Field Robot. 25 (8) (2008) 425–466, Special Issue on the 2007 DARPA Urban Challenge, Part I.

[8] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt, S. Thrun, Junior: The stanford entry in the urban challenge, in: M. Buehler, K. Iagnemma, S. Singh (Eds.), The DARPA Urban Challenge: Autonomous Vehicles in City Traffic, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 91–123, http://dx.doi.org/10.1007/978-3-642-03991-1_3.

[9] M. Zhang, N. Li, A. Girard, I. Kolmanovsky, A finite state machine based automated driving controller and its stochastic optimization, in: ASME 2017 Dynamic Systems and Control Conference, American Society of Mechanical Engineers Digital Collection, 2017.

[10] N. Li, H. Chen, I. Kolmanovsky, A. Girard, An explicit decision tree approach for automated driving, in: ASME 2017 Dynamic Systems and Control Conference, American Society of Mechanical Engineers, 2017, pp. V001T45A003–V001T45A003.

[11] C.M. Bishop, Pattern Recognition and Machine Learning, springer, 2006.

[12] M. Mukadam, A. Cosgun, A. Nakhaei, K. Fujimura, Tactical decision making for lane changing with deep reinforcement learning, 2017.

[13] E. Galceran, A.G. Cunningham, R.M. Eustice, E. Olson, Multipolicy decision-making for autonomous driving via changepoint-based behavior prediction: Theory and experiment, Auton. Robots 41 (6) (2017) 1367–1382.

[14] P. Wang, C.-Y. Chan, Formulation of deep reinforcement learning architecture toward autonomous driving for on-ramp merge, 2017, arXiv preprint arXiv:1709.02066.

[15] S. Grigorescu, B. Trasnea, T. Cocias, G. Macesanu, A survey of deep learning techniques for autonomous driving, J. Field Robotics (2019).

[16] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L.D. Jackel, M. Monfort, U. Muller, J. Zhang, et al., End to end learning for self-driving cars, 2016, arXiv preprint arXiv:1604.07316.

[17] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT press, 2018.

[18] M.L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, John Wiley & Sons, 2014.

[19] K. Arulkumaran, M.P. Deisenroth, M. Brundage, A.A. Bharath, A brief survey of deep reinforcement learning, 2017, arXiv preprint arXiv:1708.05866.

[20] V. François-Lavet, P. Henderson, R. Islam, M.G. Bellemare, J. Pineau, et al., An introduction to deep reinforcement learning, Found. Trends Mach. Learn. 11 (3–4) (2018) 219–354.

[21] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of Go with deep neural networks and tree search, Nature 529 (7587) (2016) 484.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529.

[23] S. Levine, C. Finn, T. Darrell, P. Abbeel, End-to-end training of deep visuomotor policies, J. Mach. Learn. Res. 17 (1) (2016) 1334–1373.

[24] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, 2015, arXiv preprint arXiv:1509.02971.

[25] J. Kober, J.R. Peters, Policy search for motor primitives in robotics, in: Advances in Neural Information Processing Systems, 2009, pp. 849–856.

[26] J. Kober, J. Bagnell, J. Peters, Reinforcement learning in robotics: A survey, Int. J. Robot. Res. 32 (11) (2013) 1238–1274.

[27] L.-J. Lin, Self-improving reactive agents based on reinforcement learning, planning and teaching, Mach. Learn. 8 (3–4) (1992) 293–321.

[28] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double Q-learning, in: AAAI, vol. 16, 2016, pp. 2094–2100.

[29] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, N. De Freitas, Dueling network architectures for deep reinforcement learning, 2015, arXiv preprint arXiv:1511.06581.

[30] I. Osband, C. Blundell, A. Pritzel, B. Van Roy, Deep exploration via bootstrapped DQN, in: Advances in Neural Information Processing Systems, 2016, pp. 4026–4034.

[31] O. Anschel, N. Baram, N. Shimkin, Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning, in: Proceedings of the 34th International Conference on Machine Learning, vol. 70, JMLR. org, 2017, pp. 176–185.

[32] A.M. Metelli, A. Likmeta, M. Restelli, Propagating uncertainty in reinforcement learning via wasserstein barycenters, in: Advances in Neural Information Processing Systems, 2019, pp. 4335–4347.

[33] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in: International Conference on Machine Learning, 2015, pp. 1889–1897.

[34] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, 2017, arXiv preprint arXiv:1707.06347.

[35] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: International Conference on Machine Learning, 2016, pp. 1928–1937.

[36] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018, arXiv preprint arXiv:1801.01290.

[37] A.E. Sallab, M. Abdou, E. Perot, S. Yogamani, Deep reinforcement learning framework for autonomous driving, Electron. Imaging 2017 (19) (2017) 70–76.

[38] A. Yu, R. Palefsky-Smith, R. Bedi, Deep reinforcement learning for simulated autonomous vehicle control.
[39] P. Wolf, C. Hubschneider, M. Weber, A. Bauer, J. Härtl, F. Dürr, J.M. Zöllner, Learning how to drive in a real world simulation with deep q-networks, in: Intelligent Vehicles Symposium (IV), 2017 IEEE, IEEE, 2017, pp. 244–250.
[40] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, V. Koltun, CARLA: An open urban driving simulator, 2017, arXiv preprint arXiv:1711.03938.
[41] X. Pan, Y. You, Z. Wang, C. Lu, Virtual to real reinforcement learning for autonomous driving, 2017, arXiv preprint arXiv:1704.03952.
[42] R. Liaw, S. Krishnan, A. Garg, D. Crankshaw, J.E. Gonzalez, K. Goldberg, Composing meta-policies for autonomous driving using hierarchical deep reinforcement learning, 2017, arXiv preprint arXiv:1711.01503.
[43] S. Thornton, Autonomous vehicle speed control for safe navigation of occluded pedestrian crosswalk, 2018, arXiv preprint arXiv:1802.06314.
[44] W. Xia, H. Li, B. Li, A control strategy of autonomous vehicles based on deep reinforcement learning, 2016, pp. 198–201, http://dx.doi.org/10.1109/ISCID.2016.2054.
[45] F. Codevilla, M. Müller, A. Dosovitskiy, A. López, V. Koltun, End-to-end driving via conditional imitation learning, 2017, arXiv preprint arXiv:1710.02410.
[46] M. Kuderer, S. Gulati, W. Burgard, Learning driving styles for autonomous vehicles from demonstration, in: Robotics and Automation (ICRA), 2015 IEEE International Conference on, IEEE, 2015, pp. 2641–2646.
[47] J. Zhang, K. Cho, Query-efficient imitation learning for end-to-end autonomous driving, 2016, arXiv preprint arXiv:1605.06450.
[48] D. Silver, J. Bagnell, A. Stentz, Learning autonomous driving styles and maneuvers from expert demonstration, in: Experimental Robotics, Springer, 2013, pp. 371–386.
[49] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, B. Boots, Agile off-road autonomous driving using end-to-end deep imitation learning, 2017, arXiv preprint arXiv:1709.07174.
[50] D.A. Pomerleau, Alvinn: An autonomous land vehicle in a neural network, in: Advances in Neural Information Processing Systems, 1989, pp. 305–313.
[51] P. Abbeel, A.Y. Ng, Apprenticeship learning via inverse reinforcement learning, in: Proceedings of the Twenty-First International Conference on Machine Learning, ACM, 2004, p. 1.
[52] J. Audiffren, M. Valko, A. Lazaric, M. Ghavamzadeh, Maximum entropy semi-supervised inverse reinforcement learning, in: IJCAI, 2015, pp. 3315–3321.
[53] J. Liu, P. Hou, L. Mu, Y. Yu, C. Huang, Elements of effective deep reinforcement learning towards tactical driving decision making, 2018, arXiv preprint arXiv:1802.00332.
[54] S. Shalev-Shwartz, S. Shammah, A. Shashua, Safe, multi-agent, reinforcement learning for autonomous driving, 2016, arXiv preprint arXiv:1610.03295.
[55] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, K. Fujimura, Navigating occluded intersections with autonomous vehicles using deep reinforcement learning, 2017, arXiv preprint arXiv:1705.01196.
[56] C. Paxton, V. Raman, G.D. Hager, M. Kobilarov, Combining neural networks and tree search for task and motion planning in challenging environments, 2017, arXiv preprint arXiv:1703.07887.
[57] D. Isele, A. Nakhaei, K. Fujimura, Safe reinforcement learning on autonomous vehicles, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2018, pp. 1–6.
[58] D. Chen, L. Jiang, Y. Wang, Z. Li, Autonomous driving using safe reinforcement learning by incorporating a regret-based human lane-changing decision model, 2019, arXiv preprint arXiv:1910.04803.
[59] Z. Qiao, K. Muelling, J.M. Dolan, P. Palanisamy, P. Mudalige, Automatically generated curriculum based reinforcement learning for autonomous vehicles in urban environment, in: 2018 IEEE Intelligent Vehicles Symposium (IV), IEEE, 2018, pp. 1233–1238.
[60] Y. Jaafra, J.L. Laurent, A. Deruyver, M.S. Naceur, Meta-Reinforcement Learning for Adaptive Autonomous Driving.
[61] X. Liang, Y. Liu, T. Chen, M. Liu, Q. Yang, Federated transfer reinforcement learning for autonomous driving, 2019, arXiv preprint arXiv:1910.06001.
[62] Z. Xu, C. Tang, M. Tomizuka, Zero-shot deep reinforcement learning driving policy transfer for autonomous vehicles based on robust control, in: 2018 21st International Conference on Intelligent Transportation Systems (ITSC), IEEE, 2018, pp. 2865–2871.
[63] M. Müller, A. Dosovitskiy, B. Ghanem, V. Koltun, Driving policy transfer via modularity and abstraction, 2018, arXiv preprint arXiv:1804.09364.
[64] J. Kim, J. Canny, Interpretable learning for self-driving cars by visualizing causal attention, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 2942–2950.
[65] W. Zeng, W. Luo, S. Suo, A. Sadat, B. Yang, S. Casas, R. Urtasun, End-to-end interpretable neural motion planner, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 8660–8669.
[66] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, J. Schmidhuber, Policy gradients with parameter-based exploration for control, in: International Conference on Artificial Neural Networks, Springer, 2008, pp. 387–396.
[67] J. Garcıa, F. Fernández, A comprehensive survey on safe reinforcement learning, J. Mach. Learn. Res. 16 (1) (2015) 1437–1480.
[68] R.S. Sutton, D.A. McAllester, S.P. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, in: Advances in Neural Information Processing Systems, 2000, pp. 1057–1063.
[69] J. Peters, S. Schaal, Reinforcement learning of motor skills with policy gradients, Neural Netw. 21 (4) (2008) 682–697.
[70] M.P. Deisenroth, G. Neumann, J. Peters, et al., A survey on policy search for robotics, Found. Trends Robot. 2 (1–2) (2013) 1–142.
[71] A.B. Owen, Monte Carlo Theory, methods and examples, 2013.
[72] A.M. Metelli, M. Papini, F. Faccio, M. Restelli, Policy optimization via importance sampling, in: Advances in Neural Information Processing Systems, 2018, pp. 5442–5454.
[73] D. Krajzewicz, J. Erdmann, M. Behrisch, L. Bieker, Recent development and applications of SUMO - simulation of urban mobility, International Journal On Advances in Systems and Measurements 5 (3&4) (2012) 128–138.
[74] J. Effertz, Autonomous driving in urban environments by combining object and grid based perception models, ATZelektronik worldwide 5 (2) (2010) 40–45, http://dx.doi.org/10.1007/BF03242264.
[75] T. Osa, J. Pajarinen, G. Neumann, J.A. Bagnell, P. Abbeel, J. Peters, et al., An algorithmic perspective on imitation learning, Found. Trends Robot. 7 (1–2) (2018) 1–179.

Amarildo Likmeta was born on August 16th, 1995, in Durres, Albania. He received a Masters of Science in Computer Science and Engineering (with honors) from Politecnico di Milano, in 2018. He worked as a researcher in Autonomous He is currently pursuing a joint Ph.D. Degree in Data Science and Computation at the University of Bologna and Politecnico di Milano. His main research interests include Autonomous Decision Making, Reinforcement learning and Planning.

Alberto Maria Metelli was born on March 12th, 1993, in Trescore Balneario (BG), Italy. In July 2015, he obtained the Bachelor of Science in Computer Engineering at Politecnico di Milano (with honors). In July 2017, he obtained the Master of Science in Computer Science and Engineering at Politecnico di Milano (with honors), defending the thesis "Compatible Reward Inverse Reinforcement Learning". Since November 2017, he is a Ph.D. student at the Department of Electronics, Information and Bioengineering (DEIB) of the Politecnico di Milano. His main research interests include Artificial Intelligence and Machine Learning, in particular, Reinforcement Learning.

Andrea Tirinzoni is a Ph.D. candidate in computer science at Politecnico di Milano. In 2017, he received a Master of Science in Computer Science and Engineering from Politecnico di Milano (110 summa cum laude) and a Master of Science in Computer Science from University of Illinois at Chicago (GPA 4.0/4). His main research interest is reinforcement learning, with a particular focus on transfer learning in structured MDP and bandit problems.

Riccardo Giol received the Master in Computer Science and Engineering (with honor) from Politecnico di Milano, Italy, in 2019. During the studies, he specialized in Artificial Intelligence, Data Analysis, and Machine Learning, focusing his research in Reinforcement Learning. Since September 2019 he is working as System Developer in the area of Sustainable Indoor Farming.

Marcello Restelli is Associate Professor of Computer Science and Engineer at the "Dipartimento di Elelttronica, Informazione e Bioingegneria" of the Politecnico di Milano. where he obtained the Laurea degree in Computer Science Engineering in 2000 and the Ph.D. in Information Engineering in 2004. From 2008 to 2019. He is currently teaching the "Machine Learning" and the "Information Retrieval and Data Mining" courses and he is a board member of the Ph.D. program in "Data Science and Computation". His research interests focus on machine learning algorithms and, in particular, the development of reinforcement learning techniques and their application to real-world problems (e.g., robotics, finance, autonomous vehicles, water resource management, etc.).

He has published more than 100 peer-reviewed papers on some of the most prestigious international conferences and journals in the machine-learning and robotics fields. He has served as a reviewer for several international journals and he has been a member of the program committee of the main international conference of his research area, among which NIPS, AAAI, and IJCAI. He is the principal investigator of several research projects funded both by public entities and by some of the main Italian companies.

Danilo Romano received the M.Sc (with honor) in Computer Engineering from the University of Salerno, Italy in 2016 and worked as researcher on Artificial Intelligence and Computational Neuroscience until 2017. He currently works as Decision Making Designer at the Automated Driving Technologies group of Marelli, Italy. His work is focused in designing decision maker for autonomous driving combining AI techniques with traditional approaches.