A Reinforcement Learning Approach for Enacting Cautious Behaviours in Autonomous Driving System: Safe Speed Choice in the Interaction With Distracted Pedestrians

Gastone Pietro Rosati Papini, Alice Plebe, Mauro Da Lio, *Member, IEEE*, and Riccardo Donà

***Abstract*— Driving requires the ability to handle unpredictable situations. Since it is not always possible to predict an impending danger, a good driver should preventively assess whether a situation has risks and adopt a safe behavior. Considering, in particular, the possibility of a pedestrian suddenly crossing the road, a prudent driver should limit the traveling speed. We present a work exploiting reinforcement learning to learn a function that specifies the safe speed limit for a given artificial driver agent. The safe speed function acts as a behavioral directive for the agent, thus extending its cognitive abilities. We consider scenarios where the vehicle interacts with a distracted pedestrian that might cross the road in hard-to-predict ways and propose a neural network mapping the pedestrian's context onto the appropriate traveling speed so that the autonomous vehicle can successfully perform emergency braking maneuvers. We discuss the advantages of developing a specialized neural network extension on top of an already functioning autonomous driving system, removing the burden of learning to drive from scratch while focusing on learning safe behavior at a high-level. We demonstrate how the safe speed function can be learned in simulation and then transferred into a real vehicle. We include a statistical analysis of the network's improvements compared to the original autonomous driving system. The code implementing the presented network is available at https://github.com/tonegas/safe-speed-neural-network with MIT license and at https://zenodo.org/communities/dreams4cars.**

***Index Terms*— Vulnerable road users, neural networks, reinforcement learning, transfer learning, autonomous driving, intelligent speed adaptation.**

I. Introduction

DRIVING is a task carried out in partially unpredictable environments: it is usually possible to predict changes in the environment and other agents' intentional behaviors with reasonable confidence but unexpected events may occasionally happen. For example, other road users may behave in surprising ways: they may suddenly change their mind or be distracted. Another example is when objects appear from behind occlusions.

So, while prediction accuracy is paramount for efficient driving, an autonomous driving system (ADS) must nevertheless assume that unexpected events may happen because of non-rational behavior of the other road participants or because of limitations of the system's prediction abilities (which includes imperfect perception and possibly inaccurate prediction). Thus, a safe approach to driving must be robust against these unknowns.

A. What This Work is (and is not) About

This paper deals with extending the cognitive abilities of a given driver agent. Specifically, we adopt a *layered control paradigm* where a new layer acting on top of the agent's sensorimotor system (with its perceptual, cognitive, and motor limits) steers the agent towards *cautious speed choices*.

There are other literature studies addressing a similar problem, which are compared in Section VIII. The difference is that we do not learn the whole sensorimotor stack but only a function that restricts the agent's choices. This process is similar to instructing a human driver to recognize potentially dangerous contexts in the environment and limit his/her personal driving speed accordingly.

The *safe speed* is the maximum speed that permits successful collision avoidance if a pedestrian exhibits hard-to-predict or *unpredictable* movements. Our goal is to define a mapping between the context (presence and state of pedestrians) and a self-defined speed limit that overrides the posted speed limit, thus effectively enhancing the cognition abilities of the agent.

Learning cautious behavior is formulated as an *action-selection learning* problem [1]. We seek to mimic the putative mechanism occurring in the vertebrate brain, where the basal ganglia learn —via reinforcement— to minimize the likelihood of undesirable events by inhibiting dangerous affordances.

In our implementation, the driver agent produces the pool of affordable actions. They include different types of motion primitives, such as free-flow primitives that permit to reach different final speeds and obstacle avoidance primitives of various types. Usually, the agent chooses among these admissible

Manuscript received August 21, 2020; revised February 23, 2021 and May 13, 2021; accepted May 18, 2021. This work was supported by the EU Horizon 2020 Dreams4Cars Research and Innovation Action supported by the European Commission under Grant 731593. The Associate Editor for this article was R. Arghandeh. *(Corresponding author: Gastone Pietro Rosati Papini.)*

The authors are with the Department of Industrial Engineering, University of Trento, 38123 Trento, Italy (e-mail: gastone.rosatipapini@unitn.it; alice.plebe@unitn.it; mauro.dalio@unitn.it; riccardo.dona@unitn.it).

This article has supplementary downloadable material available at https://doi.org/10.1109/TITS.2021.3086397, provided by the authors.

Digital Object Identifier 10.1109/TITS.2021.3086397

actions (e.g., the fastest collision-free trajectory withstanding the posted speed limit). The learning problem consists of discovering the fastest free-flow primitive that is safe in pedestrians' vicinity, to exclude faster ones from the admissible choices. This effectively limits the maximum traveling speed without binding the agent to one particular choice: the agent will still choose the most appropriate trajectory but restricted the safe ones. This learning problem is built upon and extends the agent's sensorimotor system: by exploiting the existing agent's perception-action abilities, the learning problem is simplified and more efficient. Furthermore, the agent learns cautious behaviors tuned to the agent's cognitive abilities (if the agent improves its prediction-control skills, then higher speed choices are possible).

In turn, the inhibition mechanism used to pre-filter the affordable actions introduces modularity: the inhibition for many pedestrians is obtained by superimposing (the set union) the inhibitions of each pedestrian (which corresponds to assuming that the safe speed is the minimum of the safe speeds evaluated for each pedestrian).

In summary, the focus of the paper is on demonstrating a mechanism for the extension of a driver agent with cautious behaviors. Improving path prediction is not the goal of this paper (it still falls within the predictable intention case, which we assume given with the agent). Instead, it is about learning cautious behavior when intentions are not predictable or the predicting models fail. Also, algorithmic extensions of the reinforcement learning method in itself are not our primary goal.

B. Driver Agent

In our work, the driver agent is [2], but other realizations may also work, and we release an open access implementation of this work. The agent can drive [3], i.e., it is capable of high-level motor planning and low-level control. Specifically, it predicts the other road users' (pedestrians) trajectories with a *mirroring* mechanism (see also [4, Section IV.A and Section V.A]) and maneuvers accordingly to avoid collisions. In case of pedestrian deviations, the agent reacts with updated maneuvers. However, in case of late unexpected deviations, the resulting maneuvers —now aimed at collision mitigation— may be unsatisfactory (see Section VIII-A).

Cautious behavior is obtained by training a neural network that maps the pedestrians' context into a safe speed, that is in turn used to pre-filter the agent's affordable actions. The network, called *safe speed neural network* (SSNN), is trained by deep reinforcement learning on a set of simulated episodes involving pedestrians crossing the street in unpredictable ways, as Fig. 1 shows.

Although the framework is trained in a simulated environment, we demonstrate that the policy can be transferred into a real vehicle without additional training. Given the modular nature of our approach, the framework can account for various assumptions about pedestrian mobility and various models of the driven vehicle dynamics (actuation delays, braking capacity, etc.).

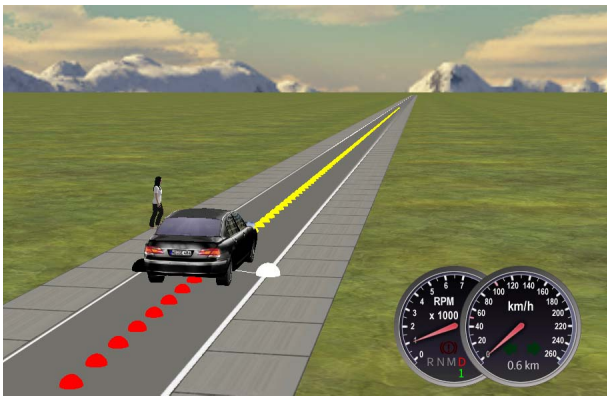


Fig. 1. The simulation environment of OpenDS showing the interaction between the vehicle and the pedestrian.

The rest of the paper is organized into eight sections. Section II presents the most relevant work on the interaction with vulnerable road users and points out the major differences with our approach. Section III introduces the reinforcement learning algorithm adopted here, and it describes the architecture of the neural network and the scenario of simulation. Section IV illustrates how the neural network is integrated with the underlying ADS (Fig. 2), and it details the training procedure. Section V presents the results obtained in simulations and a statistical comparison of the performance with and without the network, while Section VI presents the experiments carried out on a real vehicle. Section VII describes a variation of the proposed algorithm that dispenses with the incremental adjustments in the computation of the safe speed. Section VIII discusses the obtained results, the similarities and differences with the related work, and the key contributions offered by our approach. Lastly, Section IX draws the conclusion and discusses future developments.

II. Related Work

A popular approach to deal with vulnerable road users—in particular pedestrians—is to try to predict their trajectory. There is a whole body of science on pedestrian path prediction; a comprehensive review on the topic can be found in [5]. In recent years, the prediction of human motion has received increased attention across several communities, and it is widely considered a critical task for self-driving vehicles [6]. However, this approach has a fundamental limitation: it is applicable only if the pedestrian behaves intentionally. When predicting a pedestrian's trajectory, it is hardly possible to take into account the complexity of human behavior and the variety of its internal and external stimuli. There may be no way to predict the pedestrian's intentions if they are distracted or suddenly change their mind.

An alternative approach that avoids this limitation is assessing the risk of a given situation and determining the best traveling speed. This strategy lies within the broader research field on *intelligent speed adaptation* (ISA). The first ISA systems date back to the early 2000s. One of the earliest works [7] exploits GPS localization and a given map defining the environment's risk levels. Another study [8] assesses the risk

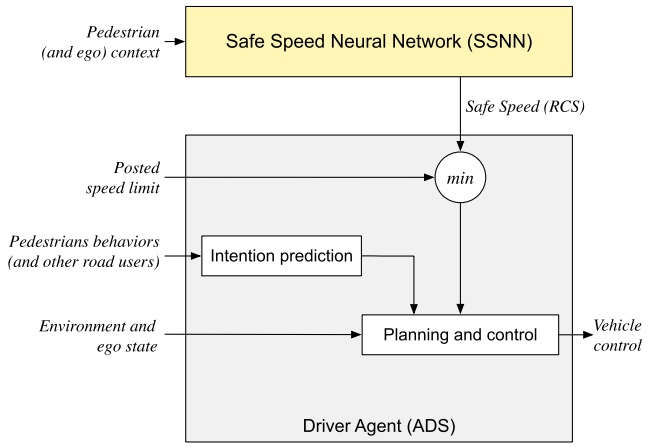


Fig. 2. System block diagram. The given driver agent (ADS) plans trajectories and controls the vehicle based on the environment and ego state perception. For planning, the agent includes a module for predictions of the other road users' intentions. The safe speed neural network (SSNN) on top learns (via offline reinforcement learning simulations) associations between the pedestrians' context and a safe speed recommendation (RCS). The latter overrides the posted speed limit.

based on some environmental factors: type of road, time of the day, speed limit, and country. Most of these early proposals are significantly simplified, as they involve just static methods, not taking into account momentary situations that may occur.

Recent studies on ISA exploit machine learning to perform dynamic evaluations of the current risk. For example, [9] proposes to determine the level of risk and the proper speed of travel using a vision system based on deep CNNs. The system requires supervised training, and it is tested on an annotated dataset provided by the same authors. This solution has the disadvantage of requiring the dataset to represent exhaustively the possible situations that may occur. Moreover, the annotations are generated according to the driving style of the human drivers employed when collecting the dataset; hence, the speed determined by the network is inevitably biased by these human factors.

A crucial issue affecting supervised approaches to ISA is the difficulty of defining the correct safe behavior and having an exhaustive set of examples of correct human behavior—assuming that the human behavior is optimal. A valid alternative to supervision is the adoption of reinforcement learning (RL), which is recently gaining more attention in robotics and autonomous driving [10]–[13].

Reinforcement learning is a general framework defining the interaction of an RL agent[1] with its environment. RL problems can be formalized as optimal control of incompletely-known Markov decision processes [14]. Simultaneously, reinforcement learning has significant connections with cognition, particularly associative learning [15] and neuroscience, in relation with reward circuits of the brain [16].

In recent years, several studies have adopted reinforcement learning to design complex tasks for autonomous vehicles, such as navigating urban environments and interacting with other road users. In this context, it is possible to distinguish between the approaches addressing the interaction with other vehicles and the approaches focusing on pedestrians' interaction. Most work on the interaction with other vehicles adopts reinforcement learning to control the vehicle in the proximity of intersections [17]–[19]. The RL agent generally produces discrete actions for steering and acceleration based on information about the surrounding vehicles and the road topology.

The studies focusing on the interaction with pedestrians generally use reinforcement learning to implement an autonomous braking system [20]–[23]. In this case, the RL network receives a description of the pedestrian's location and heading and decides to brake or accelerate depending on the risk of collision with the pedestrian (some works also consider the passenger's comfort).

An interesting work by Chen *et al.* [24] presents an end-to-end RL motion planning method that aims to deal with emergency situations, including the case of a pedestrian rushing into the lane unexpectedly. The work proposes to use an additional generative model with a VAE-GAN architecture to create virtual emergency situations; the module produces multiple video clips that are variations of the current real scene with the addition of dangerous elements. The planning model makes multiple plans in parallel for each of the virtual emergencies generated. Then, the final planning decision is determined through the analysis of both current and virtual scenes.

Our approach and the works reviewed so far differ in the purpose of reinforcement learning. In all the works mentioned above, the RL agent itself is responsible for driving the vehicle (it is trained to control the acceleration and the steering angle). Conversely, our approach adopts a layered arrangement (Fig. 2): at the bottom, a given driver agent controls the vehicle. At the top, an additional layer realizes cautious behavior by suggesting a prudent speed according to the potential risk of the situation. We separately assign the prediction of intention to the driver agent and the management of unpredictable events to the top layer. Section VIII discusses some implications of this division of tasks.

The learning process follows the scheme used for bootstrapping subsumption architectures, i.e., adding new layers that create new behaviors by re-using the existing abilities [25]. Fig. 3 shows the situation before training the safe speed function on the top and when the trained safe speed layer becomes part of an evolved agent at the bottom. In the following, we will refer to the individual components before integration with the terms: "ADS" for the original autonomous driving system without the safe speed ability and "SSNN" for the layer (to be) that produces the new sought safe speed behavior.

The method of Chen *et al.* [24] offers an attractive strategy to tackle the problem of dealing with unexpected situations. In cognitive terms, the generation of the hypothetical dangers is a form of sensor/perception *imagery carried out online*, i.e., during the course of the real action. By reusing motor planning algorithms on the hallucinated perception, the agent

[1]In Reinforcement Learning, the term "agent" usually indicates the policy-maker. To avoid confusion with the driver agent (Fig. 2), we use the term "RL agent" in this work.

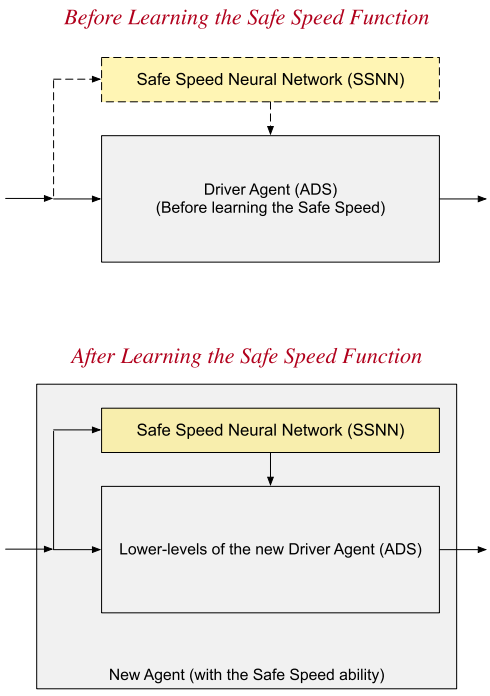


Fig. 3. The learning of a safe speed behavior follows the method used to extend subsumption architectures: a given agent (ADS) is enhanced by developing a new layer (SSNN) that subsumes the original agent's abilities, resulting in a new agent with extended sensorimotor abilities.

can verify evasive strategies and amend the current motion plan if necessary. With this respect, our approach can be seen as a form of *offline sensorimotor imagery*. Simulations about possible dangerous situations are carried out at a deferred time. They are used to *learn associations* between context and safe speed (that allow jumping to the conclusion without carrying out imagery online). Human beings possess both forms of imagery: offline imagery is typically carried out during the sleep state, e.g., [26].

To sum up, the key features that distinguish our approach from the related work are the following:

- a network is developed on top of an existing autonomous driving system; the latter is responsible for controlling the vehicle while the former suggests a safe speed depending on the potential risk of the situation;
- integrating the safe speed network with the driver agent is a practical and effective way to improve the agent with an additional high-level cautious behavior;
- since the network is not required to learn to drive the vehicle; it features a simpler reward function and faster convergence time;
- the risk assessment performed by the network complements the agent's predictions; addressing unexpected situations, like when a pedestrian is distracted or abruptly chooses to cross the street;
- to train the network, it is not necessary to employ annotated data of human behavior or complex generative models of emergencies; the simple yet effective simulation events need to include pedestrians moving along partially random paths across the road.

III. Safe Speed Neural Network

A distinguishing feature of our work is combining the driving capabilities of an autonomous driving system (ADS) with the separate module called *safe speed neural network* (SSNN) that assesses the risk of a pedestrian context (Fig. 2).

The collaboration between the two modules occurs through a high-level behavioral specification called *requested cruising speed* (RCS). The RCS influences the speeding behavior of the ADS. It is the highest safe speed and a "desired speed" for the systems (the agent may speed up to that). The objective of the SSNN is to adjust the RCS to ensure that the ADS can manage an emergency in complete safety. We train the SSNN using deep reinforcement learning and a driving simulator as follows.

A. Introduction to Q-Learning

Several components define an RL problem. First, there is a finite set $\mathcal{S}$ of possible states s representing the environment. The RL agent can carry out a finite set $\mathcal{A}$ of possible actions a. When the agent executes action a, the state s changes to s' (the transition function), and a reward $r(s, s', a)$ is obtained by the agent.

The RL agent seeks to carry out optimal actions by "reinforcing" the choices that maximize long-term accumulated rewards. The choice logic is formalized as the policy $a = \pi(s)$, a decision-making rule for selecting actions given the state.

A RL problem is a stochastic optimal control problem, solved by finding an optimal policy $\pi^*(s)$. A reward function $r(s, s', a)$ may be specified to define the optimality criterion and, hence, the learning goal.

RL exists in two different varieties: *model-based* and *model-free*. In *model-based* RL, the agent must first capture a model representing the environment, made of the transition function and the reward function. *Model-free* RL, instead, does not require learning a model of the environment and focuses on learning the action-value function, as shown below. Model-based approaches are more general and reliable, but they become impractical as the state space, and action space dimensions grow.

One of the most popular model-free RL algorithms is *Q-learning*, introduced by Watkins in 1989 [27], [28]. In Q-learning, the action-value function can be expressed without any knowledge of the transition function as follows:

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right], \qquad (1)$$

where $\mathbb{E}_\pi[\cdot]$ is the mathematical expectation of the accumulated future discounted-rewards using the policy π, and where $s_t = s$ is the current state of the system at time t, $a_t = a$ the current action chosen by the agent, r_{t+k+1} the rewards, and γ is a discount factor for rewards that gradually refer to more distant times.

Recently, Q-learning has been rediscovered in conjunction with deep learning under the name of *deep Q-learning* (DQL) by the DeepMind team [29] in the context of solving the classic Atari 2600 games. DQL approximates function $Q(s, a)$ of (1) with a deep neural network.

The approximant network is trained to minimize the mean square error between the output $Q(s_t, a_t)$ and the "target" y_t:

$$y_t = r_{t+1} + \gamma \max_{a' \in \mathcal{A}} \widetilde{Q}\left(s_{t+1}, a'\right). \tag{2}$$

Notably, in this process the output $Q(s_t, a_t)$ is computed by a network with parameters θ and the target y_t is computed by a different network $\widetilde{Q}$ with same structure but different parameters $\widetilde{\theta}$.

Network $\widetilde{Q}$ is called the *target network* and is generated by copying the parameters θ every T steps from network Q. This delayed update reduces the correlation between $\widetilde{Q}$ and Q in the computation of the loss function and improves training effectiveness.

The DQL approach adopts the ε-*greedy* policy [27], which balances exploration and exploitation as follows:

- a random action is chosen with probability ε;
- otherwise the action with currently maximum estimated reward is chosen.

The value of the ε changes during the training epochs τ according to the rule:

$$\begin{aligned} \varepsilon_0 &= 1, \\ \varepsilon_{\tau+1} &= \max(\lambda\, \varepsilon_\tau, \varepsilon^*) \quad \forall\, \tau \in \mathbb{N}, \end{aligned} \tag{3}$$

where λ is a decreasing factor, and ε^* is the minimum accepted value. Thus, the initial phase of training favours exploration by selecting random actions more frequently, whereas the final phase (when the network is improved) focuses on exploitation by selecting actions with maximum estimated reward.

B. The Adopted Algorithm

Our implementation adopts an improved version of the DQL algorithm and a more sophisticated update technique for the network weights.

The Q-learning algorithm suffers from overestimating the action-value function, which happens when using the same action-value function to select actions and evaluate action rewards. Van Hasselt addressed the problem with a new algorithm called *double Q-learning* [30], later combined with deep neural networks as *double DQL* [31]. The method decouples action choice from reward evaluation by using two separate action-value functions. The first network with weights θ learns the action-value function $Q(s, a)$ used for action choice. A second network with weights $\hat{\theta}$ learns the function $\hat{Q}(s, a)$ used to evaluate the rewards. The target is now the following:

$$y_t = r_{t+1} + \gamma\, \hat{Q}\left(s_{t+1}, \operatorname*{argmax}_{a' \in \mathcal{A}} Q(s_{t+1}, a')\right). \tag{4}$$

We choose to update the weights of network $\hat{Q}$ with a "soft update" technique [32] based on Polyak averaging [33]:

$$\hat{\theta}_{\tau+1} = (1 - \nu)\hat{\theta}_\tau + \nu\theta_\tau, \tag{5}$$

where ν is the rate of averaging.

To sum up, we use a model-free double DQL approach. However, note that the network is integrated with the ADS (Fig. 2), predicting the next vehicle state.

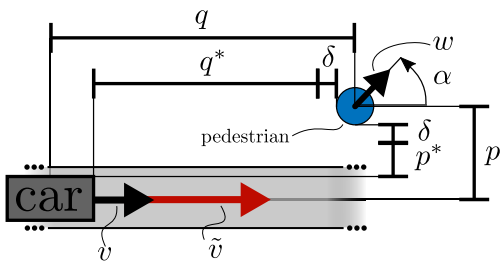


Fig. 4. Schematic representation of the relevant information for the scenario.

Hence, with the layered arrangement, it is like working with a model-based system.

In this paper, we focus specifically on the development of the SSNN. For further details on the implementation of the ADS component, please refer to [2]. Note that we will use the terms RL and (double) DQL interchangeably for the rest of the article.

C. Architecture of the Network

The networks for $Q(s, a)$ and $\hat{Q}(s, a)$ use the same following architecture (Fig. 5).

First, since actions $\mathcal{A} = \{-1, 0, 1\}$ are discrete and enumerable, instead of creating a scalar function for s and a, the networks realise a vector function $\langle Q(s, -1), Q(s, 0), Q(s, 1)\rangle$ for s, collecting the three action values in one output vector.

The input signals are six:

1) v, the speed of the vehicle,
2) $\widetilde{v}$, the requested cruising speed,
3) q^*, the longitudinal gap between pedestrian and vehicle in curvilinear abscissa (Fig.4),
4) p^*, the lateral gap between pedestrian and vehicle in curvilinear ordinate (Fig.4),
5) α, the orientation of the pedestrian relative to the road (zero when walking in the same direction of the traffic, Fig.4),
6) w, the speed of the pedestrian.

Note that for practical reasons, the lateral displacement and the pedestrian's orientation are symmetric to the road's centerline, so the orientation is always positive when the pedestrian moves away from the road. Hence, the state at time t is the vector:

$$s_t = \langle v_t,\, q_t^*,\, p_t^*,\, \alpha_t,\, w_t,\, \widetilde{v}_t\rangle. \tag{6}$$

To make the network independent of the sizes of the vehicle and the pedestrian, we use the longitudinal and lateral gaps, defined as follows (Fig.4):

$$\begin{aligned} q^* &= q - \frac{L + R}{2} - \delta, \\ p^* &= |p| - \frac{W + R}{2} - \delta, \end{aligned} \tag{7}$$

where q and p are the longitudinal and lateral differences between the curvilinear coordinates of the centers of vehicle and pedestrian, $L = 4.4$ m and $W = 1.8$ m are the length and the width of the vehicle, $R = 0.5$ m is the diameter of the

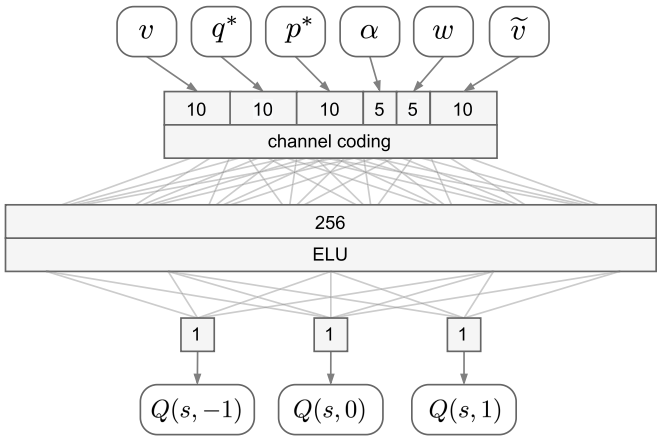


Fig. 5. Architecture of the SSNN.

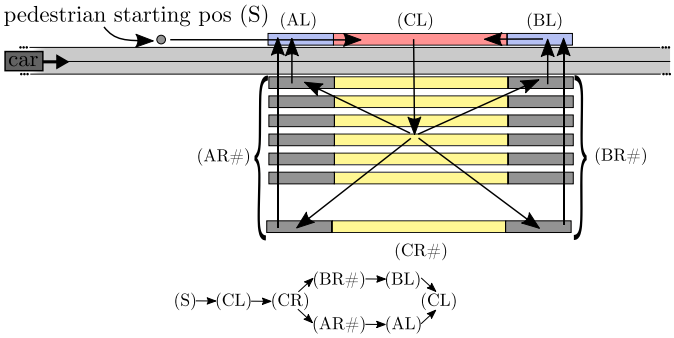


Fig. 6. Schematic representation of the pedestrian movements during the simulation loops.

circular area occupied by the pedestrian, and $\delta = 0.5$ m is a safety margin. The vehicle is considered in collision with the pedestrian when $2 * (-\frac{L+R}{2} - \delta) \leq q^* \leq 0 \ \wedge \ p^* \leq 0$, which notably includes the gaps δ.

The network's output vector $\langle Q(s,-1), Q(s,0), Q(s,1)\rangle$ collects the values of the three actions, $\mathcal{A} = \{-1, 0, 1\}$, respectively meaning: decrease, keep or increase the RCS in steps as follows:

$$\widetilde{v}_{t+1} = \widetilde{v}_t + \Delta_{\widetilde{v}}\, a_t, \tag{8}$$

where $\widetilde{v}_t$ is the current value of the RCS, $\Delta_{\widetilde{v}} = 0.25$ m/s is the unit of variation of the RCS, and $a_t \in \mathcal{A}$ is the action chosen by the network.

The rest of the network comprises two layers, the first of which is implemented using the *channel coding* technique [34]–[37]. This technique expands a scalar value from a single-neuron representation into a vector using a cumulative activation so that increasing values of the scalar input are converted into progressive activation of the coding channels. Each input is encoded by 10 channels, except for α and w, which are encoded with 5 (a total of 50 input neurons). Channel coding offers a convenient way to learn piecewise models, partitions the input space without suffering from unbalanced data, and the network also learns disentangled channels. The second layer of the network is a standard fully-connected layer made of 256 neurons followed by the exponential linear unit (ELU) activation function. The ELU worked better than RELU and hyperbolic tangent [38]. The output is a fully connected linear layer of three neurons. Fig. 5 depicts the architecture of the network.

D. Scenario of Simulation

The objective of the SSNN can be considered twofold. On the one hand, the network must identify the safe speed that allows the vehicle to perform a successful emergency brake maneuver if the pedestrian crosses the road. On the other hand, the safe speed must not be overly cautious so that the vehicle can always travel as fast as the driving context permits.

The safe speed is determined by the level of risk of the situation, which depends on the several factors described in the previous Section III-C. Among these factors, the possible behaviors and the mobility that we assume for pedestrians is prominent. In real traffic situations, pedestrians usually do not interfere in dangerous ways with the vehicle's potential paths. This might suggest considering a realistic distribution of crossing and non-crossing pedestrians. However, the purpose of the safe speed network is to support the ADS in handling rare situations. Hence, we choose to simulate only pedestrians that are very inclined to cross the road. Moreover, given that the RL algorithm learns by experience, the network needs to witness an extensive amount of dangerous situations to learn within a reasonable training time.

The simulation environment used to train the network is OpenDS,[2] an open-source driving simulator based on the JMonkeyEngine game engine. Fig. 1 shows the simulated situation for the interaction between the vehicle and a pedestrian as visualized by OpenDS.

1) Setup of the Scenario: the environment is composed of a straight road with a single lane 2.6 m wide. The pedestrian's initial position is 660 m ahead of the vehicle to let the vehicle reach the initial RCS before getting close to the pedestrian. The initial RCS is randomly set between 1.5 m/s and 12.5 m/s with a granularity of 1 m/s. The safe speed network is activated when $q^* < 200$. From this point on, in the training phase it is called every 0.2 seconds, although the driver agent replanning rate is faster (20 Hz). The simulation ends successfully when the car reaches the end of the road ($q^* \leq -100$). It ends unhappily when either the car hits the pedestrian or the car trespasses the lane marking to avoid the collision.

2) Behaviour of the Pedestrian: the pedestrian follows paths that cross the road randomly in a continuous loop. The possible walking directions are structured into different "rails" (Fig. 6). There are three rails on the left side of the road (AL, CL, BL) and 21 rails on the right side (AR#, CR#, BR#). The lateral distance between the center of the road and the left rails (AL, CL, BL) is 2.3 m, which is the same for the first line of right rails (AR1, CR1, BR1). The total width of the road is 2.6 m. The lateral separation between each rail line on the right is 2 m, except for the last three that are 7 m apart. There are seven lines on the right; the last one stands at 19.3 m from the road center. The central rails (CL and CR#) are 20 m long,

[2]https://opends.dfki.de

while the peripheral rails (AL, AR#, BL, BR#) are 10 m long. The pedestrian's speed is constant while walking along one rail or across the rails and road, but it varies when switching direction, assuming a random value in the range 0.55 m/s to 3.33 m/s (2 to 12 km/h).

The pedestrian switches of direction are unpredictable: he/she starts from the initial position (S) at 7 m from (AL) and moves towards the red rail (CL). Inside this rail, the pedestrian at any moment may cross the street perpendicularly towards one yellow rail (CR#). He/she then moves to a random point of either the (AR#) or (BR#) rails, even in diagonal directions. Once there, the pedestrian crosses the road perpendicularly, reaching a corresponding rail on the opposite side (AL) or (BL). Finally, the pedestrian walks again towards the red rail (CL), and the loop starts over.

3) Reward Function: since we choose a model-free approach, the RL agent does not need to learn the reward function as we define it. At each training step (0.2 s), the reward is the following:

$$r_t = \begin{cases} 0.5 & \text{if the episode ends with success,} \\ -v_t & \text{if the episode fails,} \\ -0.01 & \text{otherwise.} \end{cases} \tag{9}$$

The episode fails when the vehicle reaches the lane markings (off the road) or hits the pedestrian. It succeeds when the vehicle reaches the end of the road at $q^* \leq -100$ m.

The motivations for the form given to (9) are elaborated in Section VIII-C (they can be summarised here as seeking to achieve minimum travel time while tolerating rare occurrence of very-low speed unharmful collisions).

IV. Integration With the ADS

The collaboration between the ADS and the SSNN happens in a framework composed of two asynchronous interacting processes: the neural process and the simulation process, pictured in Fig. 7 (a) with the blue boxes. The neural process contains the DQL core implementing the double deep Q-learning algorithm of Section III-B. The simulation process consists of two sub-modules: the OpenDS simulator modeling vehicle dynamics and environment, and the controller of the vehicle. The latter is, in turn, composed of the ADS itself and of a DQL middleware, which is responsible for setting the RCS.

The processes store training-related information in three files, represented in Fig. 7 (a) with green cylinders: the file network containing the weights of the network and the current value of ε; and two files RB common and RB rare used as replay buffers. The replay buffers annotate simulated episodes as sequences of tuples in the form $\langle s_t, a_t, r_t, s_{t+1} \rangle$. The buffers are finite and forget the oldest annotations as new simulations are produced. The framework keeps two separate buffers, where RB rare prevents forgetting the crucial situations corresponding to failed episodes.

The DQL core is responsible for the training of the neural network. It generates the training set by sampling the tuples from the replay buffers and, at each epoch of training, it overwrites the network file by updating the weights of Q and $\hat{Q}$, defined in Section III-B, and the value of ε. The DQL middleware comes into play in the simulation process. At the beginning of a simulation loop, it loads the most recent weights of the network and the value of ε from the network file. Then, it infers the next action to be executed using the ε-*greedy* policy and saves the annotations of the episode inside the replay buffers.

TABLE I
Hyper-Parameters Used for Training

Parameter	Value
number of epochs	30000
mini-batch size	333
% of data from rare events	5%
gradient descent optimizer	ADAM
learning rate	1×10^{-4}
gradient clipping	1
L2 regularization	1×10^{-4}
discount factor, γ	0.99
averaging rate, ν	0.02
minimum value, ε^*	0.05
decreasing factor, λ	0.99

Within the simulation process, there is a two-way communication between the simulator and the controller: on the one side, the simulator sends messages containing the scenario specifications, including the six input signals expected by the SSNN (Section III-C); on the other hand, the controller sends maneuver messages for vehicle control, including the steering wheel angle and the longitudinal acceleration.

The DQL core is implemented with Wolfram Mathematica, and the communication between the ADS and the DQL middleware is realized through the Wolfram Symbolic Transfer Protocol[3] (WSTP), which is a native protocol for transferring Wolfram Language symbolic expressions with external programs. Also, when the training is completed, the system is reconfigured by converting the deep Q-network module into C++ code and embedding it into the ADS, as in Fig. 7 (b). The code for the network is available at https://github.com/tonegas/safe-speed-neural-network with MIT license, and at https://zenodo.org/communities/dreams4cars/.

A. The Training Procedure

The training procedure is carried out with two asynchronous parallel processes.

The neural process trains the SSNN as follows (the pseudo-code of the algorithm is presented in the supplementary materials Alg. 1):

1) when the RB common and RB rare files are ready to be read, i.e. the DQL middleware is not writing on the files, the DQL core loads the buffers and creates a batch of training data by randomly sampling the two buffers, taking only 5% of the data from the RB rare;

[3]www.wolfram.com/wstp

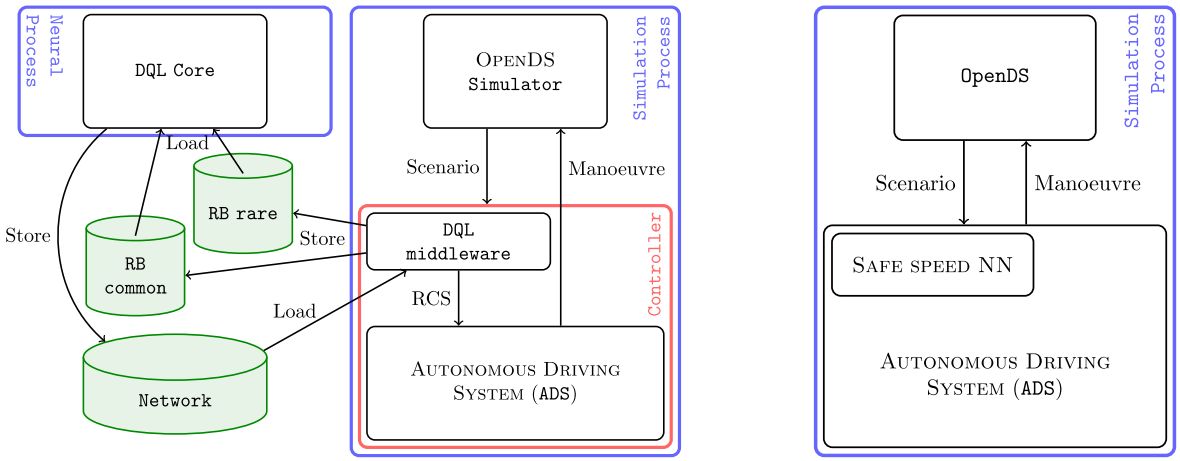


Fig. 7. Framework of integration between the autonomous driving system and the safe speed neural network, for the training phase (a) and for the inference phase (b), i.e. the use of the trained network.

2) the DQL core updates the weights of Q and $\hat{Q}$ in equation (4) using, respectively, the ADAM optimization algorithm and the Polyak averaging (5), and it stores them in the network file;
3) the DQL core updates the value of ε using the rule in equation (3), and it stores it in the network file;
4) the training stops if it reaches the maximum number of epochs, otherwise it restarts from step 1).

The simulation process produces the datasets (the pseudo-code of the algorithm is presented in the supplementary materials Alg. 2):

1) when the network file is ready to be read, i.e. the DQL core is not writing on the file, the DQL middleware loads the weights of the network and the value of ε from the network file, and the simulator starts a new episode;
2) the simulator sends a message to the controller containing the current scenario specifications;
3) the DQL middleware processes the message to identify the current state s_t and collects the reward r_{t-1} that refers to the state change from s_{t-1} to s_t caused by the past action a_{t-1}; then, it composes a new tuple $\langle s_{t-1}, a_{t-1}, r_{t-1}, s_t \rangle$;
4) if the episode fails, the DQL middleware saves the tuple in RB rare, the simulation ends and the process restarts from step 1); otherwise, it saves the tuple in RB common;
5) the DQL middleware calls the network with the current state s_t to estimate the rewards of all possible actions and chooses the next a_t using the ε-*greedy* policy;
6) the DQL middleware computes the new value of RCS, $\widetilde{v}$ using equation (8) and sends it to the ADS.
7) the process restarts from step 1) until the training is completed.

Table I summarizes the hyper-parameters we adopted for the training of the final framework. These parameters were determined with a heuristic approach based on trial and errors, guided by two criteria: a) smoothness of the Q function assessed by inspection of plots such as those shown in Fig. 8 and b) improvement of the ADS performance assessed with the statistical evaluation (Section V-C) and with the comparison of maneuvers (Section VIII-A). This heuristic approach worked for us: one should note that the layered approach (Fig. 2) may be suboptimal but cannot harm ([2], Section II.E.1). This paper is not about algorithmic optimizations, but, in case one wishes to push optimizations at the extremes, one might follow methods such as the Taguchi Method (e.g., a good example may be [39]). Further elements may be found in the supplement materials.

V. Results

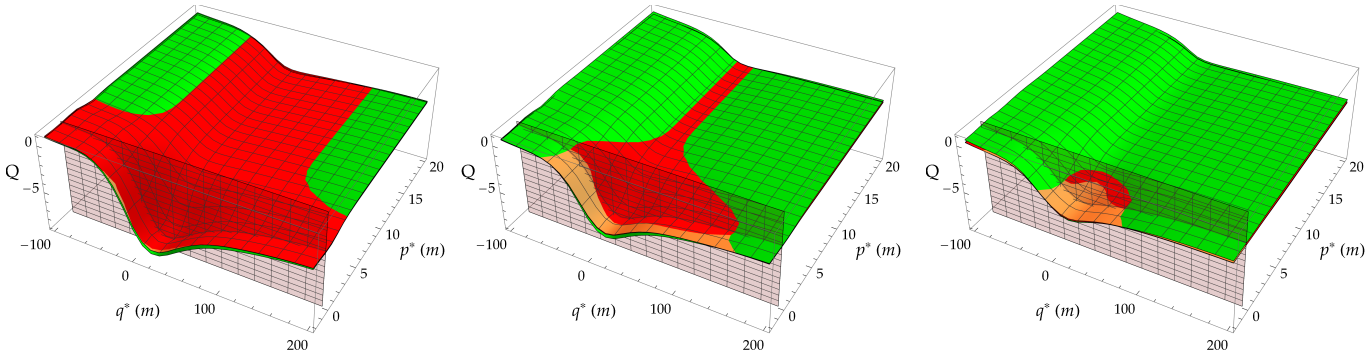
This section presents the results of applying the SSNN to a static context, where we study the output of the SSNN for given inputs, and to a dynamic context, where we test the integration of the SSNN with the ADS in simulation. Also, we include a statistical evaluation of how the SSNN improves the performance of the ADS.

A. Results in Static Context

This section illustrates the 3 outputs $Q(s, a), a \in \{-1, 0, 1\}$ produced by feeding a given state s to the SSNN.

Since the state is a six-dimensional vector (Section III-C), we plot a two-dimensional slice of the functions $Q(s)$, where 4 states are kept constant: the orientation is set at $\alpha = 0$, the pedestrian's speed is set at $w = 2.78$ m/s (9 km/h) and the speed of the vehicle is equal to the RCS. Fig. 8 shows three different plots for three different values of $v = \widetilde{v} \in \{5, 7, 9\}$ m/s. The three charts plot $Q(s)$ as a function of the longitudinal q^* and lateral p^* distances of the pedestrian: q^* varies in the range $[-100, 200]$ m; p^* in the range $[-2, 20]$ m (Fig. 8).

The vertical axis shows the value function Q for the three actions in $\mathcal{A}$: green indicates the action of increasing the RCS, red the action of decreasing the RCS, and orange the action of



(a) $v = 9$ m/s (b) $v = 7$ m/s (c) $v = 5$ m/s

Fig. 8. Visualization of the estimated future rewards Q computed by the network for a given set of states, varying the location of the pedestrian q^*, p^* and the speed of the vehicle v. The color of the surfaces represents the corresponding chosen action: increase the RCS in green, decrease the RCS in red, and keep the RCS constant in orange.

keeping the RCS unchanged. Note that each plot shows all the three surfaces of Q at once, but only the one corresponding to recommended action is visible on top of the other two.

First, let us analyze the similarities among the three plots of Fig. 8. One can immediately notice how the minimum Q always occurs near $(q^*, p^*) = (0, 0)$ where collision is most likely. Moreover, the faster the vehicle is, the lower the Q value is at that minimum point. Conversely, the maximum value of Q occurs at the end of the road, for where $q^* = -100$ (successful conclusion of the episode). This is consistent with what we would expect given the reward function definition (9).

In the charts, the boundary between regions of different colors marks the position where the network policy switches action selection. For example, the boundary from green (increase RCS) to red (decrease RCS) in Fig. 8 (a) and (b) shows the point where the approaching vehicle should start braking. Note that this transition line holds for the assumptions made for drawing the two-dimensional Q slices ($\alpha = 0$, $w = 2.78$ m/s, $v = \widetilde{v} \in \{5, 7, 9\}$ m/s) and would be different in other conditions.

One first observation is that the brake point generally occurs earlier for $v = 9$ m/s (a) than for $v = 7$ m/s (b), and it does not even happen for $v = 5$ m/s if the pedestrian is sufficiently far from the lane ($p^* > 3$ m) as shown in (c).

A second comment is that the longitudinal position q^* of the brake point decreases with the pedestrian's lateral distance p^*; i.e., if the pedestrian is farther from the lane, the car brakes later. Notably, q^* tends to a constant value when $p^* > 10$ m. For example, in (a), if $p^* > 10$ m, brake happens always at $q^* = 140$ m, regardless of p^*. This asymptotic behavior is an intentional consequence of the network design. As said in Section III-C the input signals are channel coded: the scalar value p^* is encoded by an array of ten neurons. Each neuron is activated (with logistic sigmoid) when p^* trespasses a corresponding threshold. The receptive field for p^* (the location of the thresholds) was uniformly spread in the interval $0-10$ m to obtain a good resolution in this important range of distances. For $p^* > 12$ m, all the neurons saturate. The saturation of all channels encodes the notion of "far pedestrian" (farther than $10-12$ m). In this way, we built a network that discriminates the pedestrian's lateral positions when the pedestrian is close to the lane and is less sensitive to the exact distance when the pedestrian is far. The rails of Fig. 6 span the receptive field, except the last three falling completely into saturation region. So, the safe speed network sets cautious speeds without overtrusting the pedestrians' distance beyond $p^* = 10$ m. The motivation for this feature is to demonstrate the inclusion of design characteristics in the network. For example, this feature could be useful when the pedestrian's position cannot be detected precisely (e.g., approaching locations partially occluded). Of course, one could shape the receptive fields to suit different needs.

A third point concerns the network output after the vehicle has passed the pedestrian, that is $q^* \leq -\frac{W+R}{2} - \delta$. As shown in Fig. 8 the return to the recommendations for free lanes (increase RCS) takes place smoothly. This is partly because the longitudinal distance q^* is channel coded too: there are ten channels 30 m apart that regularize the output in that direction. Also, after passing the pedestrian, less than 100 m remain until the road's end: the driver agent accelerates smoothly by itself. There is not much way in which the SSNN can gain further rewards beyond the agent's acceleration habits. At inference time, Fig. 7 (b), the SSNN is disabled when $q^* < -10$.

Finally, Fig. 8 (b) and (c) shows another interesting behavior in the area where $p^* < 0$ (pedestrian in the collision strip). Here, the selected action is the null action (orange, i.e., to keep the RCS constant). The reason derives directly from the interaction between the SSNN and the ADS. When the pedestrian is on the road, the ADS decelerates in autonomy, no matter what the network advises.

B. Results in Dynamic Context

This section presents the integrated SSNN-ADS, Fig. 7 (b), on two sets of simulations.

The first comprises three scenarios where the pedestrian walks alongside the road, not following the behavior of Section III-D, but just walking in the direction of the traffic with $\alpha = 0$ and at the constant speed $w = 1.39$ m/s (5 km/h). The three simulations differ for the lateral position of the pedestrian, i.e., $p^* \in \{1, 4, 9\}$ m—values not appearing in the training phase as they do not correspond to any

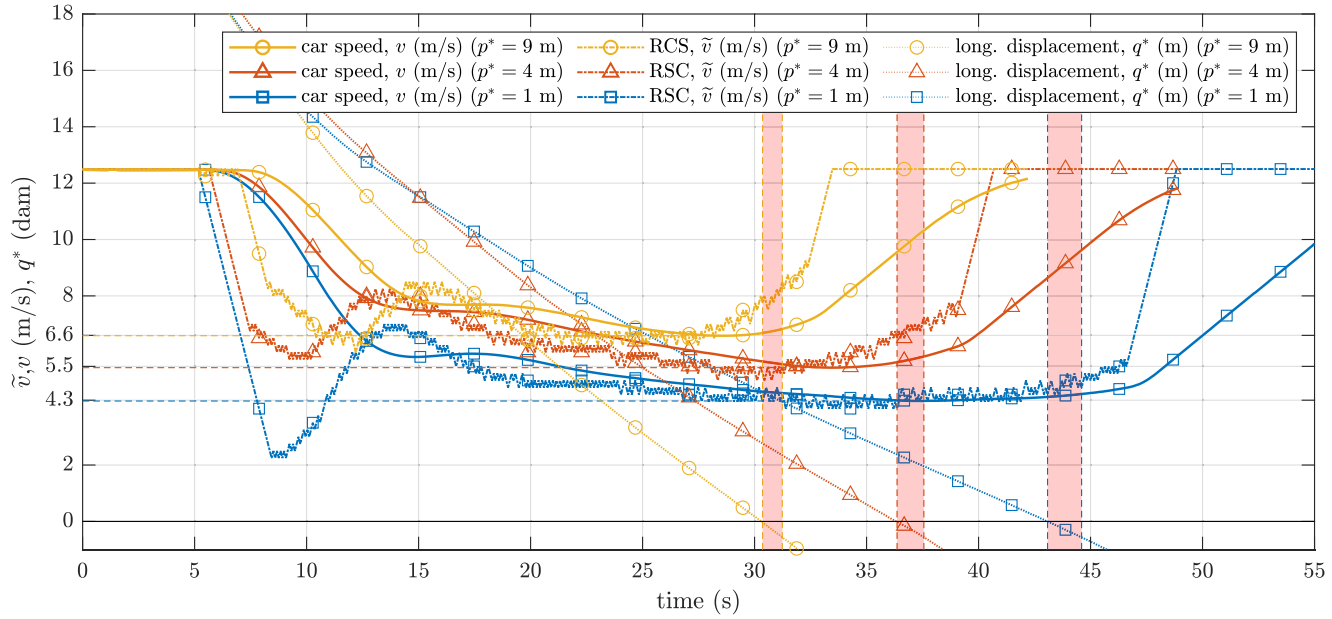


Fig. 9. Results of the integrated system on three simulations with different lateral displacement of the pedestrian. Each simulation is marked with the same color and markers. Different line styles indicate different variables. The vertical axis shows: the longitudinal displacement, q^* (dam), the RCS, $\tilde{v}$ (m/s), and the speed of the vehicle, v (m/s). Note the minimum speed undershoots. The horizontal axis shows the simulation time (s). The red vertical bands represent the areas of possible collision $(-2 * (\frac{L+R}{2} + \delta) \leq q^* \leq 0)$.

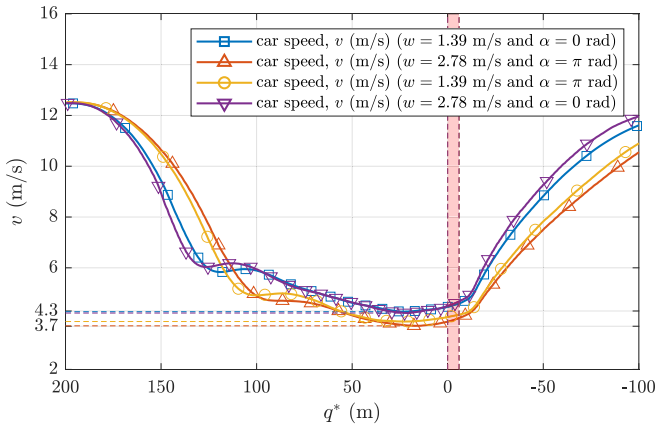
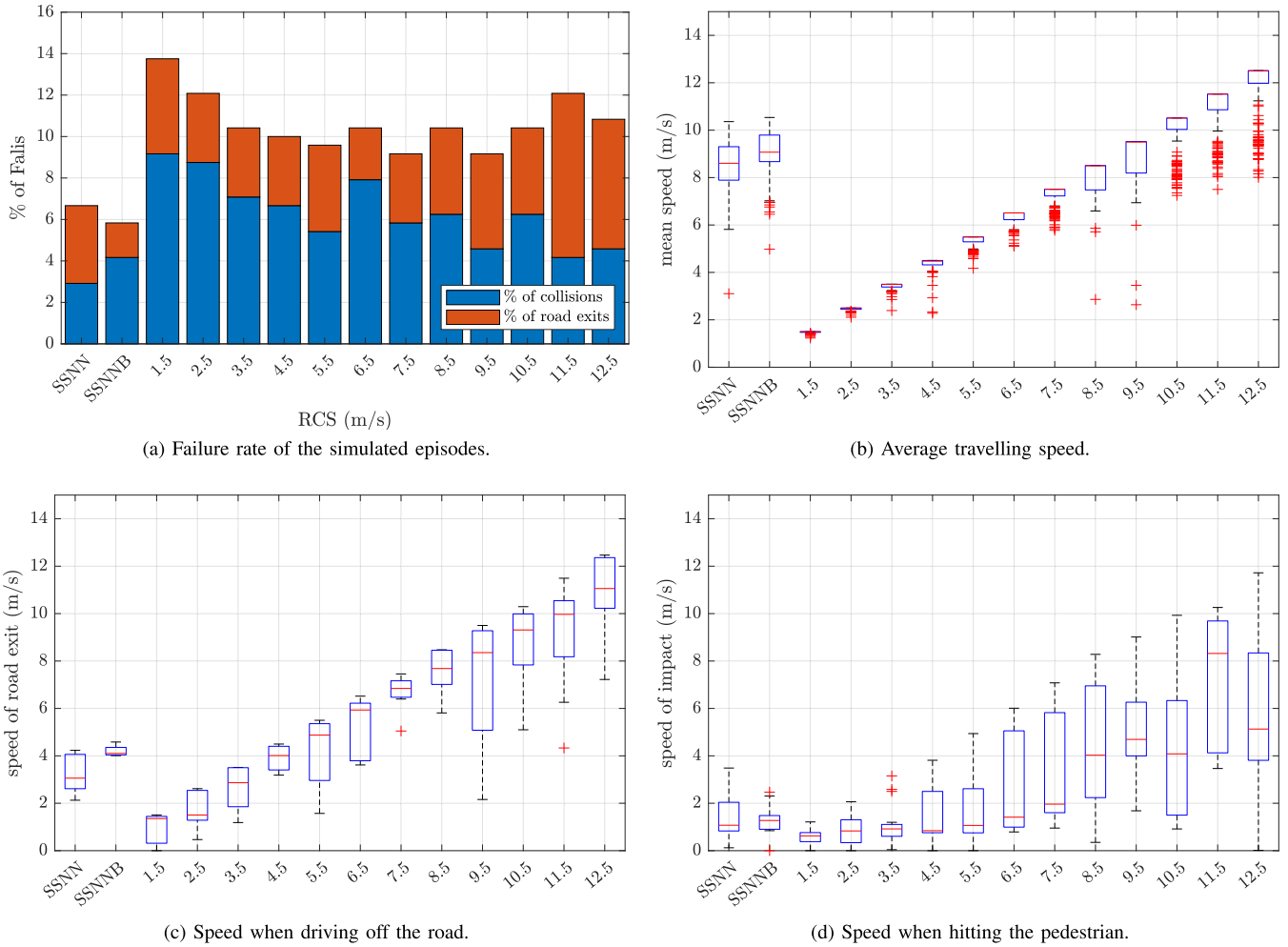


Fig. 10. Results of the integrated framework on four simulations using different constant speed and orientation of the pedestrian. The vertical axis shows the speed of the vehicle (m/s), highlighting its minimum values in the four simulations. The horizontal axis shows the longitudinal displacement of the pedestrian (m), highlighting in red the area of possible collision $(-2 * (\frac{L+R}{2} + \delta) \leq q^* \leq 0)$.

of the pedestrian "rails" defined in Section III-D.2. Note that since the pedestrian is not inside the lane, nor even walking towards it, no conflicting path may be predicted. The driver agent without SSNN would travel at the posted speed limit. However, with SSNN, the agent takes on a cautious approach: Fig. 9 plots the speed of the vehicle, the RCS, and the longitudinal distance q^* of the pedestrian. The SSNN is turned on at 5 seconds ($q^* = 200$ m). The network quickly decreases the RCS, which the ADS slowly follows (with its dynamics). Undershoots for the RCS happen because the SSNN seeks to accelerate the speed reduction (see Section VII-A). Notably, the network correctly suggests a more moderate RCS when the pedestrian is closer to the lane. In all three simulations, the vehicle reaches the minimum speed (marked with dashed horizontal gridlines) several seconds before getting to the pedestrian's side (the vertical red bands). Notably, the network increases the RCS slightly before passing the pedestrian. This behavior is, in fact, similar to what human drivers do: minimize the time to cross a potentially dangerous zone. The reason for this apparently odd behavior is to minimize the chances for the pedestrian to enter the lane when the time for reaction is the shortest. When the vehicle is about 10 m past the pedestrian, the network is deactivated, as explained in Section V-A, and the RCS rapidly returns to its initial value. One last observation concerns the oscillations of the RCS during the time window between 5 and 15 seconds. In a real vehicle, this can cause undesirable oscillation in the vehicle's speed if the low-level control is too slow to actuate the changes. A first solution to this issue is to use a smaller $\Delta_{\hat{v}}$ in equation (8) when using the framework on the real vehicle, as the next Section VI explains.

The second group of simulations concerns four different scenarios with constant lateral displacement of $p^* = 1$ m, but different combinations of speed $w \in \{1.39, 2.78\}$ m/s (5 and 10 km/h) and orientation $\alpha \in \{0, \pi\}$ rad. Fig. 10 shows the speed of the approaching vehicle in the four cases. The network suggests a slower RCS when the pedestrian walks towards the car, perpendicular to the street: the minimum car speed is $v = 3.7$ m/s when $\alpha = \pi$ rad, versus $v = 4.3$ m/s when $\alpha = 0$ rad. On the other hand, the pedestrian's speed does not affect the network's decisions as much. This might



(a) Failure rate of the simulated episodes.

(b) Average travelling speed.

(c) Speed when driving off the road.

(d) Speed when hitting the pedestrian.

Fig. 11. Statistical evaluation of the performance of the driver agent with or without safe speed network. The leftmost columns of each plot show the ADS performance with the SSNN (the SSNNB refers to a slightly different version of the SSNN described in Section VII-A). The remaining columns show the performance of the ADS without the safe speed network with different posted speed limits.

be explained by the fact that the pedestrian can change speed while crossing the road in the training scenarios, as seen in Section III-D.2. The current walking speed gives no hint about what speed the pedestrian might choose in a possible road crossing.

C. Statistical Evaluation

This section gives a statistical evaluation of some performance metrics for the ADS with and without the SSNN; the SSNNB refers to a slightly different version of the SSNN described in Section VII-A. Fig. 11 shows four different performance metrics. The histogram (a) chart reports the failure rate for the safe speed networks compared to the agent's failure rates without the safe speed network at different posted speed limits in the range $[1.5, 12.5]$ m/s. The box whisker charts make a similar comparison, respectively, for the mean traveling speed (b), the speed when leaving the road (c), and the speed of collision (d). Each case (SSNN, SSNNB, and the various posted speed limits for the driver agent without a safe speed network) is evaluated for 240 random simulations. In the simulations with the SSNN and SSNNB, the vehicle posted speed limit and initial speed is 12.5 m/s.

Fig. 11 (a) indicates that the inclusion of the SSNN/SSNNB (two leftmost bars of the histogram) almost halves the failure rate. The ADS without the safe speed network presents many failures even for very low posted speed limits. This might be explained by the fact that in these scenarios, the pedestrian's speed does not differ significantly from the speed of the vehicle, and, as a consequence, the time window when the collision might happen is more prolonged (as noticed, it is better to pass the pedestrian quickly). The proportion between out-of-road and collision events varies with the speed limit (with more out-of-road events for faster speeds). This behavior derives from the driver agent's sensorimotor system seeking collision avoidance when braking is hard.

The box whisker charts reveal that the safe speed networks permit almost the same mean speed of the case with $8.5-9.5$ m/s as the posted speed limit (b). However, the collision speed (d) is as low as the cases with $3.5-4.5$ m/s. When going off-road (c), the speed is similarly as slow as the cases with $3.5-4.5$ m/s. So, the networks do not achieve zero failure rate: the given reward function (9) tolerates a little rate of low-speed failures. However, besides reducing the failures, their severity is also reduced (of course, a designer could seek a different

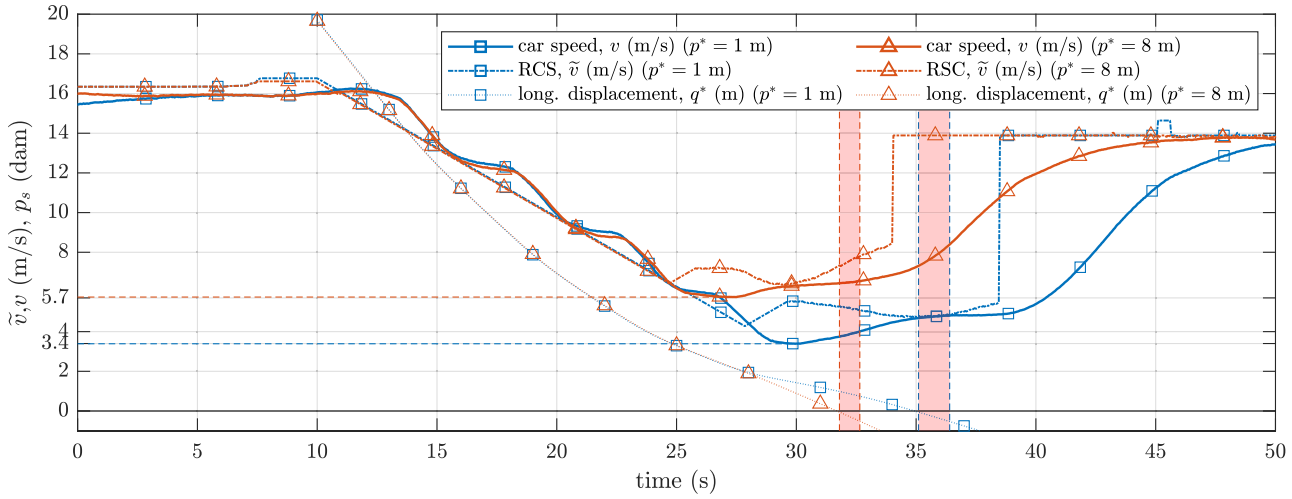


Fig. 12. Results of the integrated framework on two experiment on a real vehicle using different constant lateral displacement of the pedestrian. Each simulation is indicated using the same color and line marker. While the different line style highlights the different variables. The vertical axis shows: the longitudinal displacement (dam), the RCS, $\widetilde{v}$ (m/s), and the speed of the vehicle, v (m/s) highlighting its minimum values in the three simulations. The horizontal axis shows the time of simulation (s). The red bands represent the area of possible collision ($-2(\frac{L+R}{2} + \delta) \leq q^* \leq 0$).

trade-off). In Section VIII-A we will show that the situation is even better because the events counted as collision here, in reality, are mere violations of the safety clearance δ.

VI. Test on the Real Vehicle

This Section demonstrates the transfer of the learned policy on a real vehicle: a Jeep Renegade prototype used in the Dreams4Cars project. We performed two tests on a real track in mixed reality. The vehicle was real, but the pedestrian was not for safety reasons. The pedestrian was emulated by hallucinating the perception system. It was walking at 1.1 m/s alongside the road with $\alpha = 0$ and lateral displacement of 1 m and 8 m, respectively, in the two experiments. The initial speed of the vehicle is 16 m/s in both cases.

Fig. 12 displays the experimental results. As in Fig. 9, it plots the speed of the vehicle, the RCS, and the longitudinal position of the pedestrian. The horizontal axis is the time, which starts when the network is activated at $q^* = 200$ m. The network is disabled when $q^* = -10$ m. The vertical red bands indicate when the pedestrian is on the vehicle's side in the two cases. The results are overall consistent with the simulations presented in Section V-B. However, there are oscillations in the deceleration phase due to the gear shifts of the car's automatic transmission. The minimum velocity is also below the minimum RCS (undershoot). Apart from that, the network correctly imposes a slower RCS when the pedestrian is closer to the road. In both experiments, the vehicle reaches the minimum speed within a safe temporal margin from the potential conflict area. Shortly after passing the danger zone, the network is disabled, and the vehicle starts increasing the speed, although it does not fully resume the initial speed of 16 m/s because of the curved geometry of the track.

These tests demonstrate that the SSNN trained in simulation can be successfully applied in a real system. The reason why the transfer is successful lies in the layered approach (Fig. 2). The safe speed network steers an autonomous driving system where the driver agent is already capable of controlling a real vehicle by itself. In the approaches where the network learns control from scratch, the fidelity of the simulation environment is more crucial.

The only adjustment we made to use the network on the real vehicle (driven by the same agent) was the value of $\Delta_{\hat{v}}$ in equation (8), which was decreased at 0.035 m/s to prevent excessive oscillations in the system in the gear shifts. The problem is caused by the vehicle's low-level driveline control, which regulates the automatic transmission: the control is too slow to actuate the updated value of the RCS so that this delay excites the system and generates oscillations. By decreasing the updating step of the RCS, we managed to limit the oscillations to a tolerable amount. The next section defines an alternative way of using the SSNN, without incremental adjustments, that is immune to this drawback.

VII. Extensions and Scalability

A. RCS Without Incremental Adjustments via Bisection Algorithm

This section describes a speed specification system that does not suffer from oscillation problems, even in the presence of actuation delays. It also solves the problem of producing the final safe speed immediately, without the incremental adjustments (hence going beyond the three actions of the SSNN). For this, we integrated the SSNN within a bisection root-finding loop to determine the final equilibrium safe speed. This new function is called SSNNB.

The bisection procedure is shown in the Algorithm 3 presented in the supplementary materials. It allows the safe speed value to be obtained without having to wait for the transient in which the autonomous driving system adapts the vehicle speed to reach the recommended RCS. The idea is repeatedly calling the SSNN with hypothetical and equal vehicle speed and RCS

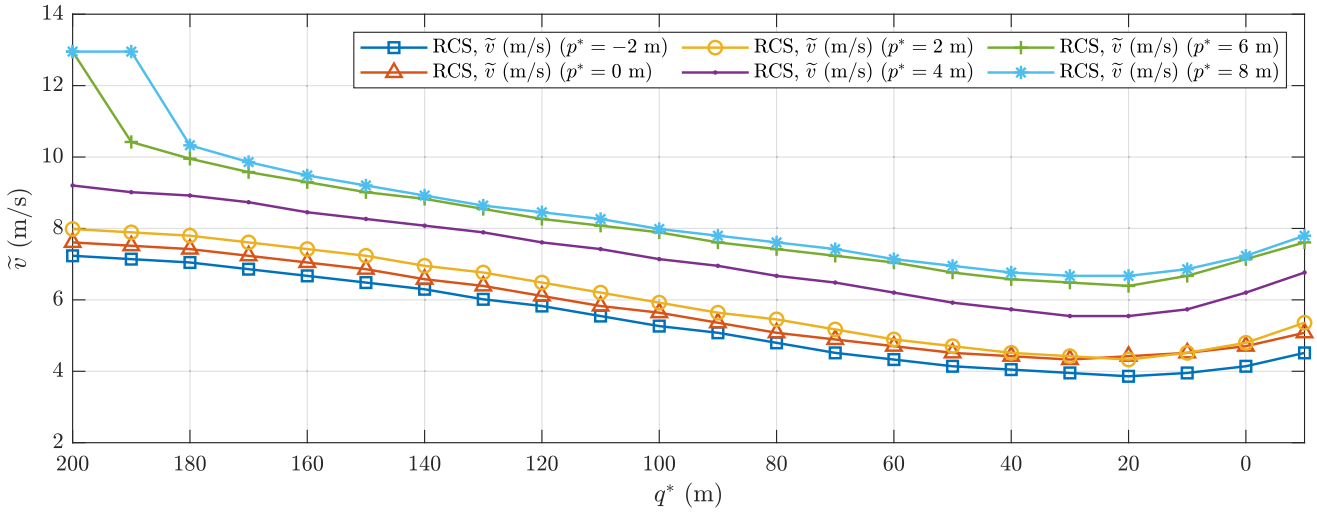


Fig. 13. The RCS suggested by the SSNNB algorithm for different lateral displacement, p^* when the pedestrian walks at 1.39 m/s (5 km/h) parallel on the road in the same direction of the ego vehicle.

($v = \hat{v}$), seeking with bisection algorithm the speed value that produces the action $a = 0$, i.e., when the hypothetical RCS does not need adjustments. By doing so, it is possible to determine the final RCS at which the network intends to stabilize.

Notably, the SSNNB algorithm completely decouples the safe speed network from the vehicle dynamics (the final safe speed is immediately computed and no longer depends on how the vehicle reacts to the requested adjustments). Figure 13 shows the output of the SSNNB for varying pedestrian positions in the case of a pedestrian walking at 1.39 m/s (5 km/h) parallel on the road in the same direction of the ego vehicle. With decoupling, the transient undershoots previously noted in Fig. 9 are eliminated (Fig. 14).

In evaluating the safe speed presented in Fig. 13, one should consider that the ADS does not have an emergency brake function: reaching the maximum deceleration is not instantaneous (the maximum jerk of vehicle controlled by the agent is about -10 m/s^2). The networks specify speed limits that account for that.

B. Handling Complex Scenarios

This section demonstrates that the action-inhibition approach implicit in layered control (Fig. 2) is modular. In layered control, the inhibitions of different origins can be superimposed [2, Section II-C.1]. This means that we can consider the safe speed for each pedestrian in isolation and take the minimum safe speed to inhibit the driver agent's choices—the agent will drive according to this new limit.

To prove the effectiveness of the method, a scenario with three pedestrians, each with a different position, orientation, and speed, is studied in this section. Pedestrian 1 walks aligned with the road at speed 2.8 m/s (10 km/h) and lateral displacement, $p^* = 2$ with direction opposite to that of the ego vehicle. Pedestrian 2 advances aligned to the road with a speed of 1.39 m/s (5 km/h) and lateral displacement, $p^* = 2$ (opposite side) with the same direction as the ego vehicle. Pedestrian 3 walks perpendicularly to the road with a speed of 0.14 m/s (0.5 km/h). Fig 14 shows the safe speed of each pedestrian in isolation. The requested safe speed is the minimum of the three. Every time the car passes one pedestrian, the safe speed is reset to the minimum of the remaining pedestrians ahead.

VIII. Discussion

As mentioned in Section II, many literature studies deal with learning to drive [6], [11], [12]. A common trait of them is the use of reinforcement learning to produce *explicit control*.

Conversely, this paper aims at *extending the cognition abilities* of a given driver agent. We use a layered control architecture where the topmost layer is the safe speed function, and the remaining layers underneath are the driver agent. The safe speed layer acts by providing a general directive (speed limit) to the agent (Fig. 2).

Thus, the main differences between the literature and this paper are: 1) the goal, which is learning vehicle control versus learning a high-level behavioral directive for a driver agent; 2) the agent architecture (layered control versus one single function). The many implications of these differences are discussed in the following. For comparison, we consider two papers tackling the problem of autonomous braking in pedestrian intersections: [20] and [21].

A. Revisiting the Network's Function

As shown in Fig. 11, the driver agents endowed with safe speed networks obtain mean speed travel similar to the baseline agent at the posted speed limit of about 8.5-9.5 m/s, but with the failure rates of the baseline agent at about 3.5-4.5 m/s.

Fig. 15 shows the set of trajectories for the baseline agent (i.e., the agent without the safe speed network) at the posted speed limit of 8.5 m/s (left), for the agent with the SSNN

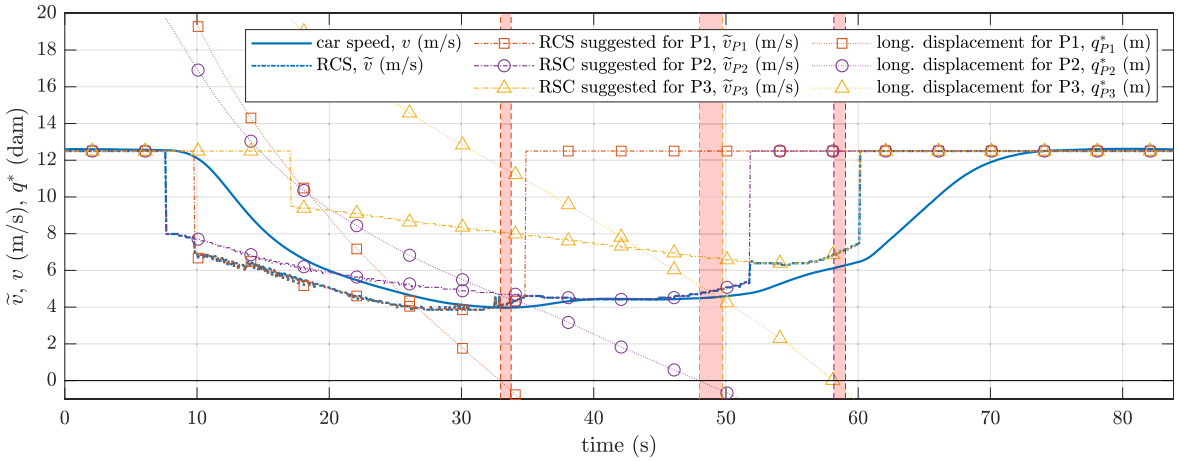


Fig. 14. Results of the integrated system using the SSNNB on a simulation with three pedestrians. P1 (pedestrian 1) advances aligned with the road with speed equal to 2.8 m/s (10 km/h) and lateral displacement, $p^* = 2$ with direction opposite to that of the ego vehicle. P2 advances aligned to the road with a speed of 1.39 m/s (5 km/h) and lateral displacement, $p^* = 2$ (opposite side) with the same direction as the ego vehicle. P3 walks perpendicularly away from the road with a speed of 0.14 m/s (0.5 km/h). For each pedestrian, the suggested RCS (dash-dotted line) and the longitudinal displacement (dotted line) are shown with the same color and line marker; for P1 using red squares; for P2 using purple circles; for P3 using yellow triangles. The proposed final RCS, $\widetilde{v}$ (m/s) and the speed of the vehicle, v (m/s) are represented in dashed blue and solid blue lines, respectively. The horizontal axis shows the time of simulation (s). The red bands represent the area of possible collision $(-2(\frac{L+R}{2} + \delta) \leq q^* \leq 0)$.

network (center), and the agent with the SSNNB algorithm (right). For each case, 240 simulations are shown.

The first row concerns the successful events. As shown in Fig. 2 the baseline agent predicts the other road users' paths before planning and control. The agent cannot predict when the pedestrians will turn towards the lane. However, shortly after the switch of intentions is apparent, the baseline agent predicts their path and plans accordingly. This happens regardless of the lateral distance p^*. So the agent has more time for a reaction if the pedestrian is farther from the lane. Thanks to path prediction, the agent can avoid most of the collisions. It fails only when the pedestrian changes intentions close to the lane and the car. Fig. 15 (top, left) shows the successful events, which are the vast majority. We can see clusters: one in which the agent successfully stops at about $q^* = 5$ m. In most of them (when the intention switch happens with enough anticipation), the agent always stops in the same way, beginning the stop maneuver at about $q^* = 40$ m. This stopping behavior follows a study on human stopping maneuvers [40]. In a few cases, the pedestrian switches intentions late, inducing sharper stops such as, e.g., those beginning at about $q^* = 15$ m (stopping in 10 m). In other cases, the agent resolves the conflict by modulating the speed without even stopping. No conflict happens in a final group of events, and the agent follows the posted speed limit.

At the center and right of Fig. 15, top row, the effect of the safe speed networks is evident. The networks modulate the speed limit in advance (SSNNB shows no oscillation mentioned above). Overall the behaviors are of the same types (stop, late stop, modulation of velocity, and no conflict), but this time two clusters are evident: the lower speed one corresponds to closer rails XL, XR1 of Fig. 6. The faster cluster to farther rails XR6-XR7.

The middle row shows the out-of-road events (we count out-of-road when the car touches the lane marking). They happen at reduced speed for the safe speed network (and less frequently for the smoother SSNNB).

Perhaps the most interesting chart is the bottom row, which presents the collisions. For the baseline agent, collisions happen for very late intention switches and occur during the stop maneuver. The inset gives a magnified view of the speeds at impact. A few collisions are counted on the car's rear when the pedestrian crosses the lane in the clearance zone δ immediately behind the car. The chart reports the fraction of total and frontal collisions (not counting the rear collisions caused by the pedestrian). One reason why the agent learns to pass the pedestrian quickly is precisely to avoid these side collisions.

One can notice that, for the safe speed networks, the collisions on the front are counted when the distance is less than the safety clearance δ. However, the speed is so slow, and the deceleration is enough that, by extrapolation, the car will barely touch (if not at all) the pedestrian. This justifies the choice of tolerating a little rate of low-speed collisions in (9). We should better count no frontal collisions for SSNNB and only 1 for the SSNN.

B. Function to be Learned

In layered control, the driver task is split among layers of competence. In principle, leaning the safe speed directive needs only a binary output: whether the current speed limit is too high or not. Subject to the speed limit, the driver agent takes care of path planning and control.

In layered control, an important point is where the prediction of pedestrians' intentions is carried out. A convenient choice is splitting this function between the agent and the safe speed layer as follows. The agent is endowed with

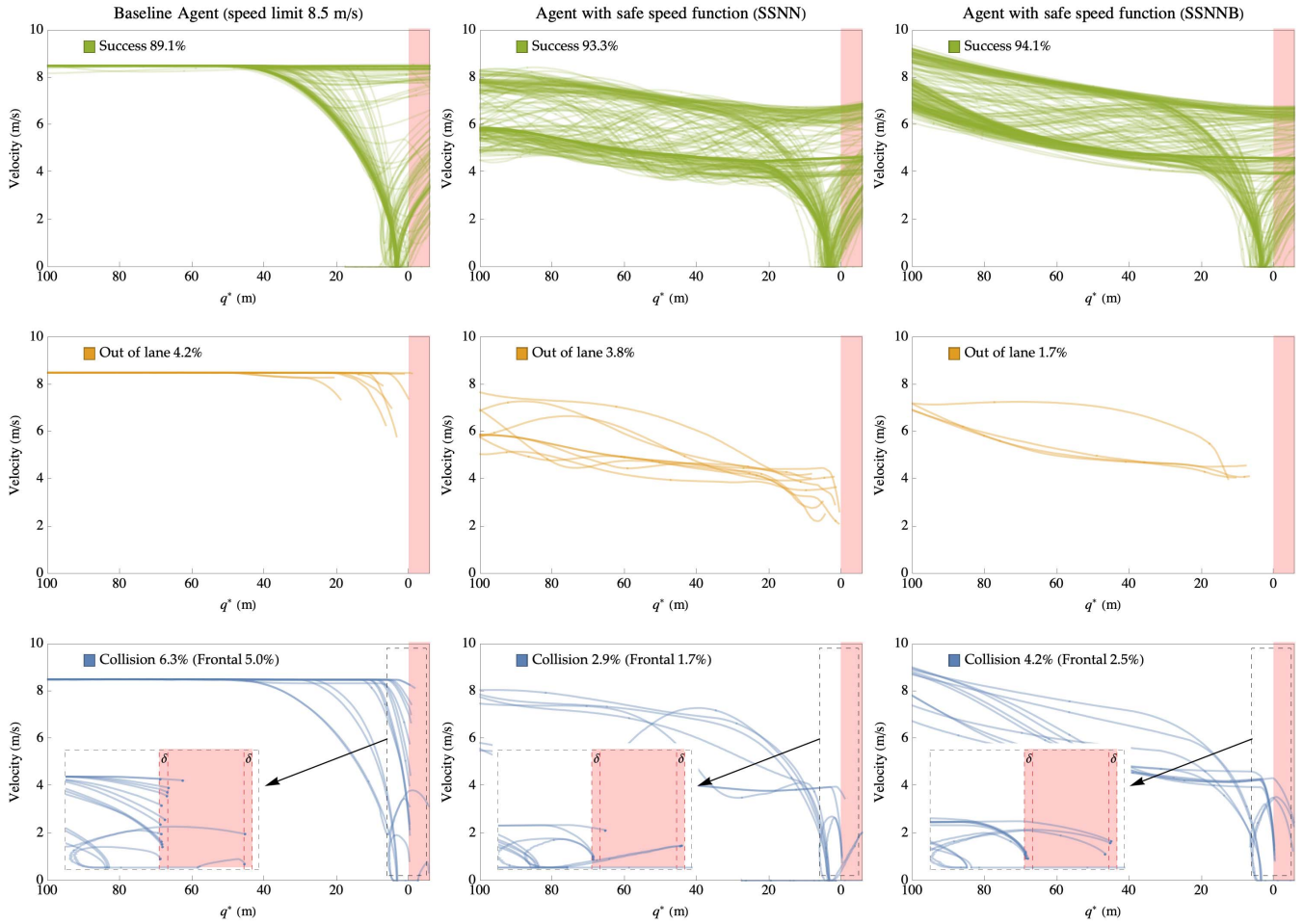


Fig. 15. Comparison between the baseline agent (the driver agent without safe speed function) at 8.5 m/s, agent with the SSNN network, and the agent with the SSNNB algorithm. The three have almost the same mean speed, but the agents with a safe speed network modulate the speed limit getting fewer and less severe failures. Detailed comments are given in the text.

predictive models of the other road users' trajectories and uses this ability for path planning to produce maneuvers free of conflicts. Whenever the pedestrian's intentions are correctly predicted, the agent drives safely. However, on occasions, the pedestrians' intentions are not predictable, and here the safe speed layer intervenes. The safe speed layer needs only to learn whether, in the given context, the ADS maneuverability is not enough to compensate for unexpected mobility of the pedestrian.

The literature examples do not use layered control. They learn the whole sensorimotor stack in one single black box. This means that, implicitly, the learning problem includes learning prediction, planning, and control aspects that in the layered approach are left to the driver agent. Where we extend the abilities of an agent developed and tested elsewhere, the literature learns everything in a single step. This means that the learned policy is more complex, with larger dimensions and number of hidden layers. For example, in [20] the number of neurons for each layer are 15 (input layer), 100, 70, 50, 70, 100, and 4 (output layer) for a total number of approximately 23 thousand trainable parameters. Paper [21] uses two methods: one with a network with three hidden layers with 256 neurons each (approximately 140 thousand neurons) and a second method (actor-critic) with two networks counting approximately 40 thousand trainable parameters each (total 80 thousand). The network used in this paper (Section III-C) uses approximately 13 thousand trainable parameters, which is between half and ten times fewer parameters. However, these figures are indicative because it seems that no particular optimizations were used (both for the literature and for our case).

The output dimension is also affected: for example, [20] uses four discrete longitudinal acceleration levels, which are still a coarse-grained control yielding piece-wise linear velocity (with jerk pulses at switch times) [20, Fig.6]. As for predicting pedestrian's intentions in the black-box approach, there is no separation between predictable and unpredictable trajectories.

C. Reward Function

The structure of the reward function follows the purpose of the learning problem. For the layered control, (9) responds to the need of learning the highest speed that is safe. Episodes end either successfully or with a collision (the car travels at a variable speed but always in the same direction). Let n be the duration of one episode. In the successful case,

the accumulated reward is $\sum_{k=0}^{n-1} -0.01\gamma^k + 0.5\gamma^n$. This value is monotonically decreasing with n and corresponds to a minimum travel time criterion. In the case of collisions, the term $0.5\gamma^n$ is replaced by $-v_n\gamma^n$ and the accumulated reward decreases by $-0.5\gamma^n - v_n\gamma^n$ for every collision. The reward function finds a trade-off between fast travel (Fig.11, b) and rare occurrence of low-speed collisions (Fig. 11, d). The trade-off tolerates a little rate of low-speed collisions because the criterion (highest speed limit subject to no collision) is implemented with a penalty formulation (9).

Conversely, for learning vehicle control, the reward function must include additional terms directing control quality figures such as limiting jerk and accelerations ([21], reward function) or other similar terms instructing how to drive ([20, Fig. 6], equation 8).

D. Training Process

The training of the network was carried out with 30000 epochs; albeit in practice, convergence was obtained after 2000 epochs, which is similar to [20, Fig.5]. Most of the computation time is spent in the simulation phase (Section IV-A). A feature shared with [20] is the use of an auxiliary replay buffer to remember the collision events (called RB rare in this paper and "trauma memory" in [20]).

E. Scalability

In the layered control paradigm, learning the safe speed limit is decoupled from vehicle control, which facilitates both scalability and transfer of the learned policy to real cars. Scalability is implicit in layered control [41]. The safe speed specification acts as a bias for the driver agent's action-selection. The agent chooses among the actions that it can recognize in the environment. If a new action is discovered, this becomes immediately available for competition with the others. For example, if the lane were winding instead of straight, the layered approach would still work: the agent would choose the speed in curves according to [42], [43] but subject to the speed limit provided by the safe speed network. Instead, the literature approaches would need additional terms to modulate the speed in the curves. The biases produced by the safe speed function can coexist with biases produced by other mechanisms, for example, the proactive lane change described in [2, Section II.C].

F. Transfer to Real Cars

In layered control, the safe speed function provides a driving directive to the ADS. To control the vehicle, the agent embeds inverse models of the vehicle dynamics. Using agents that implement the inverse dynamics of different vehicles, the combined agent-vehicle plant may look the same to the safe speed network (Section VI).

IX. Conclusion

This paper presented a novel approach, funded on layered control architectures, for learning safe behaviors in the interaction with pedestrians. We propose a method, using reinforcement learning, to learn associations between the context and the risk of unpredictable events that are countered by suggesting a proper safe speed preventively.

Our approach's key feature is to develop the RL network *in synergy* with an already functioning autonomous driving system and as an extension of it. In this way, the network specializes in learning the safe speed without the burden of learning to drive the vehicle from scratch.

The results demonstrate the advantage of extending the ADS with the neural network, such as modularity, scalability, upgrading existing agents, etc. For example, the physical experiments demonstrate that the integrated system can be transferred to a real vehicle without additional training.

The modular nature of layered control permits extensions to scenarios with multiple pedestrians: the neural module computing the safe speed for one individual pedestrian could be easily reused to compute the lowest speed for a set of visible pedestrians.

One important aspect is that the layered approach has no requirement about the kind of driver agent underneath: it may be an agent based on artificial intelligence or a more traditional one. There is also no requirement about the agent's ability (which need not be perfect): the network extensions learn to specify speed limits adapted to the agent, preventing collisions.

A. Future Work

So far, we have demonstrated a method for biasing an agent behavior towards cautious behavior. The method was demonstrated for pedestrians walking at speed up to 7 km/h who might unexpectedly change their minds. We assume they can instantaneously change traveling direction and speed (up to 7 km/h) without any prior indication. What the cautious behavior learns is to approach these agents with limited speed, permitting successful emergency maneuvers. Above all, the cautious speed depends on the pedestrian's distance from the lane and its mobility. A real pedestrian might have less mobility than the assumptions made here: they need some time to change direction, and they need some time to accelerate to the final speed. So the safe speed networks fund here might be over cautious.

On the other hand, one might wish to develop functions for other kinds of vulnerable road users. For example, children run, bikes are faster but takes time to change direction, and electric scooters are almost as fast (a significant problem today because human drivers are not prepared for the scooter's mobility). One might, for example, develop different safe speed networks to use for different kinds of road users.

A final consideration concerns occlusions. To operate in environments where pedestrians might be occluded (e.g., behind obstacles or corners in intersections), there are essentially two options.

Preliminary, the agent must know which parts of the environment are occluded (for example, this information may be derived from an occupancy grid representation obtained in an almost straightforward manner with range sensors).

Option 1 relies on a process that could be defined as "online mental simulations". The process consists of creating hypothetical pedestrians in the occluded regions, just like a human driver imagining that a pedestrian might hide behind

the corner. This hypothetical pedestrian is then used with the SSNN to check whether its related safe speed should supersede the current one. In the human driver example, this corresponds to the driver adopting the speed corresponding to the imaginary pedestrian state. This use of "online mental simulations" is similar to [24].

Option 2 uses the same approach of imagining pedestrians in occlusions but relies on "offline mental simulations". In this case, the process consists of training a *new* safe speed network that uses an occluded region as input in place of the pedestrian. This new occlusion-safe-speed-network is trained during offline simulations by randomly placing pedestrians in the occlusions and passing the occlusion position instead of the pedestrian. The network will learn a shortcut function mapping the occlusion to a safe speed. During online operation, we will then have two safe speed networks: one is the safe speed network developed in this paper, that will be used for all the observable pedestrians, the second will be the new occlusion-safe-speed-network that will be used for all the occluded positions.

In case of extensions to other scenarios—e.g., intersections with occlusions—the pedestrian movement pattern may be conveniently changed to focus attention on movements that let the pedestrian stay hidden, emerging from the occlusion at unfavorable time and distance. In Option 1, the agent may create imaginary pedestrians in the nearest occluded corners who enter the observable environments with an unfortunate time, speed, and direction. For Option 2, there is more freedom of imagining various movements among those that maintain the pedestrian hidden. The training process will place the least favorable events in the RB rare replay buffer.

REFERENCES

[1] R. Bolado-Gomez and K. Gurney, "A biologically plausible embodied model of action discovery," *Frontiers Neurorobotics*, vol. 7, p. 4, Mar. 2013.

[2] M. D. Lio, R. Dona, G. P. R. Papini, and K. Gurney, "Agent architecture for adaptive behaviors in autonomous driving," *IEEE Access*, vol. 8, pp. 154906–154923, 2020.

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

[4] M. Da Lio *et al.*, "Artificial co-drivers as a universal enabling technology for future intelligent vehicles and transportation systems," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 1, pp. 244–263, Feb. 2015.

[5] A. Rudenko, L. Palmieri, M. Herman, K. M. Kitani, D. M. Gavrila, and K. O. Arras, "Human motion trajectory prediction: A survey," *Int. J. Robot. Res.*, vol. 39, no. 8, pp. 895–935, Jul. 2020.

[6] Y. Ma, Z. Wang, H. Yang, and L. Yang, "Artificial intelligence applications in the development of autonomous vehicles: A survey," *IEEE/CAA J. Automatica Sinica*, vol. 7, no. 2, pp. 315–329, Mar. 2020.

[7] H. L. Oei and P. H. Polak, "Intelligent speed adaptation (ISA) and road safety," *IATSS Res.*, vol. 26, no. 2, pp. 45–51, 2002.

[8] M. Spichkova *et al.*, "Formal models for intelligent speed validation and adaptation," *Procedia Comput. Sci.*, vol. 96, pp. 1609–1618, Jan. 2016.

[9] C. Herranz-Perdiguero and R. J. López-Sastre, "ISA^2: Intelligent speed adaptation from appearance," 2018, *arXiv:1810.05016*. [Online]. Available: https://arxiv.org/abs/1810.05016

[10] B. R. Kiran *et al.*, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Trans. Intell. Transp. Syst.*, pp. 1–18, 2021, doi: 10.1109/TITS.2021.3054625.

[11] S. Kuutti, R. Bowden, Y. Jin, P. Barber, and S. Fallah, "A survey of deep learning applications to autonomous vehicle control," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 2, pp. 712–733, Feb. 2021.

[12] A. Haydari and Y. Yilmaz, "Deep reinforcement learning for intelligent transportation systems: A survey," *IEEE Trans. Intell. Transp. Syst.*, early access, Jul. 22, 2020, doi: 10.1109/TITS.2020.3008612.

[13] L. Jiang, H. Huang, and Z. Ding, "Path planning for intelligent robots based on deep Q-learning with experience replay and heuristic knowledge," *IEEE/CAA J. Automatica Sinica*, vol. 7, no. 4, pp. 1179–1189, Jul. 2020.

[14] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[15] R. A. Rescorla and A. R. Wagner, "A theory of pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement," in *Classical Conditioning II: Current Theory and Research*, A. H. Black and W. F. Prokasy, Eds. New York, NY, USA: Appleton Century Crofts, 1972, pp. 64–99.

[16] W. Schultz, P. Dayan, and P. R. Montague, "A neural substrate of prediction and reward," *Science*, vol. 275, no. 5306, pp. 1593–1599, Mar. 1997.

[17] C. Li and K. Czarnecki, "Urban driving with multi-objective deep reinforcement learning," 2018, *arXiv:1811.08586*. [Online]. Available: http://arxiv.org/abs/1811.08586

[18] J. Chen, B. Yuan, and M. Tomizuka, "Model-free deep reinforcement learning for urban autonomous driving," in *Proc. IEEE Intell. Transp. Syst. Conf. (ITSC)*, Oct. 2019, pp. 2765–2771.

[19] Z. Xu, J. Chen, and M. Tomizuka, "Guided policy search model-based reinforcement learning for urban autonomous driving," 2020, *arXiv:2005.03076*. [Online]. Available: http://arxiv.org/abs/2005.03076

[20] H. Chae, C. M. Kang, B. Kim, J. Kim, C. C. Chung, and J. W. Choi, "Autonomous braking system via deep reinforcement learning," in *Proc. IEEE 20th Int. Conf. Intell. Transp. Syst. (ITSC)*, Oct. 2017, pp. 1–6.

[21] R. Vasquez and B. Farooq, "Multi-objective autonomous braking system using naturalistic dataset," in *Proc. IEEE Intell. Transp. Syst. Conf. (ITSC)*, Oct. 2019, pp. 4348–4353.

[22] M. Bouton, A. Nakhaei, K. Fujimura, and M. J. Kochenderfer, "Safe reinforcement learning with scene decomposition for navigating complex urban environments," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2019, pp. 1469–1476.

[23] N. Deshpande, D. Vaufreydaz, and A. Spalanzani, "Behavioral decision-making for urban autonomous driving in the presence of pedestrians using deep recurrent Q-network," in *Proc. 16th Int. Conf. Control, Autom., Robot. Vis. (ICARCV)*, Dec. 2020, pp. 428–433.

[24] L. Chen, X. Hu, W. Tian, H. Wang, D. Cao, and F.-Y. Wang, "Parallel planning: A new motion planning framework for autonomous driving," *IEEE/CAA J. Automatica Sinica*, vol. 6, no. 1, pp. 236–246, Jan. 2019.

[25] R. Brooks, "A robust layered control system for a mobile robot," *IEEE J. Robot. Autom.*, vol. RA-2, no. 1, pp. 14–23, Mar. 1986.

[26] F. Di Rienzo *et al.*, "Online and offline performance gains following motor imagery practice: A comprehensive review of behavioral and neuroimaging studies," *Frontiers Human Neurosci.*, vol. 10, p. 315, Jun. 2016. [Online]. Available: http://journal.frontiersin.org/Article/10.3389/fnhum.2016.00315/abstrac%t

[27] C. J. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, London, U.K., 1989.

[28] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[29] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[30] H. V. Hasselt, "Double Q-learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2010, pp. 2613–2621.

[31] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[32] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*. [Online]. Available: http://arxiv.org/abs/1509.02971

[33] B. T. Polyak and A. B. Juditsky, "Acceleration of stochastic approximation by averaging," *SIAM J. Control Optim.*, vol. 30, no. 4, pp. 838–855, Jul. 1992.

[34] A. Plebe, M. Da Lio, and D. Bortoluzzi, "On reliable neural network sensorimotor control in autonomous vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 2, pp. 711–722, Feb. 2020.

[35] M. Felsberg, H. Scharr, and P. Forssén, "The B-spline channel representation: Channel algebra and channel based diffusion filtering," Linköping Univ., Linköping, Sweden, Tech. Rep. LiTH-ISY-R-2461, 2002.

[36] E. Jonsson, "Channel-coded feature maps for computer vision and machine learning," Ph.D. dissertation, Dept. Elect. Eng., Linköpings Universitet, Linköping, Sweden, 2008.

[37] P. Forssén, G. Granlund, and J. Wiklund, "Channel representation of colour images," Linköping Univ., Linköping, Sweden, Tech. Rep. LiTH-ISY-R-2418, 2002.
[38] S. J. L. Knegt, M. M. Drugan, and M. A. Wiering, "Opponent modelling in the game of tron using reinforcement learning," in *Proc. 10th Int. Conf. Agents Artif. Intell.*, 2018, pp. 29–40.
[39] S. Gao, M. Zhou, Y. Wang, J. Cheng, H. Yachi, and J. Wang, "Dendritic neuron model with effective learning algorithms for classification, approximation, and prediction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 2, pp. 601–614, Feb. 2019.
[40] M. Da Lio, A. Mazzalai, K. Gurney, and A. Saroldi, "Biologically guided driver modeling: The stop behavior of human car drivers," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 8, pp. 2454–2469, Aug. 2018.
[41] T. J. Prescott, P. Redgrave, and K. Gurney, "Layered control architectures in robots and vertebrates," *Adapt. Behav.*, vol. 7, no. 1, pp. 99–127, Jan. 1999.
[42] P. Bosetti, M. Da Lio, and A. Saroldi, "On the human control of vehicles: An experimental study of acceleration," *Eur. Transp. Res. Rev.*, vol. 6, no. 2, pp. 157–170, Jun. 2014, doi: 10.1007/s12544-013-0120-2.
[43] P. Bosetti, M. Da Lio, and A. Saroldi, "On curve negotiation: From driver support to automation," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 4, pp. 2082–2093, Aug. 2015.

Gastone Pietro Rosati Papini is currently a Researcher with the Department of Industrial Engineering, University of Trento, where he involved in advanced control systems applied to the field of autonomous driving. He is also the Co-Founder of Cheros Srl, a spinout company of the University of Pisa, that deals with renewable energies and ICT solutions. He has been involved in several EU projects, such as VERITAS, PolyWec, and Dreams4Cars. He has over seven years of research experience in applied mechanics, in particular in the design of advanced control systems that exploit machine learning techniques.

Alice Plebe received the B.Sc. and M.Sc. degrees in computer science from the University of Catania, Italy, in 2014 and 2016, respectively. She is currently pursuing the Ph.D. degree with the University of Trento. She is working on deep neural networks inspired by the human cognition of driving, as part of the EU Project Dreams4Cars.

Mauro Da Lio (Member, IEEE) received the Laurea degree in mechanical engineering from the University of Padua, Italy, in 1986. He is currently a Full Professor of mechanical systems with the University of Trento, Italy. His earlier research activity was on modelling, simulation, and optimal control of mechanical multibody systems, in particular vehicle and spacecraft dynamics. More recently his focus shifted to the modelling of human sensory-motor control, in particular drivers and motor impaired people. Prior to his academic career, he worked for an offshore oil research company in underwater robotics (a EUREKA Project). He was involved in several EU framework program six and seven projects (PReVENT, SAFERIDER, interactIVe, VERITAS, AdaptIVe, and No-Tremor). He is also the Coordinator of the EU Horizon 2020 Dreams4Cars Research and Innovation Action: a collaborative project in the robotics domain which aims at increasing the cognition abilities of artificial driving agents by means of offline simulation mechanisms broadly inspired to the human dream state.

Riccardo Donà received the M.Sc. degree in mechatronics engineering from the University of Trento, Italy, where he is currently pursuing the Ph.D. degree. He is currently working on the EU Project Dreams4Cars with the University of Trento. His main scientific interest is autonomous driving vehicles which constitutes the core topic of his Ph.D. research project.