Received October 10, 2020, accepted October 19, 2020, date of publication October 21, 2020, date of current version November 2, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3032780

A Mental Simulation Approach for Learning Neural-Network Predictive Control (in Self-Driving Cars)

MAURO DA LIO[1], (Member, IEEE), RICCARDO DONÀ[1], GASTONE PIETRO ROSATI PAPINI[1], FRANCESCO BIRAL[1], AND HENRIK SVENSSON[2]

[1]Department of Industrial Engineering, University of Trento, 38123 Trento, Italy
[2]Interaction Laboratory, University of Skövde, 541 28 Skövde, Sweden

Corresponding author: Mauro Da Lio (mauro.dalio@unitn.it)

This work was supported by the European Union (EU) Horizon 2020 Dreams4Cars Research and Innovation Action funded by the European Commission under Grant 731593.

ABSTRACT This paper presents a novel approach to learning predictive motor control via ''mental simulations''. The method, inspired by learning via mental imagery in natural Cognition, develops in two phases: first, the learning of predictive models based on data recorded in the interaction with the environment; then, at a deferred time, the synthesis of inverse models via offline episodic simulations. Parallelism with human-engineered control-theoretic workflow (mathematical modeling the direct dynamics followed by optimal control inversion) is established. Compared to the latter human-directed synthesis, the mental simulation approach increases autonomy: a robotic agent can learn predictive models and synthesize inverse ones with a large degree of independence. Human modeling is still needed but limited to providing efficient templates for the forward and inverse neural networks and a few other directives. One could consider these templates as the efficient brain network typologies that evolution produced to permit live beings quickly and efficiently learning. The structure of the neural networks —both forward and inverse ones— is made of interpretable ''local models'', which follows the cerebellar organization (and are also similar to local model approaches known in the literature). We demonstrate the learning of a first-round model (contrasted to Model Predictive Control) for lateral vehicle dynamics. Then, we demonstrate a second learning iteration, where the forward/inverse neural models are significantly improved.

INDEX TERMS Mental simulation, neural networks, optimal control, predictive control, vehicle dynamics.

I. INTRODUCTION

Today the predominant approach for developing self-driving cars relies on the careful engineering of behaviors carried out by expert human designers [1]–[3], where motion planning, decision making, and control follow a symbolic representation of the environment, which is populated by the perception system.

In this scheme, Artificial Intelligence (AI) may be used for perception, where the power of deep learning outclasses other approaches in vision-based object detection and classification. Deep neural networks are not immune to problems [4]–[6], which stem from being a quite opaque black box component but, because of the unparalleled performance, they tend to be accepted for the ''perception layer''. Conversely, behavior planning, decision making, and control are human-engineered because the system must be explainable, and control-theoretic methods can be used to guarantee formal properties, such as safety and stability. These systems are capable of dealing very well with the situations that were considered by the human designer but tend to lack autonomy: occasionally, situations not foreseen at the design stage may leave the system incapable of acceptable action [7]–[11].

A central aspect of autonomy, in Robotics and Artificial Cognition, is the ability of a system to behave correctly in situations that were not programmed previously. Given the

The associate editor coordinating the review of this manuscript and approving it for publication was Jianyong Yao.

outstanding performance of "AI" technologies in perception, one might wonder whether some form of Artificial Intelligence —or, better, *Artificial Cognition*— could help develop more robust and flexible behavior. However, the advantages of traditional approaches, such as explainability and interpretability, should be preserved.

One recent notable example of "AI" in driving is [12], which demonstrated a deep neural network capable of lane following in traffic. Differently than the above human-coded approaches, the network is trained end-to-end via expert human driver examples.

Whether end-to-end approaches can be generalized to produce an agent capable of driving in general contexts was considered in [13], but unsuccessfully, because of the vastness of nuanced situations and examples that would be required ("*simple imitation of a large number of expert demonstrations is not enough to create a capable and reliable self-driving technology*"). In the same study, a way to go beyond pure imitation and generate a more comprehensive set of examples via perturbations of the human demonstrations was proposed as an alternative ("*ChauffeurNet: learns by synthesizing suitable training data via perturbations of the expert driven trajectory*").

In the domain of Robotics and Artificial Cognitive Systems, the EU Horizon 2020 Dreams4Cars project [14] studied a learning approach inspired by human wake-sleep/dream cycles. At wake state, an agent collects data used for training predictive models of the world at various levels of abstractions (a.k.a. "forward models"). Then, offline (at sleep/dream state), the same agent synthesizes "inverse models" for control and behaviors via, respectively, embodied simulations (low-level detailed simulations) and episodic simulations (higher-level more abstract simulations). The first step —construction of forward models— is a supervised training phase where input-output pairs are either the sensory effects that follow specific motor commands or the states of the environment that usually follow previously observed states. The second step —synthesis of inverse models— is an unsupervised learning phase where the forward networks are inverted in a set of *ad hoc* generated episodes. Compared to the perturbation idea mentioned above, this method is more general (also Dreams4Cars was proposed earlier) and does not need any human demonstration. The agent is potentially autonomous in exploring the world, making its predictive models, and then exploiting these models for efficient synthesis of control and behaviors carried out offline in a self-created simulated safe sandbox environment (the dreams).

A. WHAT THIS PAPER IS (AND IS NOT) ABOUT

This paper illustrates a novel learning paradigm —learning via mental simulations [15] — which is inspired by "wake-sleep" cycles. Although this process may be carried out at various behavioral levels, the focus here is on learning low-level motor control. The method is illustrated with reference to the domain of autonomous driving, which was the application area of Dreams4Cars.

As said, learning via mental simulations develops in two phases: learning of predictive models, with data collected in the "wake" state, and synthesis of control, at a "sleep" state. Notably, these two steps parallel control-theoretic methods consisting of developing mathematical direct-dynamics models and their use in the synthesis of controller —specifically optimal controllers.[1]

Across the paper, we establish the similarity of the two processes to explain the proposed method in a familiar conceptual framework. The similarities guide the understanding of the method and help strengthen its validity (i.e., the workflow is well-proven).

Learning via mental simulation depart from traditional control-theoretic approaches because *ad-hoc* neural networks replace analytical plant models and because episodic simulations are used to train neural networks that enact inverse models, which functionally replace optimal control and predictive control. Of course, these two differences are intended and motivated by the desire of developing a robot able to improve its abilities autonomously, at least in part.

Drawing parallelism between control-theoretic methods and the new learning paradigm necessarily leads to some quantitative comparisons. However, they must not be interpreted here as performance-wise comparisons of the two different methods. We do not claim that learning via mental simulations outclasses traditional methods in terms of absolute performance (to do this, a great effort should be spent to optimize one application, and this would draw attention away from the core of this paper, which is the illustration of the learning paradigm).

1) CONTRIBUTION IN DETAIL

The contribution illustrates how an artificial driving agent can first learn a neural network model of the vehicle dynamics and then use that model to synthesize a predictive neural controller (Section III).

This method is contrasted (functionally, not in terms of absolute performance) with the traditional workflow used for developing a Model Predictive Controller (MPC) (Section IV).

The parallelism with the traditional workflow is further enhanced by using interpretable neural networks so that the final agent is an explainable white box.

In the second part, the paper illustrates how the forward-inverse models can be progressively extended along with the agent's life —ideally following the "wake-sleep" cycles— (Section V).

Finally, this method is opposed to control-theoretic methods and Reinforcement Leaning in Section VI.

[1] Incidentally, one might argue that engineers unwittingly developed this process as the empowerment of their innate biological process: i.e., with mathematics, they found ways to produce superb predictive models and effective ways for their optimal control.

II. RELATED WORK

A. DEEP LEARNING FOR VEHICLE CONTROL

A survey of Deep Learning methods for vehicle control has been recently proposed [16]. The review acknowledges two broad categories: supervised learning and reinforcement learning.

Supervised learning approaches consist of training a (deep) neural network by giving expert driver demonstration examples, such as, e.g., the end-to-end system mentioned in the introduction [12]. One problem of this approach is the vastness and diversity of the training set necessary "*to train a generalisable model which can drive in all different environments*". Another recently discovered issue is the so-called causal confusion, i.e., the inability to grasp the "*causal structure of the interaction between the expert and the environment*" [17]. Finally, end-to-end trained networks are little explainable [18]. For example, the features that most contribute to the steering action can be visualized [19]. However, human actions are often a superintended choice between different affordances [20], [21]. This explanation level is missing. It remains unclear which alternative actions may have been evaluated and why they have been discarded (we argue this shortfall is also related to the above-mentioned lack of causal understanding).

Reinforcement Learning (RL) approaches follow biological inspirations of the putative method for action discovery in the animal reign [22]. Instead of providing context-action examples, RL discovers and optimizes actions via (essentially) trial and error. For this, criteria for assessing the success and goodness of actions must be provided in place of context-action pair examples. When the dimensionality of state and action spaces is large, the trial and error approach tends to be very slow and sample inefficient [23]. Reinforcement Learning is an approach that discovers optimal control strategies by testing actions and observing the rewards. In principle, hence, it does not need a model of the plant. This feature is a strength of the approach because models may not always be available or might need a human modeler to be engineered (hence an agent would not learn in autonomy). However, the same feature may also be a weakness: if the plant is a physical system, testing actions on the real system may be dangerous, and training must be carried out in a simulation environment in any case, re-introducing the need for a plant model and raising the issue of transferring the policy to the real system. The need for creating plant models as a safe environment for training significantly reduces the advantage of Reinforcement Learning as a direct optimal control approach. Furthermore, if a model is required to create a safe training environment, that same model could be, in principle, exploited in a much more efficient way than trial and error. So far, no example of high-quality low-level vehicle control transferred to real vehicles is given in [16].

A notable work related to the notion of learning with self-instantiated simulations (which potentially tackles the safe sandbox issue mentioned above) is [24]. In this work, a predictive model of a car racing game is first learned and then used within a Reinforcement Learning scheme (this work will be part of the discussion in Section VI).

In a different domain (hydraulic plants), one example of the use of neural networks for the control of highly nonlinear and difficult-to-model systems is given in [25]. The study adopts a neural network to estimate the discrepancy between the plant and a nominal plant model. In this respect, the idea is similar to the augmentation method of Section V-A2.b. The neural network is a fully connected nonlinear multi-layer perceptron. For the controller synthesis, the network is manipulated symbolically (symbolic gradients) and integrated into a robust integral of the sign of the error (RISE) control scheme. The manipulation of the network is another example that if a learned model is available, it can be fruitfully exploited for control (compared to model-free approaches).

B. LEARNING VIA MENTAL SIMULATION

Human beings (and many animals) exhibit intelligent behaviors that go beyond reactive state-action associations.

A distinctive characteristic of intelligence is the ability to evaluate the consequences of several potential and alternative actions without actually executing them [26] in order to choose among the available alternatives [27] and intentionally navigating the affordance landscape for long-term goals [20]. Also, a remarkable advantage of this form of mental simulation is safely testing the consequences of actions that might be dangerous if carried out in the reality [26].

Furthermore, mental simulation abilities extend beyond this simple prediction of the course of action. They include the ability to imagine —at a deferred time— hypothetical situations to develop action strategies for them in advance. This ability allows acting correctly in conditions not met before without needing time to carry out mental simulations at that moment, which might be a time-critical context [15], [28].

The various forms of mental simulations are enabled by prior learning of internal neural predictive models [29] and the generation of hypothetical episodes [15].

Hence, while simulations can be carried out inline (i.e., during actions), mental simulations can also be carried out offline (i.e., at a deferred time) to prepare "inverse models" for behavior and control, including for yet hypothetical situations.

For a more in-depth discussion on the utility of offline mental simulations, see [28]; for a discussion of different learning mechanisms at sleep/dream, see also [30].

C. TRADITIONAL PREDICTIVE CONTROL

Many automated vehicle functions, from lane-keeping assistance to autonomous driving, require controlling the lateral dynamics of a vehicle to produce the desired path. Three general reviews concerned with motion planning and control have already been mentioned [1]–[3].

Given the desired path, there are several methods to produce vehicle control for that path (e.g., [2, Section V]). The method that is more relevant for this paper is model predictive

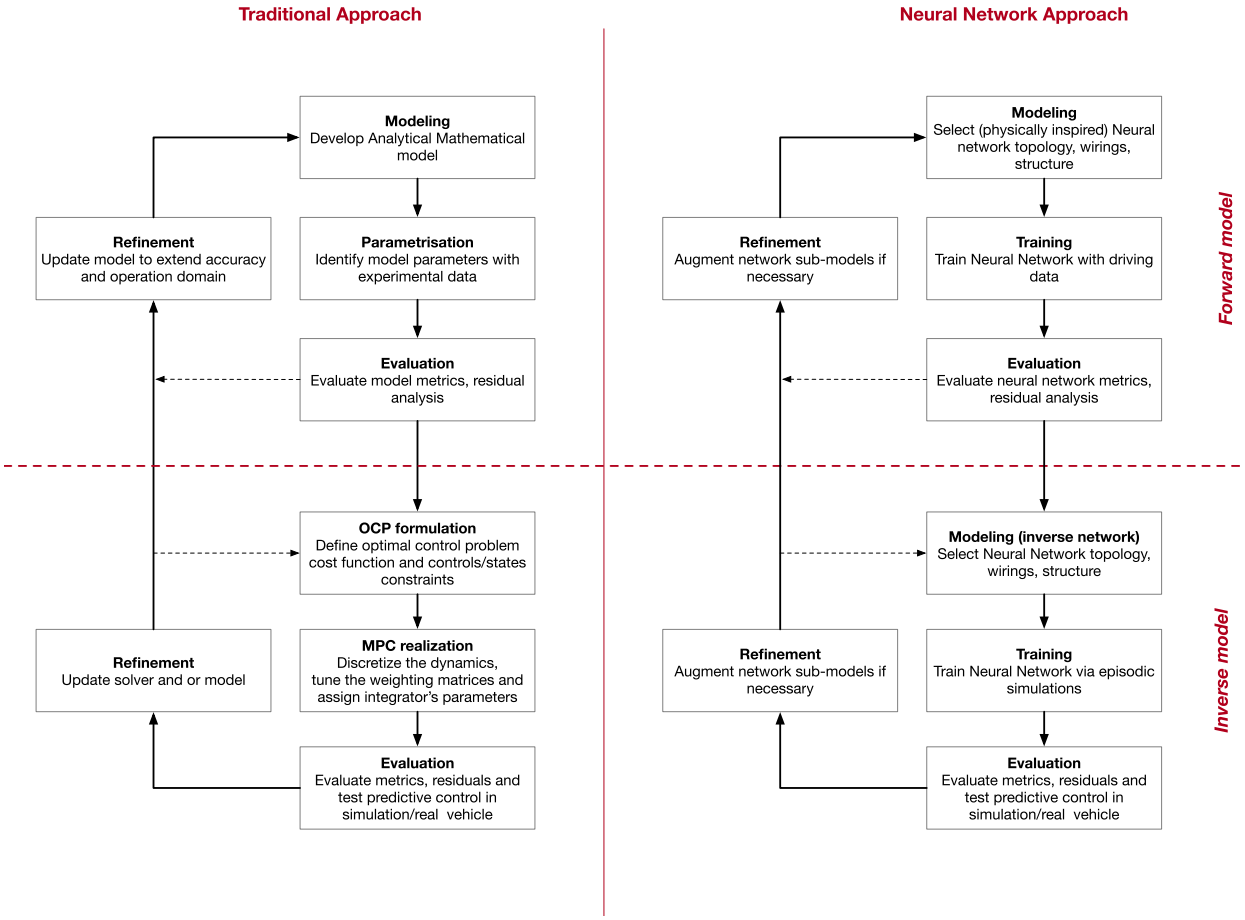


FIGURE 1. Traditional vs Cognitive approach workflows.

control (MPC) [31]–[34]. With MPC, a human-engineered mathematical model of the vehicle dynamics, usually formed by ordinary differential equations, is employed to solve a receding-horizon optimal control problem, returning the instantaneous (and future) control. Detailed models of vehicle dynamics may be nonlinear, which poses convergence issues and computational burden to MPC. An often-used workaround consists of linearizing the equations online in the current operating conditions, reducing to a linear but time-varying (LTV) MPC problem [35], which ensures fast online computation, guaranteed convergence, and stability. Piecewise linear models may also be used for the same purpose.

Apart from the linearization, one should note that the original mathematical model itself, to be tractable, brings about many necessary simplifications. For example, it adopts almost always the classical bicycle simplification, and tire models are not known precisely as the Newtonian dynamics of the chassis.

The steering actuator dynamics may add additional complexity. It is a complex mechatronic system that (just like tires) is not easily modeled from basic principles and may, instead, be modeled with a data-driven approach (for example, [33]; see also Appendix).

Hence, even without considering linearization, the vehicle model forming the foundation of MPC is approximate. Furthermore, the model parameters have to be identified, which may not be a trivial task, and state observers (for example, side slip observer) may also be necessary.

III. LEARNING VIA MENTAL SIMULATION

Fig. 1 gives an overview of how learning predictive control via mental simulation is implemented (Fig. 1, right), and how it is similar to the traditional workflow (Fig. 1, left). They both involve two distinct activities.

1) The model of the plant is learned. In the traditional approach, this is the development of the plant dynamics model. The model describes the forward dynamics (hence the name of ''forward model''), e.g., predicting the trajectory produced with a given steering/gas/brake command. In the machine learning case, the model is a trained neural network, with a particular architecture. In the traditional case, the model is, usually, a set of ordinary differential equations with parameter values.
2) The model of the inverse dynamics is synthesized. In the traditional approach, this corresponds to developing a model predictive controller (MPC). The model

describes the inverse dynamics (hence the name of ''inverse model''), e.g., returning the steering/gas/brake input necessary to pursue a future desired trajectory. In the machine learning case, the inverse model is a neural network again. A difference between the two is that the neural network is trained offline via episodic (mental) simulations, whereas MPC is solved inline. In both cases, the inversion process takes significant advantages of the previously developed predictive models so that inversion is more efficient than trial and error exploration.

A. LEARNING FORWARD MODELS

Learning forward models can be broken into three phases: 1) modeling, 2) training, and 3) evaluation (Fig. 1, top right). To refine the model these can be iterated. Similar phases exist for the traditional process (Fig. 1, top left).

1) MODELING

Neural networks can model discrete-time causal dynamical systems. At first glance, the two most apparent network architectures are shallow recurrent networks and shallow feed-forward networks.

In the case of a recurrent network, internal states $\boldsymbol{x}$ exist, and the network essentially learns a discrete-time dynamical system of the type $\boldsymbol{x}_{t+1} = \boldsymbol{F}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ and $\boldsymbol{y}_t = \boldsymbol{G}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ where $\boldsymbol{y}$ are the network outputs, $\boldsymbol{u}$ the inputs, t the discrete time and $\boldsymbol{F}(\cdot)$ and $\boldsymbol{G}(\cdot)$ are functions learned by the network. One simple implementation of this type may be obtained with a basic recurrent layer followed by a single or multi-layer perceptron, implementing respectively $\boldsymbol{F}(\cdot)$ and $\boldsymbol{G}(\cdot)$.

In the alternative case of a feed-forward network, no internal state exist, and the network computes the output given a sufficiently long history (n) of past inputs, i.e., of the type:

$$\boldsymbol{y}_t = \boldsymbol{F}(\boldsymbol{u}_t, \boldsymbol{u}_{t-1}, \ldots, \boldsymbol{u}_{t-n}). \tag{1}$$

Both solutions are general-purpose networks that may approximate any input-output map to any degree of accuracy, given sufficient size of the recurrent states $\boldsymbol{x}$ or sufficient dimensions of the hidden layers of the feed-forward networks.

However, just because they are entirely generic, they are not as efficient as possible in learning the particular types of functions holding for physical systems. In [23], Botvinick points out how networks that are biased toward particular types of functions deploy greater ''sampling efficiency''. Properly biased networks have fewer parameters than unbiased ones, which are best focused on what is necessary to approximate the system. A similar conclusion is found in [36], where neural networks with structure informed by the physical problem are more robust, accurate, and efficient than general-purpose networks.

a: FEED-FORWARD NETWORKS WITH STRUCTURE

Artificial networks can be shaped by the human designer with the choice of an appropriate network topology following physical insight.

One first technique is recognizing the superposition of effects. A feed-forward network that models K independent effects, can be written as follows:

$$\begin{aligned}\boldsymbol{y}_t = {} & \boldsymbol{F}_1(\boldsymbol{u}_{1,t}, \boldsymbol{u}_{1,t-1}, \ldots, \boldsymbol{u}_{1,t-n}) \\ & + \ldots + \boldsymbol{F}_K(\boldsymbol{u}_{K,t}, \boldsymbol{u}_{K,t-1}, \ldots, \boldsymbol{u}_{K,t-n}).\end{aligned} \tag{2}$$

This decomposition takes advantage of the fact that the individual effects $\boldsymbol{F}_i(\cdot)$ have inputs $\boldsymbol{u}_i$ that usually are small size subsets of all the system inputs $\boldsymbol{u}$.

In the Newtonian dynamics, the acceleration of a multi-body system is the superposition of the individual forces. Each force has its distinct causes (for example, the brake force depends on the brake pedal pressure but not on the gas pedal pressure [36]). Decomposing the plant into the combination of simpler sub-plants (the force effects), each with reduced dimensionality input, is a very efficient and robust modeling approach [36]. This structuring form needs only the little physical insight required to tell which forces act on the system and their causes $\boldsymbol{u}_i$. So, the total number of parameters in (2) is usually significantly smaller than in (1) because the $\boldsymbol{F}_i$ operate on reduced dimension subsets of $\boldsymbol{u}$ [36], [37].

A second technique for shaping a neural network model consists of interpolating the output of M parallel models as follows:

$$\begin{aligned}\boldsymbol{y}_t = {} & \phi_1(\boldsymbol{x})\boldsymbol{F}_1(\boldsymbol{u}_{1,t}, \ldots, \boldsymbol{u}_{1,t-n}) \\ & + \ldots + \phi_M(\boldsymbol{x})\boldsymbol{F}_M(\boldsymbol{u}_{M,t}, \ldots, \boldsymbol{u}_{M,t-n}),\end{aligned} \tag{3}$$

where the $\boldsymbol{F}_i$ may be models like (1) or individual components of (2), weighted according to $\phi_i(\boldsymbol{x})$, where $\boldsymbol{x}$ is a convenient set of states.

There are various possibilities of implementation with (3), depending on how functions $\phi_i(\boldsymbol{x})$ are realized. One option is that the $\phi_i(\boldsymbol{x})$ are functions with local support, i.e., $\phi_i(\boldsymbol{x})$ is non-zero only in a neighbourhood of a given point $\boldsymbol{x}_i$. In this case (3) is the combination of *local* models. The neighbourhood is also called a ''receptive field'' (a term that actually comes from biology), while the $\phi_i(\boldsymbol{x})$ may be also called ''activation functions'' or ''validity functions''.

In local model approximations, the functions $\phi_i(\boldsymbol{x})$ may be subject of learning together with the individual sub-models $\boldsymbol{F}_i$. This is, for example, the case of methods like the locally weighted projection regression (LWPR) algorithm [38], or of the local linear model trees (LOLIMOT) [39].

As an alternative, the receptive fields $\phi_i(\boldsymbol{x})$ may be pre-defined (based on some prior knowledge) and only the $\boldsymbol{F}_i$ are learned. This may be the case of so called channel coding approaches, in which individual channels *partition* the state space $\boldsymbol{x}$ [40]–[43].

There are advantages and disadvantages to both approaches. On one side, adaptive receptive fields favor incremental learning and contribute to better approximations by gathering the receptive fields where needed. On the other hand, we argue that, if modeling is driven by physical insight, it may be preferable to maintain some control over the meaning of the space $\boldsymbol{x}$ and its partition [43]. Also, learning individual sub-models $\boldsymbol{F}_i$ is less affected by imbalanced data-sets if the learning domain is partitioned. One example of local models with fixed receptive fields is given in [36], [37] where the drive-line sub-models are learned for different gears independently.

The local models $\boldsymbol{F}_i$ may be linear, i.e.:

$$\boldsymbol{F}_i(\cdot) = \boldsymbol{b}_i + \boldsymbol{w}_{i,1}\boldsymbol{u}_{i,t} + \ldots + \boldsymbol{w}_{i,n}\boldsymbol{u}_{i,t-n}, \tag{4}$$

where $\boldsymbol{b}_i$ are possible biases and $\boldsymbol{w}_i$ the finite impulse response (FIR) coefficients of the approximated dynamical system. So, with (4), (3) is also known as an NFIR scheme (N models of FIR type). A comparison of NFIR with other schemes such as NARX (N models with autoregressive, AR, component) may be found in [44]; for a comparison of (3) with recurrent networks and fully connected multi-layer perceptrons see also [36].

b: SIMILARITY WITH CEREBELLAR NETWORKS

It is interesting to note that cerebellar networks implement a similar —but more sophisticated— principle of combination of sub-models.

Fig. 2 shows schematic wiring of two elementary cerebellar filters. The output —now scalar— of one filter, say $y_{1,t}$ is produced by one Purkinje cell, that computes the weighted sum of the signals found on the parallel fibres $p_{i,t}$, *i.e.*, $y_{1,t} = \lambda_1\, p_{1,t} + \ldots + \lambda_N p_{N,t}$. The weights λ_i can be adapted quickly through training signals carried by climbing fibers (not shown) [45]. Each parallel fiber is the axon of one Granular cell, which receives input signals from the Mossy fibers. Mossy fibres may carry the same scalar signal sampled at different past times (delayed copies of that signal) such as, e.g., $u_{1,t}, \ldots, u_{1,t-n}$ as well as signals of different types such as, e.g., u_1, u_2 etc. These signals may have various origins, representing sensory data, brain states, and motor commands.

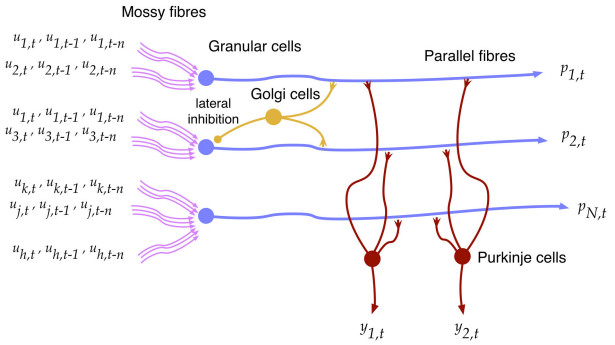


FIGURE 2. Cerebellar neural network wiring from [29].

However, each Granular cell receives a limited number of inputs. For example the granular cell in the middle of Fig. 2 receives only the scalar inputs u_1, u_3. Thus, the Granular cells individually operate as detectors of "relatively simple contexts" [29] and their output may be written as $p_{i,t} = F_i(\boldsymbol{u}_i)$ where vector $\boldsymbol{u}_i$ collects all relevant scalar inputs (e.g., $\boldsymbol{u}_2 = \{u_1, u_3\}$ for cell number 2). The functions $p_{i,t} = F_i(\boldsymbol{u}_i)$, also called basis functions may be nonlinear. Each Purkinje cell may receive input from as many as one million parallel fibers, thus learning to combine a vast number of basis functions. The same parallel fiber serves many Purkinje cells. A final mechanism that allows increasing the network computation complexity is given by cross inhibitions with the Golgi cells. This way, the signal on one parallel fiber can inhibit the output of another Granular cell/fiber, in practice realizing the context-related activation/inhibition equivalent to the functions $\phi(\boldsymbol{x})$.

c: A NEURAL LTV MODEL FOR VEHICLE LATERAL DYNAMICS

We now use the principles above for constructing a neural network capable of learning a Linear Time-Varying (LTV) model of a vehicle's lateral dynamics (see [36], [37] for a different example).

Physical inspiration for organizing the network connectivity can be obtained from the analysis that precedes classical mathematical modeling, e.g., in [46]. The primary goal is to note the causal input (steering angle causing side forces on tires) and the time-varying parameters that influence the dynamics (velocity). A secondary goal is also to understand assumptions and simplifications to reveal other possible inputs and nonlinear phenomena. For example, this leads to note that, besides tires, other forces might affect the lateral dynamics. A list should count lateral wind, gravity gradients within roads with lateral inclination, lateral component of the traction force in steered wheels, yaw moment caused by differential traction of right/left wheels, vehicle asymmetries, and others. Concerning the nonlinear phenomena, tires nonlinear characteristics, the influence of traction forces, and geometric nonlinearities are prominent. This plethora of aspects that would require individual sub-models elucidation in the traditional approach (traditional models are, for this reason, always a trade-off between complexity and modeled details), need here to be annotated only in terms of input-output and linear-nonlinear characteristics.

The process of developing a neural network model can be carried out incrementally. As a first step, one can start with a simple model, e.g., a network producing a linear time-varying system parametric in the velocity. The network may be later augmented (Section V) to account for nonlinearities and additional inputs that have been noted but not used in the first round.

Now, let the input be the steering wheel angle δ and the output be the curvature κ of the vehicle trajectory (κ is the lateral acceleration $\ddot{y}$ divided by the square traveling velocity v^2, i.e., $\kappa = \ddot{y}/v^2$).

An inspiring traditional mathematical model that helps defining an initial network structure may be the single-track model [46, Chapter 3]. In terms of input-output frequency response it reads as in [46, Equation (3.87)'], i.e., with some adaptations:

$$\kappa(s) = \frac{G_0}{1 + Av^2} \frac{1 + T_1(v)s + T_2(v)s^2}{1 + b_1(v)s + b_2(v)s^2}, \tag{5}$$

which is a second order transfer function with poles, zeros and gain that are function of the velocity (v).

This dynamical system is approximated with an NFIR model enacted by a network with the connectivity shown in Fig. 3, where branch "1" computes M linear models of type (4) and branch "2" computes the M activation functions $\phi_i(v)$ which, in turn, are modeled with:

$$\phi_i(v) = \frac{1}{1 + Av^2} \Lambda\left(\frac{v - v_i}{\Delta v}\right), \tag{6}$$

where v_i are the centres of the receptive fields, Δv the receptive fields radius and $\Lambda(\cdot)$ is the Heaviside Lambda function: $\Lambda(x) = \max(1 - |x|, 0)$. The scaling factor $1 + Av^2$, with learnable A, is introduced to conveniently represent the common under steering gradient (A) effect. With (6) the network in Fig. 3 interpolates linearly between the impulse responses learned at the receptive field centres v_i.

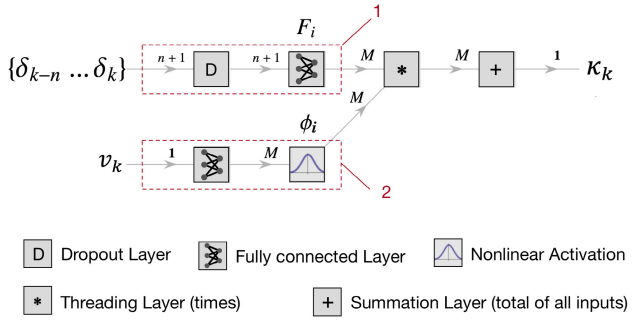


FIGURE 3. Neural network connectivity for LTV lateral dynamics.

2) TRAINING (OF THE FORWARD NETWORK)

Forward models are trained via supervised learning. For the collection of the training sets $\{\delta, v\} \rightarrow \{\kappa = \ddot{y}/v^2\}$ there are several possibilities.

In principle, one might borrow the human approach, especially infants, when facing an unknown object or plant, i.e., *motor babbling*: the plant is stimulated with small input to observe its reaction, beginning to create input-output pairs. With careful execution, this approach allows bootstrapping *ab initio* a model of a plant that can be gradually enriched by probing wider and wider stimuli sets, e.g., [47].

While this approach is conceptually interesting, a quicker start is possible for an autonomous vehicle because a car can be driven with a simple controller, or manually, to produce command-response examples representative of the control task to be solved.

In Dreams4Cars, there were two test vehicles: a research platform based on a small electric vehicle and a production vehicle, which was a Jeep Renegade. The latter is considered here. It was driven on a safe test track (GPS 45.0098, 7.5604) using the lane-keeping assistance function with a modified steering actuator to collect the training sets. The "requested steering wheel angle" command (input to the modified steering actuator) was logged together with the produced steering wheel angle and the output vehicle dynamics (yaw rate, accelerations, velocity). The same vehicle was also modeled in the IPG CarMaker[2] environment for development and software/hardware in the loop testing —in the following referred as "model in the loop" (MIL). The CarMaker vehicle model is a multi-body system, with many degrees of freedom, parametrized with design data known by the manufacturer. Its dynamics is significantly more complex than typical single-track models used for control design. The real-world experiments and the fidelity of the MIL simulations are discussed in Appendix .

However, for the rest of the paper, we carried out *ad hoc* new CarMaker simulations that allowed developing the many variations presented below. For these, we used a slightly different simulated vehicle, without the confusing steering actuator (see discussion in Appendix). The simulations of this paper are referred below as "virtual vehicle" to make a distinction with Dreams4Cars MIL.

a: TRAINING AND VALIDATION SETS

Table 1 lists the data sets (labeled: 1-7) used in this paper and obtained by driving the virtual vehicle in CarMaker with an initial version of the controller.

The training and validation sets span different types of scenarios and track-layouts, as described in the following.

There are two "highway" scenarios with random traffic: 1 is a low-density traffic situation used in the training set and 7 is a higher density traffic case used in the validation set. The road layout is a hypothetical two-lanes oval with two long straights and two 500 m radius curves. The agent overtakes slower vehicles when possible in the straights ([48] beginning at time 3:53). Lane changes, and related trajectories, are different in the training and validation set. Furthermore, because of denser/slower traffic, the speed interval (v range) of the two is different.

There are also four "safety-center" scenarios. They correspond to the virtual version of the test track mentioned above (Fiat Centro Sicurezza, Cascina Mellano, Torino, Italy; see also [48] beginning at time 3:29). The scenarios/data labeled 2, 3, and 4 are part of the training set; 6 is part of the validation set. The track is a two-lane road without traffic that is driven at an average speed (3 and 6), slow speed (2), and fast (4). There are several curves with different radius (the sharpest one is 60 m radius). Cases 3 and 6 are different

[2]https://ipg-automotive.com/products-services/simulation-software/carmaker/

TABLE 1. Training and validation sets.

Training set						
ID	type	v range (m/s)	max curvature (m^{-1})	max a_y (m/s^2)	length (m)	samples
1	highway-traffic	25.2 - 35.5	0.0020	1.476	4907.4	3868
2	safety-centre slow	12.5 - 19.3	0.0167	2.785	1945.5	2393
3	safety-centre babbling	13.6 - 19.4	0.0167	4.024	1940.9	2313
4	safety-centre fast	14.2 - 19.5	0.0167	4.518	1906.0	2230
5	"square" track	8.8 - 14.0	0.0500	4.519	3249.0	4769
Validation set						
ID	type	v range (m/s)	max curvature (m^{-1})	max a_y (m/s^2)	length (m)	samples
6	safety-centre normal	13.6 - 19.4	0.0167	3.391	1847.4	2214
7	highway-traffic variant	25.6 - 33.0	0.0020	1.479	4981.0	3605
8	lane change	15.0 - 27.8	0.0000	1.217	744.0	595

because, albeit the speed is the same, in 3 there is Gaussian noise superimposed to the steering wheel command.

The "square track" 5 is a hypothetical square path with four curves of increasing curvature, with the radius as small as 20 m. There is no traffic. This scenario is only present in the training set.

The "lane change" 8 is a two-lanes straight road where the vehicle is initially traveling at 90 km/h and must pass between two standing vehicles: the first on the left lane and the second on the right lane 50 m ahead. For the lane change, the agent modulates the speed significantly. This scenario is not present in the training set.

Overall the training set spans the speed interval of $30-120$ km/h with lateral accelerations that slightly exceed the typical acceleration used by humans in curves [49], [50], up to 4.5 m/s^2. Phases with significant longitudinal acceleration, within the typical range of normal human driving styles [49], are also present.

b: REGULARIZATION AND RELATION WITH LEAST SQUARE FIR ESTIMATION

In essence, the neural network of Fig. 3 models the linearised lateral dynamics by learning the impulse response parametric in the velocity v. The training is carried out with stochastic gradient descent algorithms that minimize a given loss function. As long as the network remains linear (nonlinear augmentation is discussed in the following) and the loss function is the default mean square loss, the training goal is equivalent to a least-square fit. Least-square estimation of FIR models may be noisy and require regularization. This is well explained theoretically in [51], [52].

In deep learning frameworks, several methods are available for regularization: dropout layers, L2 regularization, custom loss functions and use of validation sets. Dropout with small probability may be used to counter high-frequency noise. L2 regularization may be used to minimize the magnitude of the FIR coefficients and, with some expedient (threading input and output over given weights vector), it is also possible to minimize the FIR coefficients with exponentially decaying weights, such as in [51, Equation 26], which works for stable systems. Custom loss functions offer the maximum freedom to regularise, for example, penalizing the derivatives of the output to impose smoothness (e.g., implementing the equivalent of cited kernel methods). Finally, stochastic descent algorithms can mitigate over-fitting by measuring the network's performance on a secondary test data —the validation set, which is not otherwise used for training— in such a way that only the training steps that improve it are finally retained. This rejects noise that happens to be different in the training and validation sets. The first three approaches require the definition of regularization hyperparameters (dropout probability, L2 weights, custom loss terms). The latter does not (see [51], [52] for optimal hyperparameters choice).

In the examples presented in this paper, the impulse response is relatively short (1.5 s corresponding to 30 samples at 20 Hz rate), the CarMaker environment data have little noise, and the regularization issue is not severe. In practice, of the regularization approaches mentioned above, it was sufficient to use only the latter method (the validation test data).

3) EVALUATION

a: TRAINED NETWORKS

Table 2 lists the performance metrics of two trained networks evaluated on the validation set.[3] The first is a single impulse response model ($M = 1$). The second ($M = 3$) interpolates between three impulse responses with receptive field centres at $v_1 = 0$, $v_2 = 19$ m/s, $v_3 = 38$ m/s, and with $\Delta v = 19$ m/s.

Table 2 reports the fraction of variance left unexplained by the models (FVU) and the root mean square error of the predicted curvature (RMSE). Both are better for $M = 3$. Since there is virtually no sensor noise in the CarMaker environment, these figures must be interpreted as the approximation when modeling the CarMaker virtual vehicle.

Akaike's Information Criterion (AIC) is also reported. It is an information-theoretic criterion that weights model accuracy versus model complexity. It is commonly used for model

[3]The table lists also the metrics of the Single Track classical model, which is commented later.

TABLE 2. Trained models performance metrics (on the Validation set).

Validation set				
Model	Params.	AIC	FVU	RMSE (km^{-1})
Network (M=1)	31	-12103.2	0.097%	0.095
Network (M=3)	91	-17562.5	0.036%	0.061
Single Track	5	-11957.3	0.110%	0.097

selection (the lower, the better) [53]. Given two models, the quantity $\exp((\Delta AIC)/2)$ is the relative likelihood of the two. Hence model $M = 3$ is the most likely.

Fig. 4, which shows the impulse response of the latter at two different velocities, reveals that the shape of the impulse response varies. The overall magnitude also varies because of the under-steering gain $(1/(1+Av^2))$. So, the model $(M = 3)$ is an LTV model that adapts to velocity both in magnitude and shape of the impulse response, which corresponds to the variation of both gain and poles/zeros in (5). Network $(M = 1)$ would be a simpler LTV model adapting only in magnitude. Both have the same filter length $(n + 1 = 30)$, which corresponds to 1.5 s, a figure informed by the typical time constants of the lateral chassis dynamics. This length is enough as also evident from Fig. 4.

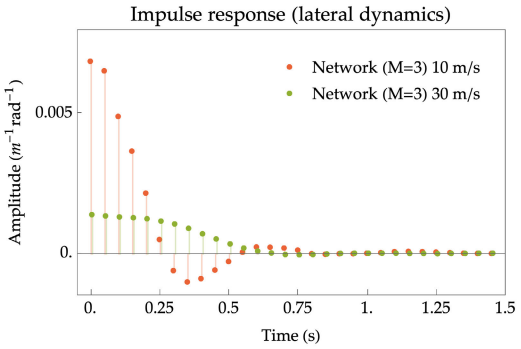


FIGURE 4. Impulse response at different velocity of the trained network ($M = 3$).

b: ITERATIONS

The dashed arrow in Fig. 1 indicates iterations that may be necessary at this point if the trained forward models are not satisfactory. Here, we assume that $M = 3$ is selected to develop a first-round inverse model and continue with the following section.

B. SYNTHESIS OF INVERSE MODELS

1) "MENTAL SIMULATIONS"

Once a forward model is learned, how can an agent produce the "corresponding" inverse model? One possible process is via "mental simulations". As outlined in Fig. 5, the inverse-model-to-be (white block) is coupled to the formerly learned forward model (cyan block). Let $\kappa_i(t)$ be an "episode", i.e., an imaginary output that the agent might wish to produce. Ideally, the inverse model should convert the desired output $\kappa_i(t)$ into its generating control $\delta_i(t)$. During the training process, $\delta_i(t)$ can be tested on the forward model, obtaining a prediction $\bar{\kappa}_i(t)$. This testing step is, in essence, a simulation carried out on the learned forward model (more precisely, it is an "embodied mental simulation" [15]).

The training procedure consists of updating the weights of the inverse model network to minimize the distance between $\kappa_i(t)$ and $\bar{\kappa}_i(t)$ on a suitable set of episodes $i \in \{Episodes\}$.

This is an unsupervised learning problem, similar to the training of auto-encoders where $\bar{\kappa}_i(t) \rightarrow \kappa_i(t)$ (of course, during training, only the weights of the inverse model are updated).

The weights of the inverse model network are updated with gradient descent, where gradients for the network under training are informed by gradients generated (propagated) across the learned forward model. In this way, the training of the inverse model is driven by the forward model.

With training, the cascade network (inverse plus forward models) becomes as close as possible to a unitary plant. This means that the inverse model realizes the best approximation of the inverse dynamics of the plant.

From a cognitive perspective, the scheme realizes an elementary form of "mental simulation" [26], similar to those that may happen in motor learning at sleep [30].

From an engineering perspective, the scheme belongs to the class of model-based optimal control, but the plant model is mostly self-generated.

2) CREATION OF EPISODES

Mental simulations (Fig. 5) require the definition of a suitable set of episodes. Together, they define the domain where the inverse/forward model cancellation is forced to occur.

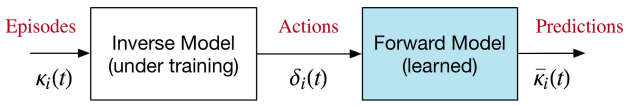


FIGURE 5. Principle of learning via mental simulations.

A first possibility is generating episodes by extracting salient manoeuvers from the training set. In this way, the inverse model will work on the same real-life situations used for learning the plant dynamics.

However, one might wish to guarantee the inversion on a broader set of conditions.[4] Perhaps conditions not met yet and perhaps somewhat dangerous, but which might happen.

Generating a broader set of episodes aims at preparing a controller for testing the plant in new conditions at the borderline of the experiences collected so far. It might also happen that, even if the forward model was trained on a given set of conditions, the model might have some extrapolation

[4]Note that, if one were to train the inverse model by using only extractions from experienced events, there would be no need at all for the scheme of Fig. 5, because it would be possible to train the inverse network directly with the examples $\kappa(t) \rightarrow \delta(t)$ [45, Section 3.2].

abilities. So, a controller that works beyond the simple set of experienced conditions may again be desirable.

a: CROSSOVER OPERATOR

A way for going beyond the realm of reality in a controlled way is using genetic operators, particularly the crossover operator. General examples of combining concepts encoded in segregated parts of a latent space representation are given in [54], [55].

In the case of this paper, the crossover operator is used for combining two randomly selected recorded trajectories, i and j in a novel one as follows:

$$e_{i,j}(t) = \kappa_i(t)w(t - t_0) + \kappa_j(t)(1 - w(t - t_0)), \qquad (7)$$

where $w(t - t_0)$ is a function switching from 0 for $t < t_0$ to 1 for $t > t_0$.

A convenient choice for the switch function $w(\cdot)$ is a low-pass filtered unit step, with filter cutoff frequency f_0. Low pass filtering prevents introducing high-frequency components at which the forward model may not be accurate. The choice of f_0 hence sets the frequency up to which the mental simulation process seeks to invert the plant model.

In terms of dreaming, the interpretation of the episode $e_{i,j}(t)$ may be imagining of pursuing the path $\kappa_i(t)$ that, surprisingly, becomes path $\kappa_j(t)$ around time t_0.

b: MUTATION OPERATOR

With crossover, a large number of episodes can be generated starting from recorded events. However, more exotic events can be created using other genetic operators, such as, e.g., the "mutation" operator where the $\kappa_i(t)$ may be altered (for example, scaled before crossover, reversed, stretched etc.). We do not use the mutation operator in this paper, albeit we felt essential to mention it. Episodes based on design standards (e.g., step, sine, target trajectories) may also be introduced.

3) INVERSE NETWORK ARCHITECTURE

An inverse model neural network aims at solving a predictive control problem in discrete time: given a desired plant output $\{\boldsymbol{y}_t, \boldsymbol{y}_{t+1}, \ldots\}$ return the input $\{\boldsymbol{u}_t, \boldsymbol{u}_{t+1}, \ldots\}$ to produce it. To solve this problem, the initial state $\boldsymbol{x}_t$ of the plant must also be given.[5]

To avoid using recursive states, in a fashion similar to (1), $\boldsymbol{x}_t$ can be replaced by related observable signals: for example the past input $\{\boldsymbol{u}_{t-1}, \ldots \boldsymbol{u}_{t-n}\}$. Hence, like for (1), we seek learning a non-recursive network that returns $\boldsymbol{u}_t$ given the past input $\{\boldsymbol{u}_{t-1}, \ldots \boldsymbol{u}_{t-n}\}$ (replacing an estimator of $\boldsymbol{x}_t$) and the desired trajectory $\{\boldsymbol{y}_t, \boldsymbol{y}_{t+1}, \ldots\}$,

$$\boldsymbol{u}_t = \boldsymbol{F}(\boldsymbol{y}_t, \boldsymbol{y}_{t+1}, \ldots, \boldsymbol{y}_{t+n}\,;\, \boldsymbol{u}_{t-1}, \ldots, \boldsymbol{u}_{t-n}). \qquad (8)$$

In a receding horizon implementation the network can be re-used for computing $\{\boldsymbol{u}_{t+1}, \boldsymbol{u}_{t+2}, \ldots\}$ iteratively.

[5] As mentioned in Section III-A1, the forward dynamics can be described conceptually with $\boldsymbol{x}_{t+1} = \boldsymbol{F}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ and $\boldsymbol{y}_t = \boldsymbol{G}(\boldsymbol{x}_t, \boldsymbol{u}_t)$. Hence the output $\{\boldsymbol{y}_t, \boldsymbol{y}_{t+1}, \ldots\}$ is caused by $\{\boldsymbol{u}_t, \boldsymbol{u}_{t+1}, \ldots\}$ and by $\boldsymbol{x}_t$.

Considerations similar to those given in Section III-A1 also hold here: in particular, decomposition into local models and implementation with a cerebellar-like neural architecture (Fig. 2), here made of one basis function for the target trajectory and one for the initial state.

Specifically, the inverse network studied here adopts the simple structure of Fig. 6, which produces a linear version of (8). It has two sub-models: one branch for the initial conditions (an Auto-Regressive linear model of the vehicle self dynamics) and another branch for pursuing the target trajectory. The latter accounts for velocity effects with layer ϕ_0, threading v, and κ into $(1 + Av^2)\kappa$, which models the under-steering gradient effect re-using the understeering coefficient A from the forward dynamics.

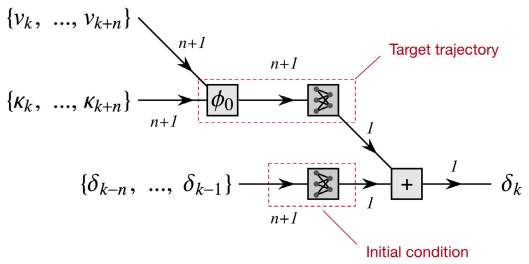


FIGURE 6. Neural network architecture for the inverse model.

Notably, the target trajectory branch of this network is made of a single FIR model. In fact, inverse model networks do not necessarily need to be as complex as the forward ones: one might wish using simplified networks that still fit the inverse dynamics acceptably in a given domain defined by the set of episodes. Hence we investigate here a 1-FIR inverse network (for a 3-FIR forward model). Upgrade to 3 local models will be demonstrated in round 2 (Section V).

4) TRAINING (OF THE INVERSE NETWORK)

The training episodes are generated from samples of 300 points (15 s) randomly extracted from the training set and combined with the crossover operator (7). The number of episodes is 13853 for a total number of training points of about four million.

To implement the principle of Fig. 5, some adaptations are necessary. First, because the networks of Fig. 2 and Fig. 6 are not recursive, first-in-first-out buffers are required to form the input vectors of the inverse network (Fig. 6). Then, the scalar output of the inverse network is also collected in a buffer forming the input vector of the following forward network. Finally, a one-step delayed version of that output buffer is recursively fed back as the past steering wheel input to the inverse model (effectively realizing a recursive ARX architecture).

The loss function is computed as the mean square difference between aligned $\bar{\kappa}_i(t)$ and $\kappa_i(t)$ after dropping the first $n + 1 = 30$ samples to discard initial transient response.

The validation episodes are formed in the same way as the training ones, but taking samples from the validation set.

A final comment concerns velocity: albeit not shown in Fig. 5 episodes need the specification of the velocity $v(t)$ at which the curves are driven. Hence, the episodes are actually formed by $\{e_{i,j}(t), v_i(t)\}$, where $v_i(t)$ are extracted from the recorded events of the training/validation sets.

The computational burden for training (13853 episodes for 300 sample points each) is about two hours on an Intel i9 CPU (41 minutes on a Titan Xp GPU), which is about 50 times (15 times for the GPU) slower than the training of the forward model (15348 samples, 2.7 min).

5) EVALUATION

a: QUALITY OF INVERSION

One first evaluation criterion concerns how well the inverse network cancels the forward dynamics. This may be of particular interest, especially if we use a simplified inverse network (like the one we are investigating here).

After training, the coupled inverse-forward models shown in Fig. 5 forms a quasi-unitary plant (see [56] for a study of the impact of imperfect cancellation when the inverse network is used within a rapid re-planning receding horizon scheme). Table 3 lists the residuals of the forward-inverse model cancellation on two different domains: the validation *set* (Table 1, ID 6-8) and the wider validation *episodes* (formed with the crossover operator as explained in Section III-B4).

TABLE 3. Quality of the inverse models.

	Params.	Validation episodes RMSE (km^{-1})	Validation set RMSE (km^{-1})
Network (Fig. 6) cancelling Network (M=3)	60	0.307	0.064

The validation set is the most significant one because it represents situations closer to what might really happen. One might notice that the root mean square error is similar to the error of the forward model (Table 2).

Conversely, the validation episodes, formed via the crossover operator, are more extreme hypothetical situations. Here, the inversion is of lesser quality, which can be regarded as an upper bound indication for inversion errors in more complex situations than those observed in the recorded sets.

b: INVERSE MODEL ACCURACY

A second evaluation criterion concerns the accuracy of the inverse model in itself. This can be evaluated on the validation set for which the actual steering wheel angle is available for comparison. Table 4 lists the performance metrics of the inverse model. The root mean square error on the steering wheel angle is 0.005 rad (0.3 deg). The fraction of variance left unexplained is similar to model $M = 1$ in Table 2 (note the inverse network has 60 parameters instead of 61 because A is inherited from the forward model and not trained).

TABLE 4. Trained inverse model performance metrics.

Validation set			
Model	Params.	FVU	RMSE (rad)
Network (Fig. 6)	60	0.11%	0.005
MPC	12	0.23%	0.010

c: COMPUTATIONAL PERFORMANCES

The inverse model network computing time is $\approx 2 \cdot 10^{-6}$ s. For comparison, each iteration of the RTI scheme in the next MPC computation takes $\approx 1 \cdot 10^{-5}$ s. Both on an Intel i7 2.9 GHz processor.

IV. TRADITIONAL APPROACH

This section reviews the traditional model-based approach (Fig. 1, left) with the intent of contrasting learning via simulation to this familiar process.

A. MODELING THE PLANT DYNAMICS

1) CHOICE OF A VEHICLE MODEL

The choice of a model for a vehicle dynamics application is a very critical trade-off between model simplicity and descriptive capacity. Although many models are available in the literature when adopting/adapting one, an essential point is realizing the modeling assumptions, simplifications, and limitations. Even so, the model adequacy for the specific application may not be immediately evident, and it may be necessary to test different model versions.

A popular model for lateral dynamics is the single-track model [46, Chapter 3]:

$$\begin{cases} \dot{\beta}(t) = -\left(1 + \dfrac{(l_f K_f - l_r K_r)}{mv(t)^2}\right)\Omega(t) \\ \qquad\qquad + -\dfrac{K_f + K_r}{mv(t)}\beta(t) + \dfrac{K_f}{mv(t)}\dfrac{\delta(t)}{\tau} \\ \dot{\Omega}(t) = -\dfrac{l_f^2\, K_f + l_r^2\, K_r}{Iv(t)}\Omega(t) \\ \qquad\qquad + -\dfrac{l_f K_f - l_r K_r}{I}\beta(t) + \dfrac{l_f K_f}{I}\dfrac{\delta(t)}{\tau}. \end{cases} \tag{9}$$

The model (9) has two states: the yaw rate $\Omega(t)$ and the side slip angle $\beta(t)$ and is controlled via the steering wheel angle $\delta(t)$. The model is linear and parametric in the travelling velocity $v(t)$ and can be converted to the transfer function representation (5).

The model has seven parameters: vehicle mass m, yaw central moment of inertia I, front and rear axles location (l_f and l_r) in a barycentric reference frame, front and rear tire stiffness (K_f and K_r) and the ratio τ between the rotation of the steering wheel and the steering rotation of the front wheels.

A non-exhaustive list of simplifications and assumptions of this model may be: 1) the right and left wheels are lumped into one virtual central wheel (the bicycle assumption).

Consequently, the effects due to vehicle body roll are ignored. Also, the effects due to differential traction/braking on right/left wheels are neglected. 2) the model is not coupled with the longitudinal dynamics. In particular, the traction force side component on the front wheels is neglected. 3) pitch oscillations are also ignored. Consequently, vertical loads on the front and rear wheels are assumed to be constant. 4) geometric nonlinearities (for example, finite steering angles) are linearised. The change of wheel axles orientation during suspension's travel is ignored. The steering wheel linkage is not precisely linear (the transmission ratio τ is not constant and is influenced by the suspensions). 5) the tire model is a simple linear model. However, tires are complex nonlinear systems difficult to model and parametrize: tire cornering forces depend on the vertical load, longitudinal slip, side slip, camber angle, tire-road friction, etc.. 6) tires have own dynamics: forces are not produced instantaneously but need some (short) time to build up. 7) in automated driving applications, the steering wheel angle is driven by an actuator that introduces dynamical effects that should also be considered.

If a few only of those assumptions were to be removed, the model complexity would quickly explode (see, for example, [34]). As a final consideration, analytical models are derived from physical principles, but they still need to incorporate data-driven sub-models: for example, tires (the widely used Pacejka model is a fitted formula).

2) MODEL PARAMETRISATION

A few of the parameters in (9), such as mass m and wheelbase $l = l_f + l_r$ can be measured easily. Others can be measured with somewhat more complex procedures (moment of inertia, the position of the center of mass, etc.). Full Pacejka models require expensive testing procedures. As an often used alternative, models may also be parameterized with standard manoeuvers like steering-pad and slalom tests.[6]

For this paper, the model mass m and wheelbase $l_f + l_r$ are known. The remaining five parameters are obtained by fitting the model onto the training set (Table 1) with a prediction error method that needed approximately 70 minutes on an Intel i9 CPU.

3) EVALUATION/UPDATES

The performance metrics of the fitted single-track model are listed in Table 2. Overall they are similar to the network with $M = 1$, but the kind of approximations are different. The single-track model adapts to velocity both in gain and poles/zero (5), whereas Network $M = 1$ has an impulse response that does not vary with velocity.

On the other hand, with five parameters, the single-track model transfer function is restricted to span a subset of all possible transfer functions/impulse responses. Note, in particular, that (5) is an improper transfer function with the same order at numerator and denominator. It predicts an instantaneous response component that is physically implausible.

The differences are also revealed by the bode plots of the single-track and neural-network models (in this case, the most precise $M = 3$), shown in Fig. 7. The two modeling approaches are equivalent below 1 Hz but the neural network —with more parameters to learn the impulse response— captures higher-order dynamics that the rigid structure of (5) cannot.[7]

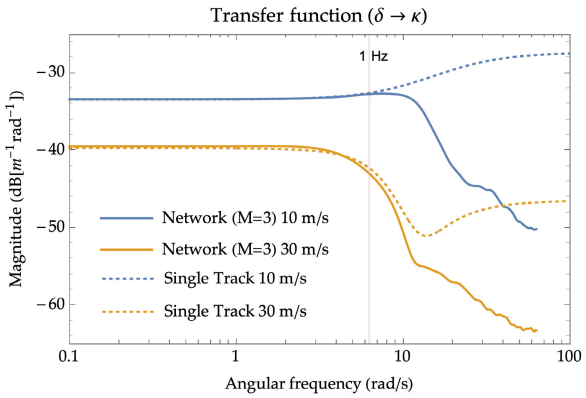


FIGURE 7. Transfer function $\delta \to \kappa$ of the trained network ($M = 3$) and of single-track model. The neural network captures higher-order dynamics.

The single-track model is hence oversimplified at high frequency, predicting implausible instantaneous response components. Nonetheless, finding what is missing is an art needing acute physical modeling abilities. Obviously (9) and (5) neglect some higher-order dynamics effect, but which? Reconsidering the list of assumptions and simplifications listed in Section IV-A1, one might restrict the possibilities to hypothesis 1 (vehicle roll slows lateral response) or hypothesis 6 (tire relaxation length slows the build-up of lateral forces). Both would require a significant increase in model complexity.

In the continuation of the paper, we will not extend the single-track model because our goal is not optimizing models but, instead, to illustrate the process used for developing neural network control and contrast this the traditional process.

B. MODEL PREDICTIVE CONTROL REALIZATION

The typical formulation for trajectory tracking MPC reads as:

$$\underset{\mathbf{x},\,\mathbf{u}}{\text{minimize}}\, J = \sum_{k=0}^{N-1} \left\| h\left(\mathbf{x}_k, \mathbf{u}_k\right) - \tilde{\mathbf{y}}_k \right\|^2_{\mathbf{W}_k} + \left\| h_N\left(\mathbf{x}_N\right) - \tilde{\mathbf{y}}_N \right\|^2_{\mathbf{W}_N} \tag{10a}$$

$$\text{subject to: } \mathbf{x}(0) = \mathbf{x}_0 \tag{10b}$$

$$\mathbf{x}_{k+1} = f\left(\mathbf{x}_k, \mathbf{u}_k\right), \quad \forall k = 0 \ldots N-1, \tag{10c}$$

[6]This was exactly done for the neural network models of the other Dreams4Cars vehicle, see open data examples in [57].

[7]Interestingly, single-track models were originally developed to study vehicle handling, for which they are perfectly adequate because intentional human control is typically limited below 0.7 Hz [58].

$$\|\mathbf{x}_k\| \leq \left\|\mathbf{x}_{k,\text{bound}}\right\|, \quad \forall k = 0 \ldots N, \quad \text{(10d)}$$
$$\|\mathbf{u}_k\| \leq \left\|\mathbf{u}_{k,\text{bound}}\right\|, \quad \forall k = 0 \ldots N-1, \quad \text{(10e)}$$

where $\mathbf{x} \in \mathbb{R}^{n_x}$ denotes the vector of the states of the system, $\mathbf{u} \in \mathbb{R}^{n_u}$ is the control input vector, $h \in \mathbb{R}^{n_y}$ and $h_N \in \mathbb{R}^{n_{yN}}$ are the reference functions (respectively for the running and terminal cost) whereas the weighting matrices are marked with $\mathbf{W}_k \in \mathbb{R}^{n_y \times n_y}$ and $\mathbf{W}_N \in \mathbb{R}^{n_{yN} \times n_{yN}}$, notice that $\mathbf{W}_k, \mathbf{W}_N \succeq 0$. Symbols $\tilde{\mathbf{y}}_k \in \mathbb{R}^{n_y}$ and $\tilde{\mathbf{y}}_N \in \mathbb{R}^{n_{y,N}}$ denote respectively the time-varying and end term references. States and controls are bounded by respectively $\left\|\mathbf{x}_{k,\text{bound}}\right\|$ and $\left\|\mathbf{u}_{k,\text{bound}}\right\|$.

The MPC realization for the path tracking problem was carried out with the ACADO [59] toolbox. Firstly, an optimal control problem is defined by augmenting the plant dynamics (9) with:

$$\dot{\delta}(t) = u_{\text{MPC}}(t), \quad (11)$$

that introduces the MPC manipulated variable $u_{\text{MPC}}(t)$. With (11), the optimal control formulation thus includes a yaw-rate tracking cost, with additional conditions set for the control input $u_{\text{MPC}}(t) \in [-1.5\,\pi,\ 1.5\,\pi]$ and state $\delta(t) \in [-3.0\,\pi,\ 3.0\,\pi]$. The system augmentation in (11) ensures the continuity of the steering angle $\delta(t)$ and increases the smoothness of the MPC guidance.

The ACADO toolbox was then used for code, generating an optimized ANSI-C code [60] to be later implemented online for the path tracking task.

The assignment of the hyperparameters in Table 5 and weighting matrices $\mathbf{W}_k$ and $\mathbf{W}_N$ was carried out with a comprehensive trial and error tuning stage (the workflow is graphically represented in Fig. 1) to achieve the desired closed-loop performances in the simulation environment. Eventually a diagonal formulation for both $\mathbf{W}_k$ and $\mathbf{W}_N$ was implemented as in (12), where the first three diagonal entries weight the states according to the order proposed in (9), (11) whereas the last entry penalizes the control action (no terminal weight is assigned to the control action).

$$\mathbf{W}_k = \text{diag}\left([10^{-4},\ 1,\ 10^{-5},\ 5 \times 10^{-6}]\right)$$
$$\mathbf{W}_N = \text{diag}\left([10^{-6},\ 1,\ 10^{-6}]\right). \quad (12)$$

TABLE 5. MPC hyperparameters.

Prediction horizon	Integrator	Discretization	frequency
2.5 s	RK45	Multiple shooting	20 Hz

The total number of parameters of the MPC controller is hence 12: five are the vehicle parameters in (9), four the (diagonal) entries of $\mathbf{W}_k$ and three the entries of $\mathbf{W}_N$.

Table 4, lists the metrics of the MPC controller evaluated on the validation set.

V. NEURAL NETWORK MODEL MAINTENANCE

This section deals with extending the operation domain and the accuracy of the neural network forward and inverse models. This is the outer loop of Fig. 1, right.

Enhancing predictive and control models typically require collecting additional training data. From a biological perspective, this maintenance loop corresponds to alternating ''wake'' and ''sleep'' cycles. Similarly, in an artificial system, the controller developed at one round is used to drive the vehicle and collect new training data for the next round.

Besides collecting more training data, at each round, the system can also carefully test maneuvers at the borderline of the current operation domain (testing at the borders of the known operational domain is a form of exploratory ''motor babbling'').

In the following, however, we omit to augment the training set: we demonstrate a round of model enhancement using the same training/validation set (Table 1) to better appreciate the improvements in the model itself, without confusing the effect of additional training data (of course, in reality, model and data-set augmentation proceed in parallel).

A. NETWORK AUGMENTATION

Enhancing models needs network augmentation.

One obvious form of augmentation is increasing the number of local models. For example, Table 2 lists two forward models, with respectively $M = 1$ and $M = 3$ local models. With more training data, smaller receptive fields over a wider range of velocity could be considered. One might use a suitable information criterion (e.g., AIC) for selecting the most appropriate number of local models. The receptive fields should be adapted to contain a similar number of training/validation examples so that the individual local models have similar AIC. Algorithms for automatizing the local model refinement are described in [38], [44].

Alternatively, the structure of the forward and inverse model networks may be altered following a number of principles that are suggested by the biological solution (Fig. 2): *a*) learning context-related cross-inhibitions, *b*) learning nonlinearities incrementally and *c*) learning secondary inputs.

1) LEARNING CONTEXT-RELATED CROSS-INHIBITIONS

The idea behind this approach comes from the operation of the Golgi cells in the cerebellar network (Fig. 2). The distilled principle is as follows: Golgi cells receive input from parallel fibers that act as context detector (the parallel fibers originate from Granular cells that detect particular combinations of inputs, classifying one particular context). In turn, the Golgi cell inhibits other parallel fibers that represent local models that must be disabled in the context. This mechanism allows learning sub-models for different contexts in parallel and passes only those sub-models that are valid in the current context.

A demonstration of this principle was given in [37, Section II.D], where models for the engine-drive

line-propulsive force were learned for individual gears, with only the current gear released as a contribution to the output.

Since this aspect was well described in the cited paper, we focus on the remaining two proposed approaches (albeit we felt essential to point it).

2) LEARNING NONLINEARITIES INCREMENTALLY

The ideas behind this approach are: *a*) using nonlinear basis functions, inspired by the scheme of Fig. 2 and *b*) to augment the existing linear network with the learning of nonlinear terms incrementally.

Fig. 8 shows two possible implementations. Method A looks for a simple neural network mapping $\delta \rightarrow f(\delta)$ where $f(\cdot)$ is a suitable unary function with learnable parameters. This approach seeks a nonlinear transformation $f(\delta)$ which acts uniformly on all the past inputs δ_k. Hence it might learn static non linearities such as, e.g, the nonlinear kinematics of the steering linkage or the steady-state part of tire cornering nonlinearities. The alternative, B, seeks a more general nonlinear function $\bar{\boldsymbol{F}}(\delta_{k-n}, \ldots, \delta_k)$ for learning nonlinear corrections to the local models.

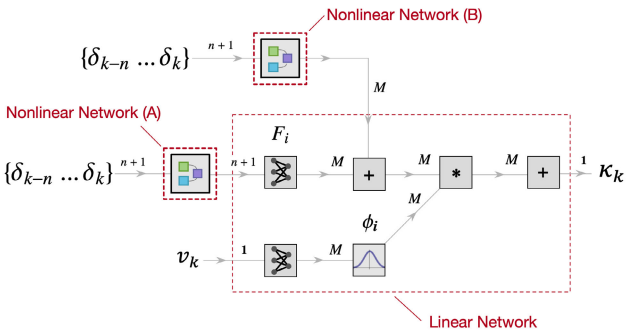


FIGURE 8. Augmentation of the forward model neural network: A) seeks to find static nonlinear mapping on the steering input; B) seeks a more general dynamic nonlinearity.

The analysis of the model residuals constitutes an important tool to guide the network extensions. Fig. 9 shows the forward model residuals on the training and validation sets. Since they cover different regimes, residuals are not i.i.d. (independent and identically distributed), which would be ideal. In particular, on straights and large bends, like in the motorway cases (1 and 7), the residuals are minimal. Larger residuals are found in the safety center test track (cases 2, 4, and 6), with superimposed motor noise (case 3), in the square test track (5), in particular at the sharpest curves, and in the lane change (8).

Inspection of the residual charts reveals steady-state residuals: the bias in the straights of track 5 (which is also present, albeit less evident, in the safety center track and almost disappears only at high speed in the motorways). These velocity-dependent residuals are due to tiny asymmetries in the vehicle. Learning these is almost immediate with the neural network. Explaining the asymmetries in the single-track model would, however, be not so simple (one may, of course, resort to a data-driven approach for the left/right asymmetry also in the analytical model).

Periodograms of the residuals give further information concerning the spectral content of the modeling errors. So, Fig. 10 shows that a large fraction of the residual power is at low frequency. This means that simple model augmentation like A in Fig. 8, targeting steady-state errors, are expected to produce appreciable improvements.

a: METHOD A

We elaborate on three possible variants of approach A, where the function $f(\delta)$ respectively tries to learn biases, quadratic, and cubic nonlinearities.

Learning biases. For learning the biases, we do not even need to augment the network with module A. In fact, it is sufficient to activate the biases $\boldsymbol{b}_i$ in (4) or in the layer F_i in Fig. 3. They were inactive in the network models of Table 2 because it was (incorrectly) assumed that vehicles had to be symmetric. Note that a different bias is learned for every local model (hence a velocity-dependent bias is learned).

Learning quadratic and cubic nonlinearities. For learning quadratic and cubic nonlinearities a module A implementing respectively $(f(\delta) = \delta + a\delta^2)$ and $(f(\delta) = \delta + a\delta^3)$ where a is a learnable parameters is used. Quadratic nonlinearities would not be expected in symmetric vehicles, which should instead show cubic terms as the first nonlinear term. Note that in this case, $f(\delta)$ is applied before channels. So, it may work for learning a velocity independent nonlinear effect, such as, e.g., nonlinear effects of the steering linkage (module should be threaded on the channels for learning a velocity-dependent quadratic/cubic nonlinearity).

Table 6 lists the main performance metrics of the three above (respectively ID 2.1, 2.2., 2.3). One can notice that modeling the biases gives an appreciable improvement in the root mean square error and the AIC. The addition of cubic and quadratic nonlinearity (pre-channels) does not improve significantly further, in any case.

TABLE 6. Improved models performance metrics.

	Validation set			
ID	Model	Param.	AIC	RMSE (km^{-1})
1	Network (M=3) (round 1)	91	-17562.5	0.0615
2.1	Network (M=3) (A: with bias)	94	-20962.3	0.0473
2.2	Network (M=3) (A: bias+cubic)	95	-20960.4	0.0473
2.3	Network (M=3) (A: bias+quadratic)	95	-21061.7	0.0469
2.4	Network (M=3) (B: S=2)	124	-23411.9	0.0390
2.5	Network (M=3) (B: S=3)	148	-23201.8	0.0394
2.6	Network (M=3) (B: bias+linear acc.)	95	-24707.1	0.0354
2.7	Network (M=3) (B: bias+piecewise acc.)	98	-26150.4	0.0317

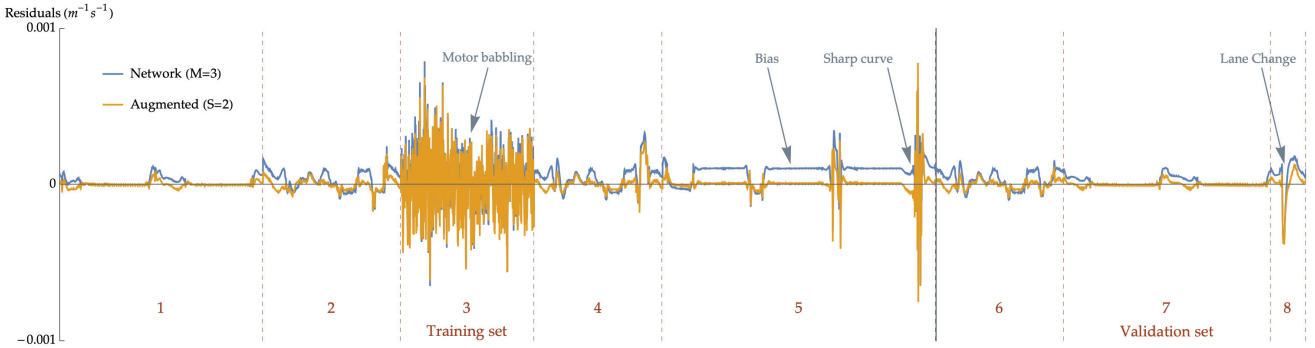


FIGURE 9. Residuals of the neural network models.

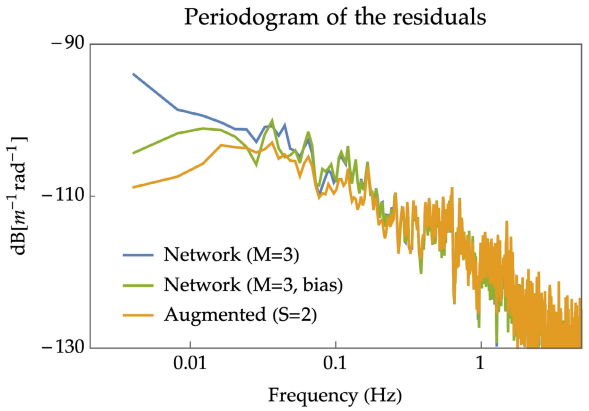


FIGURE 10. Periodogram of model residuals.

b: METHOD B

Method B seeks general nonlinear functions $\bar{F}_i(\delta_{k-n}, \ldots, \delta_k)$ to correct the output of the linear local models (Fig. 8).

A good choice for the structure of the functions —and hence of the network B— moves from the consideration that the nonlinear effects to be modeled are produced by a mechanical dynamic system. Hence, a recursive neural network such as, e.g., a naive basic recurrent layer might work (which is adequate to model the short-term memory of the systems (Fig. 4)):

$$\boldsymbol{x}_{i,t} = \tanh(\boldsymbol{A}_i \boldsymbol{x}_{i,t-1} + \boldsymbol{B}_i \delta_t + \boldsymbol{b}_i). \tag{13}$$

This network "accumulates" the effects of subsequent inputs δ_t across the temporal dimension, with $t \in \{k-n, \ldots, k\}$. The initial state x_{k-n-1} is set to zero, in order to truncate the memory of the system and let $\bar{F}_i$ be a function of only the last $\{\delta_{k-n}, \ldots, \delta_k\}$. The choice of (13) is a form of regularization of the more general $\bar{F}_i(\delta_{k-n}, \ldots, \delta_k)$, which imposes invariance of the effects of subsequent inputs δ_t across the temporal dimension via shared weights $(\boldsymbol{A}_i, \boldsymbol{B}_i, \boldsymbol{b}_i)$. The subscripts i indicate that there are distinct $\bar{F}_i(\delta_{k-n}, \ldots, \delta_k)$ for each local model $(i \in \{1, \ldots, M\})$, which is also evident from Fig. 8. The complexity of the dynamic model (13), and hence its explanatory capability, can be varied through the choice of the dimension, S of the (nonlinear) states $\boldsymbol{x}_i$. The output $y_i = \bar{F}_i(\delta_{k-n}, \ldots, \delta_k)$ can be obtained from a linear combination of the final states $\boldsymbol{x}_{i,k}$, i.e.:

$$y_{i,k} = \boldsymbol{C}_i \boldsymbol{x}_{i,k} + d_i. \tag{14}$$

Eq. 13 and (14) combined together form $\bar{F}_i(\delta_{k-n}, \ldots, \delta_k)$ under the assumption that the system is time invariant, which in turn limits the number of learnable parameters to $S^2 + 3S + 1$ for each channel, i.e.: $S^2 + S + S$ for (13) and $S + 1$ for (14).

The implementation of (13) and (14) —Fig. 8 block B— is obtained by means of a basic recurrent layer followed by a linear layer acting on the last output of the recurrent network (combined together they from block B).

An important point that should be stressed is that sub-network B is meant to model only the nonlinearities of the plant. This means that the training of the network (Fig. 8) is carried out in two phases: first, the linear network is trained (which was done in round 1) without the nonlinear component B. This learns the underlying average linear model. Then, the linear network weights are blocked, and the supplementary network B is trained, thus learning the nonlinear part.

It should be observed that, due to the tanh activation function in (13) the states $\boldsymbol{x}_{i,k}$ are bounded within the interval $]-1, 1[$. Hence the corrections y_i are bounded in the intervals $d_i \pm \|\boldsymbol{C}_i\|_1$. This permits to establish bounds for the magnitude of the nonlinear corrections, which compensate for the fact that the nonlinear module B is of more difficult interpretation than the linear part (at least one knows how large the deviations from the linear part might be).

Table 6 lists two networks obtained from the baseline model ($M = 3$, round 1) with, correspondingly, states of dimension $S = 2$ and $S = 3$. They have, respectively, 124 and 148 parameters and perform quite similarly in terms of RMSE (root mean square error). Network with $S = 3$ should work better, but in practice, the training may be stuck into local minima since they are nonlinear networks. The solutions indicated in the table were the best found with four different random initializations for $S = 2$ and 15 for $S = 3$.

It is expected that with more extensive search, the latter might perform somehow better.

Overall, the improvements, compared to the static correction of method A, are appreciable. Fig. 9 compares the residuals of the network $S = 2$ (ID 2.4) to the residuals of the baseline network (round 1, ID 1). Biases are removed, and corrections also occur in many curves. Fig. 10 compares the periodograms of the baseline network (ID 1), of the network with biases (method A, ID 2.1), and the network with $S = 2$ (method B, ID 2.4). Both achieve a reduction of residuals at low frequency, but the latter performs better and on a wider frequency range.

3) LEARNING SECONDARY INPUTS

A further look at Fig. 9 shows that, even after the above refinements, there still are errors with clear patterns, especially near sharp curves and the lane change. Also, Fig. 10 shows that, while low-frequency errors were reduced significantly, the $0.02 - 0.1$ Hz band was little attenuated.

The explanation is that the plant dynamics may be sensitive to interference inputs, as mentioned in section III-A1.c. Hence, one way to improve models is to learn the effects of observable (and significant) disturbances.

This section extends the model by learning the effects of the longitudinal dynamics. The coupling of longitudinal and lateral dynamics originates in the motive and brake forces that modify the lateral slip in cornering conditions.

Fortunately, we do not need to provide a detailed mechanism of how this effect develops (which would be a significant difficulty in an analytical model). In the remainder, we are going to learn two models of the effect of the longitudinal dynamics by: *a*) learning a linear correction proportional to the longitudinal acceleration, and *b*) learning a piecewise linear correction that makes a distinction between acceleration and deceleration phases.

a: ACCELERATION EFFECT (LINEAR MODEL)

In this simple model the effect of the longitudinal forces is summarised with:

$$\Delta\kappa_k = \alpha_1 a_k, \tag{15}$$

where $\Delta\kappa_k$ is the change in trajectory curvature produced by the longitudinal forces that cause the observable acceleration a_k and α_1 is a learnable parameter. In practice, (15) expresses the deviation of the trajectory caused by longitudinal actions (it should be zero in a symmetric vehicle). (15) is implemented with an additional branch in the model (just like Fig. 8, B) that contributes to κ_k. Table 6 shows that the simple correction for the longitudinal acceleration on top of the bias correction is worth more than the corrections for nonlinear steering wheel angle effects (ID 2.6).

b: ACCELERATION EFFECT (PIECEWISE LINEAR MODEL)

Since brake and engine forces act on different axles, a refined acceleration sub-model might consider splitting the effects between acceleration phases (engine acting and the front wheels) and deceleration phases (brake acting on both axles). Also, one might expect that the effect is a function of the steering wheel angle too. Hence, a piecewise nonlinear model could be tested in place of the above, made as follows:

$$\Delta\kappa_k = \begin{cases} \alpha_1\, a_k + \alpha_2\, a_k \delta_k, & \text{if } a_k \geq 0 \\ \beta_1\, a_k + \beta_2\, a_k \delta_k, & \text{if } a_k < 0, \end{cases} \tag{16}$$

with α_1, α_2, β_1 and β_2 learnable. As shown in Table 6 this is indeed the best model so far (ID 2.7).

B. UPDATING THE INVERSE MODEL

Among the previously studied models, Network ID 2.4 is selected. In terms of RMSE, the difference between Network ID 2.7 is not large. The AIC would vote for the latter, but the AIC only accounts for the complexity of the number of parameters. We evaluate the implementation cost of an additional signal and opt for the slightly less performing model, which is perfectly adequate to illustrate the second training round.

So, given an updated forward model, the synthesis of the inverse model is repeated according to the same steps illustrated in Section III-B.

However, in parallel to improvements in the forward model, one might also wish to improve the inverse network. Thus, the original inverse network of Fig. 6 is upgraded: the "target trajectory" branch is now formed of 3 channels (instead of the original one) and including biases. This means that instead of ϕ_0, Fig. 6 there are now three receptive fields, ϕ_i, similar to (6) but with term $1+Av^2$ at the numerator. Hence there are 31 learnable parameters per channel (30 weights and one bias). There also are 29 learnable parameters for the initial condition branch for a total of 120 parameters (in round 1, the parameters were 31+29). The inverse network is still linear, where the forward network is nonlinear. Thus, we seek a best-fitting linear inverse model (it may, indeed, be convenient to have a linear approximate inverse network). The now three channels of the inverse network realizes a LTV inverse model (with velocity modulating both the impulse response and the gain).

1) EVALUATION

a: QUALITY OF INVERSION

Table 7 compares the inverse network developed in the current round (round 2) to the formerly one (Table 3). Although the forward model is now expressing a richer dynamic, the inverse model cancellation is almost twice as good.

TABLE 7. Quality of the inverse models.

Network	Params.	Validation episodes RMSE (km^{-1})	Validation set RMSE (km^{-1})
Round 1	60	0.307	0.064
Round 2	122	0.172	0.033

An interesting aspect is revealed in Fig. 11, which shows the frequency response of respectively: the forward model,

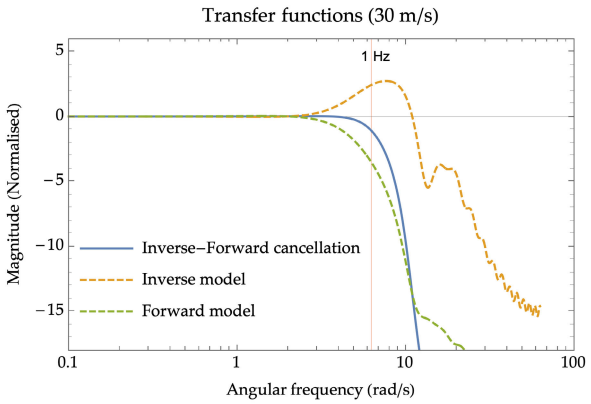


FIGURE 11. The inverse model compensates the forward model attenuation (as much as permitted by the validation set).

the inverse model, and the cascade of the two. For clarity, the transfer functions are normalized by the respective static gains. The chart shows that the inverse model has learned to compensate for the drop of the forward model response for the pole near 1 Hz with an increased magnitude of the inverse model network (the compensation process does not over-fit, being limited by the validation set).

b: INVERSE MODEL ACCURACY

Table 8 compares the performance metrics of the current inverse model to the former one. The root mean square error on the steering wheel angle is almost twice as good, and the fraction of variance left unexplained is almost three times better.

TABLE 8. Trained inverse model performance metrics.

Network	Param.	FVU	RMSE (rad)
Round 1	60	0.11%	0.005
Round 2	122	0.04%	0.003

C. STABILITY ANALYSIS

The training of the inverse model —according to the scheme given in Section III-B and Fig. 5— works by matching the input and the output of the inverse-forward cascade onto a set of given episodes $\kappa_i(t)$. Intuitively, the training process will likely end with a stable controller (the inverse model) because, at least for the many given episodes, the output $\bar{\kappa}_i(t)$ is very close to the specified trajectory $\kappa_i(t)$.

One might wonder whether it is possible to prove that the inverse model is stable for *any* possible input $\kappa(t)$, i.e., including functions not used during training such as, e.g., trajectories spanning longer time-horizons. The question makes sense because the inverse model network, as shown in Fig. 6, has the structure of an ARX system, which may indeed be unstable.

Disregarding the biases, and for constant velocity v, the output of the inverse network (Fig. 6) reads as $\delta_k = a_1\delta_{k-1} + \ldots + a_n\delta_{k-n} + b_0\kappa_k + \ldots + b_n\kappa_n$, where the coefficients a_i are the weights of the "initial condition" branch and the coefficients b_i the weights of the "target trajectory" branch. The latter are obtained from the three channels[8] weighted according to the activation functions $\phi_i(v)$. In terms of Z transform, the output reads as $\delta = (a_1\, z^{-1} + \ldots + a_n z^{-n})\delta + (b_0 + \ldots + b_n z^n)\kappa$ or, more concisely $\delta = A(z)\delta + B(z)\kappa$, which corresponds to the the ARX block diagram shown in Fig. 12.

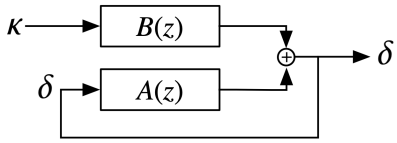


FIGURE 12. The neural network inverse model is equivalent to the following block diagram, which is an ARX system.

Hence, the transfer function of the inverse model[9] is $H(z) = B(z)/(1 - A(z))$. Note that the coefficients of $B(z)$ are parametric in v because they derive from the interpolation of three local velocity-dependent models, but the coefficients a_i are constant because the auto-regressive (AR) branch is unique.

The inverse model stability can be verified from the inspection of the poles of the AR branch $1/(1 - A(z))$, which are plotted in Fig. 13. They fall in a circular annuls with maximum magnitude equal to 0.85.

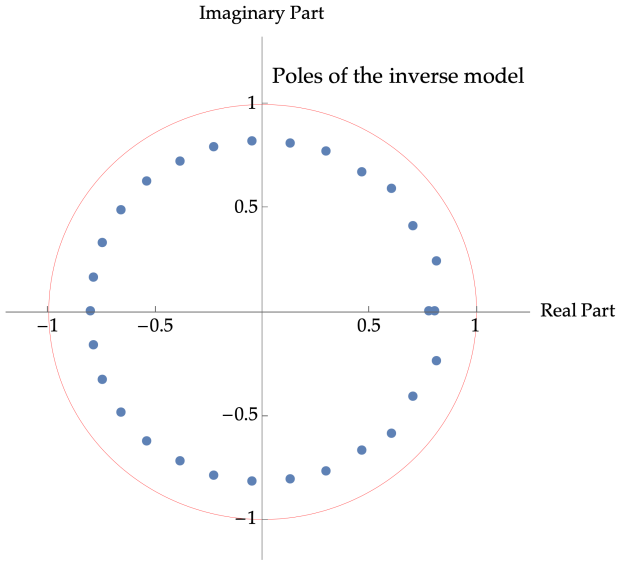


FIGURE 13. The poles of the inverse model network fall within the unit circle.

[8]Fig. 6, which is related to round 1, shows one channel but, in round 2, the target trajectory branch was upgraded to three channels.

[9]The magnitude of $H(z)$ is plotted in Fig. 11.

The inverse model is hence a stable system. The cascade with the forward model is also stable because the forward model is stable, being a FIR system. When the forward model is replaced by a real (stable) vehicle, the cascade remains stable.

Finally, note that this section demonstrates that the synthesized inverse model in itself is stable when tracking a given trajectory. However, autonomous driving applications need to update the planned trajectory in response to environmental changes such as the movement of other vehicles [1]–[3]. The planned trajectory may also be updated when accumulated drifts exceed a given threshold. A study concerning the stability of the re-planning loop, for the agent architecture described in [61], is given in [56] (see also next section).

D. EVALUATION IN CLOSED LOOP

As the last step of the demonstration, the three controllers (MPC, inverse network from round 1 and inverse network from round 2) are tested in a challenging obstacle-avoidance maneuver. For this, they are used, one after the other, in the motor control module of an agent for autonomous driving, responsible for trajectory planning.

The agent is described in [61] but, in order to understand the following section, some clarifications are given concerning trajectory planning and its interaction with trajectory execution. The agent elaborates action plans cyclically, in the form of two functions of time: the longitudinal control $j(t)$ and the lateral control $r(t)$. They represent the rate of change of longitudinal acceleration and curvature, respectively (longitudinal jerk and lateral jerk scaled by square velocity).

As for the lateral control, this means that the planned curvature is obtained by integration $\kappa(t) = \kappa_0 + \int_0^t r(\tau)\, d\tau$, where κ_0 is the current curvature, which is considered a state ($\kappa(t)$ is used as input to the inverse models). Further integrations yield orientation (of the velocity vector) and lateral position in the lane, and they also begin with the current orientation and lateral position. In other words, the planned trajectories begin at the current vehicle position, orientation, and curvature of the trajectory (equivalent to imposing continuity in the lateral acceleration).

To respond to context changes as quickly as possible, the agent produces new plans at every cycle (every 50 ms). This means that the agent also reacts to errors in the execution of the trajectory that might have happened in one cycle (initial position/orientation/curvature are updated). All deviations happening in one cycle are managed by the agent with the generation of a new motor plan that pursues the agent long-term goals from the deviated position, resulting in minimization of interventions (the "minimum intervention principle" [62], [61, Section II.D.3]).

At every cycle the agent select a trajectory among 1681 possible actions that correspond to lumping trajectories $\{j(t), r(t)\}$ into bins that share the same initial control $j(0) = j_0,\ r(0) = r_0$. There are 41×41 bins, which have a neuralised tensor organisation [61, Section II.B]. The choice of one particular trajectory among the 41×41 candidates is based on their "salience" (a merit function). There are two algorithms for action-selection: winner-takes-all (the trajectory with the momentarily instantaneous highest salience is chosen) and the multi-hypothesis sequential probability ratio test. The latter accumulates evidence of the most likely best choice on short observation times, which is more robust against motor and sensor noise [61, Section II.D]. In the following, the winner-takes-all algorithm is used because it does not hide/reduce the motor noise caused by inverse model errors.

Finally, the stability of the agent motor control with continuously updated trajectories under imperfect inverse models (imperfect cancellation of the forward dynamics and effect of execution errors in the following plans) is studied in [56]. It shows that the agent can, in principle, correct significant mismatches between the real forward dynamics and the inverse models, albeit at some cost.

1) EMERGENCY LANE CHANGE MANOEUVRE

The simulation scenario is an emergency lane change maneuver where the ego vehicle, initially traveling at 100 km/h, detects a stopped vehicle late, when it is only 50 m ahead. At the moment of detection, the time to collision is ≈ 1.6 s. In terms of maneuver severity, lateral acceleration, promptness, and precision of response, the situation goes beyond those found in the training and validation sets. It is a great benchmark to assess the quality (and generalization) of control obtained with the three models.

Fig. 14 shows the trajectories produced by the three controllers (black line) and the trajectories planned by the agent at selected time steps (thin colored lines). The vertical dashed

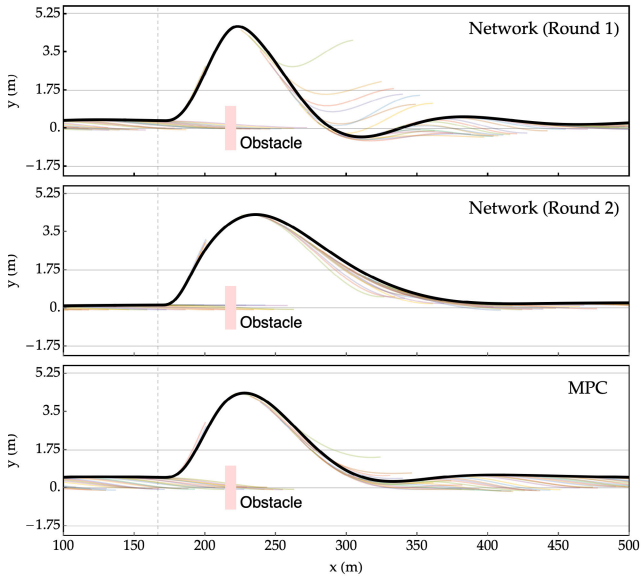


FIGURE 14. Trajectories for the three inverse models. The black line is the executed trajectory. The thin coloured lines are planned trajectories at selected time-steps. The dashed vertical line is the moment of detection of the obstacle.

TABLE 9. Trajectory metrics.

Inverse model	Lane change	
	Clearance to left lane margin (m)	Clearance to the obstacle (m)
Network (Round 1)	-0.329	2.816
Network (Round 2)	0.051	1.879
MPC	-0.009	2.193

line is the moment of detection of the obstacle. Table 9 lists trajectory related quality metrics.

Before detecting the obstacle, with Network 1 (round 1) and MPC controllers, the vehicle is off lane center: planned trajectories point towards the center, but the vehicle asymmetry (not included in the inverse models) keeps it off. After the detection, in a first phase, the agent moves to the left lane quickly, passing at the minimum distance to the obstacle and the lane listed in Table 9. Notice how both the round 1 network and MPC controller lead the vehicle slightly out of the lane left margin, whereas the updated network keeps sufficient clearance to both the obstacle and the lane's margin. The return in the lane would be excessively quick with Network 1, and the agent compensates with the planning of maneuvers that try to keep it back. For Network 2, the return is smooth. The trajectory slides compared to the plans because the agent is operating with a receding horizon scheme and, hence, continuously postpones the re-entry end-point. The MPC is in the middle of the two situations. When the car finally returns in the original lane, some oscillations for Network 1 and MPC are caused by the incorrect cancellation of the forward dynamics with the mechanics elucidated in [56].

A perfect inverse model should return an executed trajectory that completely overlaps the planned one (unless the agent modifies plans for external reasons, like in the commented re-entry phase). The updated network better quality can be visualized by comparing planned and executed trajectories in Fig. 14 (with the said caveat about sliding horizon in the re-entry phase). It is also worth noticing the interaction between the inverse model and the high-level planner [56]. The first network causes the planner to often change its motor plans due to the inaccuracies at the execution level. Instead, the better network gives birth to a more stable interaction between the two.

Fig. 15 compares the lateral accelerations. The relevant metrics are listed in Table 10. The peak acceleration for the lane change —to steer clear of the obstacle— are similar (the slightly smaller leftward acceleration required by MPC and Network round 1 is because they start already left off-center). Notably, the re-entry uses less acceleration for Network 2. The root mean square of the lateral acceleration is also shown in the table.

Finally, Fig. 16 compares the instantaneous planned curvature rate $r_0 = r(0)$ with the actually executed one $\dot{\kappa}(0)$. The difference between the two, integrated over one cycle, is the error $\Delta\kappa_k$ that occurs in the trajectory (curvature) execution at

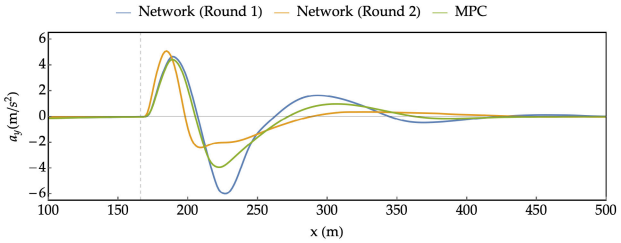


FIGURE 15. Lateral acceleration comparison.

TABLE 10. Acceleration and jerk metrics.

Inverse model	Lane change		
	$\lvert a_y\rvert$ max (m/s^2)	a_y RMS (m/s^2)	$\lvert r_0 - \dot{k}(0)\rvert$ RMSE (m^{-1}s^{-1})
Network (Round 1)	6.020	1.472	0.0059
Network (Round 2)	5.199	0.996	0.0038
MPC	4.478	1.142	0.0052

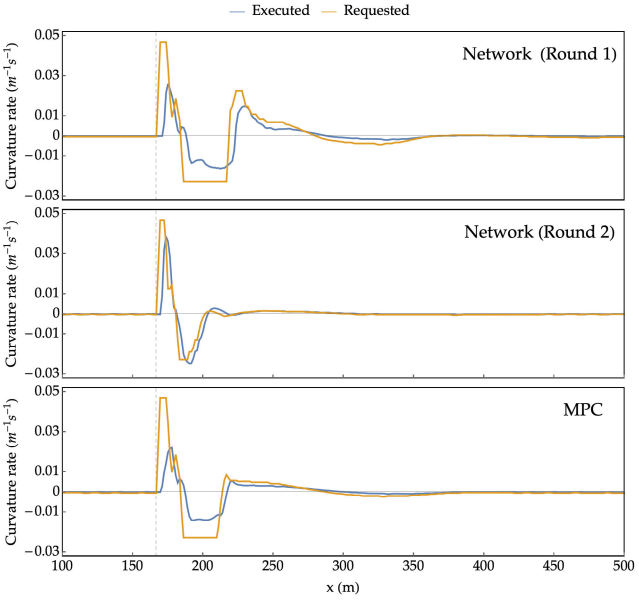


FIGURE 16. Steer rate comparison.

cycle k. The difference may also be read as the instantaneous mismatch between the inverse model and the actual forward dynamics. So, when the obstacle is detected the instantaneous requested lateral control $r(0)$ saturates at about 0.05 m^{-1}s^{-1}, which roughly corresponds to steering rate $\dot{\delta}(0) = 4$ rad/s. Network (round 2) responds with the smallest delay and producing almost the entire request. After then, the executed steering rate and the requested one match very closely. The MPC and the Network (round 1) execute the requests with greater errors and delay and a longer tail.

VI. DISCUSSION

Developing predictive control for machines is an art founded on the expertise of specialized engineers who —using human

intelligence— construct mathematical models of the plants. A variety of methods are then available for the synthesis of control, which efficiently exploit the knowledge of the forward dynamics model (think, for example, the Pontryagin principle for Optimal Control, or the model manipulation cited in [25]). This approach has a long story of success. It is also safe in the sense that the controller can be safely tested in simulations (as well as theoretical figures can be given for their limits). As long as the mathematical model matches the real plant, control can be used on the latter.

With the development of robotics, and complex systems such as autonomous vehicles, the need for autonomy and adaptability was also realized. For example, it is well known that robots might need to operate in situations that were not known at the design stage with sufficient detail. Automated vehicles in themselves fall in this condition: they operate correctly for thousands of miles, but they should operate so for billions of miles [63]. Furthermore, high-level behaviors of these systems are based on low-level motor control such that, if the latter is hard-coded, then behaviors almost necessarily need to be hard-coded too.

Within Machine Learning, Reinforcement Learning was developed to learn optimal action policy without needing the specification of the plant dynamics, i.e., by testing actions on the real system and reinforcing the choices leading to rewarding effects. Under given directives (for example, the definition of rewards), a robot could, in principle, construct optimal control policies with a greater degree of autonomy. There are, however, a few recognized caveats: the process may be very slow (or better sample inefficient) [23]. Furthermore, the process is not safe in the original incarnation: direct interaction with the real system exposes to risks. This is one of the reasons why living beings developed predictive abilities [26], [29]. Ha [24] developed a similar idea where an agent learns a predictive model for use within a reinforcement learning scheme. This is demonstrated in a game environment. The predictive model is made of a variational auto-encoder module, which compresses the bird view of the game playground into a lower-dimensional latent space and a recursive neural network predicting future states. A similar approach, within the first-person view in a more realistic simulated environment, is developed in [54], [55] as the first step towards visual dreaming in a road scenario. While appealing, developing this latter idea to the point it can be used on the real roads will still need research.

One of the reasons while Reinforcement learning is not very efficient lies in its nature of not asking —but not exploiting too– forward models beyond the safe sandbox purpose. However, the use of predictive models, once available, goes beyond that. A predictive model may be manipulated, opening possibilities for very efficient inversion with a logic similar to human engineers manipulation of mathematical models (be it the gradient descent case of this paper or the many syntheses approaches available for classic control theory). In other words, once the burden of learning a predictive model is undertaken, it is worth exploiting the model to the full extent, realizing the synthesis process like shown in this paper (Section III-B) and like usually exploited by intelligent human design.

The process of synthesis via mental simulation illustrated in this paper uses a gradient descent method to create the inverse model. It is not as efficient as some (closed form) mathematical approaches in control theory but, on the other hand, permits an agent some degree of autonomy in learning motor control. To maintain supervision over the whole process, human designers can give templates for the networks and other directives, still requiring physical insight and interpretations. However, one thing is understanding that, for example, longitudinal force influences tire side slips and lateral dynamics; another is developing the detailed mechanics of this for an analytical model.

Human beings learn by simulations in the sleep state but not all learning is via a reinforcement learning process [30]. This paper has proposed a different unsupervised learning approach that can be easily understood and controlled by human engineers. It allows a greater degree of autonomy and adaptability, which occurs in a safe sandbox. The process is the same for a simulated environment (like the CarMaker used here) or a real-world vehicle: it is only necessary to replace the training data source. In fact, in Dreams4Cars, two real vehicles were controlled in this way (see Appendix).

The notion of interpretability and explainability (for safety and standards reasons) was mentioned earlier but is also worth a final remark: neural network forward and inverse models are, in essence, linear NFIR/NARX models (with bounded nonlinear corrections). Also, as shown in round 2, it is possible to seek a linear best fitting NARX inverse model, which is fully interpretable within control theory.

VII. CONCLUSION, NEXT STEPS AND FUTURE WORK

This paper illustrated the learning of motor control via mental simulation. Together with [61] that deals with the enabling agent architecture, it gives the two first findings of Dreams4Cars. They constitute the foundation for further developments, part of which was carried in Dreams4Cars, and part may be subject to further research. There are essentially two lines: *a*) improving motor control and *b*) bootstrapping a sensorimotor architecture producing higher-level behaviors (see also project deliverable 'D3.3 Report on the simulation system (public version)').

Concerning point *a*, one aspect studied was the learning of stochastic forward models and the synthesis of robust control. Stochastic forward models can be created with neural networks that learn distributions for the output. There are many possibilities. The one that was developed in Dreams4Cars was based on using the bootstrapping method [64], which is a technique based on the re-sampling of the training data that produces many slightly different forward models that, together, realize an empirical distribution of the predicted quantity. A robust inverse model can then be obtained by minimizing the total loss of the distribution of the predictive models (in practice, the deterministic forward model in

Fig. 5 is replaced by the bootstrapped models and the loss corresponding to each prediction is accumulated). A research alternative might be modeling the output distribution with a predefined distribution type (for example, Gaussian) and learning the distribution parameters (for example, mean and variance).

Still concerning point *a*, another aspect that was in part studied and which might be worth further research, is adaptive control via forward-inverse models for different contexts. For example, one might learn the forward model for a fully loaded and empty car. In operation, the two predictive models can be run in parallel, and predictions can be compared to the actual dynamics of the car, estimating which model holds better (it could be an interpolation between the two).[10] Once the context is identified, the matching inverse model can be better used for control. However, the rapid update of the inverse-forward pair may lead to instabilities which have to be studied.

Concerning point *b*, with a stochastic forward scheme, Fig. 5 realises a (learned) probabilistic motion model. This can, in turn, be used for a higher level of episodic simulations: i.e., imagining short movements and the associated uncertainties, and then proceed with longer-term trajectories. In this way, it is finally possible to estimate the salience (value or reward) of different actions or action sequences. Episodic simulations similar to those introduced here, but of a higher level (short and longer-term goals) can thus be developed to learn associations between sensory data and the value of different behaviors as described in [61]. This process is better outlined in 'D3.3 Report on the simulation system (public version)' and will be described in future publications.

APPENDIX
REAL WORLD EXPERIMENTS

To illustrate the process of learning via simulation, this paper used *ad hoc* simulations in the CarMaker environment because it permitted to develop diverse examples tailored to the particular needs of this paper and independently from the original experiment program of the Dreams4Cars project.

Conversely, the real-world experiments of Dreams4Cars were focused on the needs of the project and are described in the public deliverables 'D5.2 Test methods and metrics (public version)' and 'D5.3 Evolved agent', available at [14] or [65].

Among the many experiments of Dreams4Cars, a group of activities, called ''simulation fidelity tests'', was arranged. They consisted of comparing the actual vehicle to a model in the loop version (MIL) on the following scenarios: 1) Lane following and speed adaptation, 2) Car following, 3) Overtaking a slow vehicle, and 4) Lane change with two stationary vehicles.

These situations were created on a real test track in mixed reality, i.e., with the real vehicle, the real test track but with emulated obstacles obtained by hallucinating the object detection module. The mixed reality was useful for reducing risks, but the primary motivation was to move the obstacles with precise predefined trajectories. The same situations were recreated in the CarMaker environment (MIL), with the obstacles moving in the same way and the same test track geometry. As for the model in the loop vehicle dynamics, the multi-body model built into CarMaker was instantiated with parameters values of the actual vehicle known to the manufacturer.

A significant difference between the Dreams4Cars real/MIL vehicles and the vehicle model used in this paper concerns the steering actuator. In Dreams4Cars, the lateral control of the Jeep Renegade was obtained by adapting the steering actuator of the lane-keeping assistance function. As such, the bandwidth was limited, the actuator presented an appreciable dead zone, and there was some hysteresis (these would not be present in actuators specifically designed for autonomous driving). Hence, a model of the steering actuator was included in the MIL environment. However, in this paper, the steering actuator is omitted, as if it were an ideal actuator. This omission is done for clarity because the actuator dynamics would have confused the dynamics of the vehicle and made more difficult the comparisons with the bicycle model and the MPC control in Section IV. The real and MIL vehicles of Dreams4Cars also presented more noise, in part because of the steering actuator motor noise and in part (only for the real vehicle) because of sensory noise.

Apart from this difference, the process used on the real and MIL vehicles in Dreams4Cars was the same illustrated here. We first learned the systems forward models, but including the actuator dynamics. We used the neural architectures used here, except that, in input, the steering wheel angle was replaced by the requested steering wheel angle (input to the steering actuator). In this way, the network learned the dynamics of the plant formed by the actuator-vehicle cascade. Then, the forward models of the real and MIL vehicles were compared, and a slight, but appreciable, difference was found. Distinct inverse models were, hence, trained for the MIL and real vehicles.[11]

In the following step, the agent described in [61] drove both the real and the MIL vehicles according to the control scheme shown in Section 5.2 of 'D5.2 Test methods and metrics (public version)' or Section 3.1 of 'D2.3 Report on the Runtime system (public version)' (see also [56]). Accordingly, trajectory planning was carried out by the same agent with the same logic, but trajectory execution (i.e., the conversion from trajectory to requested steering wheel angle) was tuned to each particular vehicle via the matching inverse model. Even if the trajectory planning logic was the same, the final trajectories were different because there were different execution errors followed by different corrections.

[10] Inline use of the predictive models can also serve for anomaly detection and for sensor data filtering.

[11] Where in typical control design, the controller is developed on a vehicle model and used on the real one; in this case, the inverse network parameters are adapted to each vehicle by training on the (slightly) different forward networks.

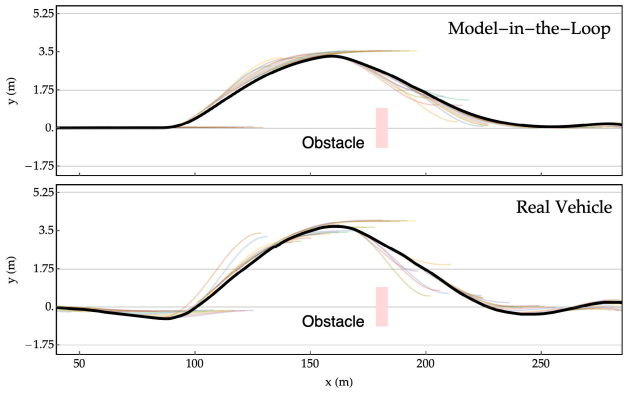


FIGURE 17. Trajectories for the real and simulated cases. The black line is the executed trajectory. The thin coloured lines are planned trajectories at selected time-steps.

TABLE 11. Trajectory metrics, real vs. simulation case.

Vehicle	Dreams4Cars Lane Change. Clearance to left lane margin (m)	Clearance to the obstacle (m)
Real	1.080	1.181
MIL	1.017	0.823

As an example, Fig. 17 compares the trajectories of the real and MIL vehicle in the lane change scenario. This situation is similar to the situation studied in this paper Section V-D1, except that in Dreams4Cars the speed was 60 km/h and the obstacle was detected at the distance of 100 m. So the maneuver is less extreme and can also be anticipated. Table 11 reports the same metrics used in Table 9. Since the evasive maneuver (before obstacle) is less severe, so the clearance with the left margin is wider in both cases. Conversely, the clearance with the obstacle is smaller because the re-entry maneuver is anticipated. If one compares the real and MIL vehicles, the former shows more undulated trajectories and more corrections. This is because of the actuator dead zone and hysteresis (not present in the MIL vehicle). The root mean square distance between the two trajectories is 0.294 m. The coefficient of correlation between the lateral position of the real vehicle and the MIL vehicle is $R^2 = 0.96$. So, even in more difficult conditions (noise due to the steering actuator and sensors of the real vehicle), the method illustrated in this paper worked on a real vehicle in a way comparable to the simulation environment.

REFERENCES

[1] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 4, pp. 1135–1145, Apr. 2016.
[2] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Trans. Intell. Vehicles*, vol. 1, no. 1, pp. 33–55, Mar. 2016.
[3] S. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. Eng, D. Rus, and M. Ang, "Perception, planning, control, and coordination for autonomous vehicles," *Machines*, vol. 5, no. 1, p. 6, Feb. 2017.
[4] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2013, *arXiv:1312.6199*. [Online]. Available: http://arxiv.org/abs/1312.6199
[5] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 427–436.
[6] W. Samek, T. Wiegand, and K.-R. Müller, "Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models," 2017, *arXiv:1708.08296*. [Online]. Available: http://arxiv.org/abs/1708.08296
[7] *Waymo Riders Describe Experiences on the Road*. Accessed: Dec. 6, 2019. [Online]. Available: https://www.theinformation.com/articles/waymo-riders-describe-experiences-on-the-road
[8] *Uber's Self-Driving Car Didn't Know Pedestrians Could Jaywalk*. Accessed: Dec. 6, 2019. [Online]. Available: https://www.wired.com/story/ubers-self-driving-car-didnt-know-pedestrians-could-jaywalk/
[9] NTSB Finds. (Sep. 2019). *En-US Tesla Car Was on Autopilot When it Hit a Culver City Firetruck*. [Online]. Available: https://www.latimes.com/business/story/2019-09-03/tesla-was-on-autopilot-when-it-hit-culver-city-fire-truck-ntsb-finds
[10] NTSB. Preliminary Report Released for Crash Involving Pedestrian. Uber Technologies, Inc., Test Vehicle, 2018. [Online]. Available: https://www.ntsb.gov/news/press-releases/Pages/NR20180524.aspx
[11] *Collision Between Vehicle Controlled by Developmental Automated Driving System and Pedestrian*. Accessed: Dec. 6, 2019. [Online]. Available: https://www.ntsb.gov/news/events/Pages/2019-HWY18MH010-BMG.aspx
[12] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 2016, *arXiv:1604.07316*. [Online]. Available: http://arxiv.org/abs/1604.07316
[13] *Learning to Drive: Beyond Pure Imitation—Waymo—Medium*. Accessed: Dec. 6, 2019. [Online]. Available: https://medium.com/waymo/learning-to-drive-beyond-pure-imitation-465499f8bcb2
[14] *Dreams4Cars—Dream-Like Simulation Abilities for Automated Cars*. Accessed: Feb. 6, 2020. [Online]. Available: https://www.dreams4cars.eu/en
[15] H. Svensson and S. Thill, "Beyond bodily anticipation: Internal simulations in social interaction," *Cognit. Syst. Res.*, vol. 40, pp. 161–171, Dec. 2016.
[16] S. Kuutti, R. Bowden, Y. Jin, P. Barber, and S. Fallah, "A survey of deep learning applications to autonomous vehicle control," Dec. 2019, *arXiv:1912.10773*. [Online]. Available: http://arxiv.org/abs/1912.10773
[17] P. de Haan, D. Jayaraman, and S. Levine, "Causal confusion in imitation learning," Nov. 2019, *arXiv:1905.11979*. [Online]. Available: http://arxiv.org/abs/1905.11979
[18] G. Ras, M. van Gerven, and P. Haselager, "Explanation methods in deep learning: Users, values, concerns and challenges," in *Explainable and Interpretable Models in Computer Vision and Machine Learning* (The Springer Series on Challenges in Machine Learning), H. Escalante *et al.*, Eds. Cham, Switzerland: Springer, 2018. [Online]. Available: http://doi-org-443.webvpn.fjmu.edu.cn/10.1007/978-3-319-98131-4_2
[19] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, "Explaining how a deep neural network trained with end-to-end learning steers a car," 2017, *arXiv:1704.07911*. [Online]. Available: http://arxiv.org/abs/1704.07911
[20] G. Pezzulo and P. Cisek, "Navigating the affordance landscape: Feedback control as a process model of behavior and cognition," *Trends Cognit. Sci.*, vol. 20, no. 6, pp. 414–424, Jun. 2016.
[21] G. Marti, A. H. P. Morice, and G. Montagne, "Drivers' decision-making when attempting to cross an intersection results from choice between affordances," *Frontiers Hum. Neurosci.*, vol. 8, Jan. 2015, Art. no. 1026.
[22] R. Bolado-Gomez and K. Gurney, "A biologically plausible embodied model of action discovery," *Frontiers Neurorobot.*, vol. 7, no. 4, Mar. 2013, Art. no. 4.
[23] M. Botvinick, S. Ritter, J. X. Wang, Z. Kurth-Nelson, C. Blundell, and D. Hassabis, "Reinforcement learning, fast and slow," *Trends Cognit. Sci.*, vol. 23, no. 5, pp. 408–422, May 2019.
[24] D. Ha and J. Schmidhuber, "World models," May 2018, *arXiv:1803.10122*. [Online]. Available: http://arxiv.org/abs/1803.10122
[25] Z. Yao, J. Yao, and W. Sun, "Adaptive RISE control of hydraulic systems with multilayer neural-networks," *IEEE Trans. Ind. Electron.*, vol. 66, no. 11, pp. 8638–8647, Nov. 2019.
[26] G. Hesslow, "The current status of the simulation theory of cognition," *Brain Res.*, vol. 1428, pp. 9–71, Jan. 2012.
[27] P. Cisek, "Cortical mechanisms of action selection: The affordance competition hypothesis," *Phil. Trans. Roy. Soc. B, Biol. Sci.*, vol. 362, no. 1485, pp. 1585–1599, Sep. 2007.

[28] D. Windridge, H. Svensson, and S. Thill, "On the utility of dreaming: A general model for how learning in artificial agents can benefit from data hallucination," *Adapt. Behav.*, 2020, Art. no. 1059712319896489. [Online]. Available: https://journals.sagepub.com/doi/full/10.1177/1059712319896489

[29] K. L. Downing, "Predictive models in the brain," *Connection Sci.*, vol. 21, no. 1, pp. 39–74, Mar. 2009.

[30] C. T. Smith, J. B. Aubrey, and K. R. Peters, "Different roles for REM and stage 2 sleep in motor learning: A proposed model," *Psychologica Belgica*, vol. 44, nos. 1–2, pp. 81–104, 2004, doi: 10.5334/pb.1018.

[31] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—A survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.

[32] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert, "Constrained model predictive control: Stability and optimality?" *Automatica*, vol. 36, no. 6, pp. 789–814, Jun. 2000.

[33] E. Kim, J. Kim, and M. Sunwoo, "Model predictive control strategy for smooth path tracking of autonomous vehicles with steering actuator dynamics," *Int. J. Automot. Technol.*, vol. 15, no. 7, pp. 1155–1164, Dec. 2014.

[34] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat, "Predictive active steering control for autonomous vehicle systems," *IEEE Trans. Control Syst. Technol.*, vol. 15, no. 3, pp. 566–580, May 2007.

[35] P. Falcone, F. Borrelli, H. E. Tseng, J. Asgari, and D. Hrovat, "Linear time-varying model predictive control and its application to active steering systems: Stability analysis and experimental validation," *Int. J. Robust Nonlinear Control*, vol. 18, no. 8, pp. 862–875, May 2008.

[36] M. Da Lio, D. Bortoluzzi, and G. P. R. Papini, "Modelling longitudinal vehicle dynamics with neural networks," *Vehicle Syst. Dyn.*, vol. 58, no. 11, pp. 1675–1693, Nov. 2020.

[37] S. S. James, S. R. Anderson, and M. D. Lio, "Longitudinal vehicle dynamics: A comparison of physical and data-driven models under large-scale real-world driving conditions," *IEEE Access*, vol. 8, pp. 73714–73729, 2020.

[38] S. Vijayakumar. (Jul. 2005). *LWPR: A Scalable Method for Incremental Online Learning in High Dimensions*. [Online]. Available: https://era.ed.ac.uk/handle/1842/3705

[39] O. Nelles, A. Fink, and R. Isermann, "Local linear model trees (LOLIMOT) toolbox for nonlinear system identification," *IFAC Proc. Volumes*, vol. 33, no. 15, pp. 845–850, Jun. 2000.

[40] 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.

[41] P. Forssén, G. Granlund, and J. Wiklund, "Channel representation of colour images," Linköping Univ., Linköping, Sweden, Tech. Rep. LiTH-ISY-R-2418, 2002.

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

[43] 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.

[44] T. Münker and O. Nelles, "Nonlinear system identification with regularized local FIR model networks," *Eng. Appl. Artif. Intell.*, vol. 67, pp. 345–354, Jan. 2018.

[45] J. Porrill, P. Dean, and S. R. Anderson, "Adaptive filters and internal models: Multilevel description of cerebellar function," *Neural Netw.*, vol. 47, pp. 134–149, Nov. 2013.

[46] M. Abe, *Vehicle Handling Dynamics: Theory and Application*. London, U.K.: Butterworth, 2015.

[47] Y. Demiris and A. Dearden. (2005). *From Motor Babbling to Hierarchical Learning by Imitation: A Robot Developmental Pathway*. pp. 31–37. [Online]. Available: http://cogprints.org/4961/

[48] *Explaining the Dreams4cars Autonomous Driving Capabilities*. Accessed: Jun. 17, 2020. [Online]. Available: https://www.youtube.com/watch?v=ODuw9TNanE8

[49] 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.

[50] 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.

[51] T. Chen, H. Ohlsson, and L. Ljung, "On the estimation of transfer functions, regularizations and Gaussian processes—Revisited," *Automatica*, vol. 48, no. 8, pp. 1525–1535, Aug. 2012.

[52] G. Pillonetto, F. Dinuzzo, T. Chen, G. De Nicolao, and L. Ljung, "Kernel methods in system identification, machine learning and function estimation: A survey," *Automatica*, vol. 50, no. 3, pp. 657–682, Mar. 2014.

[53] H. Akaike, "A new look at the statistical model identification," *IEEE Trans. Autom. Control*, vol. AC-19, no. 6, pp. 716–723, Dec. 1974.

[54] A. Plebe, R. Donà, G. P. P. Rosati, and M. Da Lio, "Mental imagery for intelligent vehicles," in *Proc. 5th Int. Conf. Vehicle Technol. Intell. Transp. Syst.*, Crete, Greece, 2019, pp. 43–51.

[55] A. Plebe and M. D. Lio, "On the road with 16 neurons: Towards interpretable and manipulable latent representations for visual predictions in driving scenarios," *IEEE Access*, vol. 8, pp. 179716–179734, 2020.

[56] R. Donà, G. P. R. Papini, M. Da Lio, and L. Zaccarian, "On the stability and robustness of hierarchical vehicle lateral control with inverse/forward dynamics quasi-cancellation," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 10559–10570, Sep. 2019.

[57] M. Da Lio, E. Berghöfer, and M. Yüksel. (Jan. 2020). *Dreams4Cars Experimental data from Autonomous Test Vehicle*. [Online]. Available: https://zenodo.org/record/3582953#.XwrdiJMzbUK

[58] F. Biral, D. Bortoluzzi, V. Cossalter, and M. Lio, "Experimental study of motorcycle transfer functions for evaluating handling," *Vehicle Syst. Dyn.*, vol. 39, no. 1, pp. 1–25, Jan. 2003.

[59] B. Houska, H. Ferreau, and M. Diehl, "ACADO toolkit—An open source framework for automatic control and dynamic optimization," *Optim. Control Appl. Methods*, vol. 32, no. 3, pp. 298–312, 2011.

[60] B. Houska, H. J. Ferreau, and M. Diehl, "An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range," *Automatica*, vol. 47, no. 10, pp. 2279–2285, 2011.

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

[62] E. Todorov and M. I. Jordan, "A minimal intervention principle for coordinated movement," in *Proc. Adv. Neural Inf. Process. Syst.*, 2003, pp. 27–34.

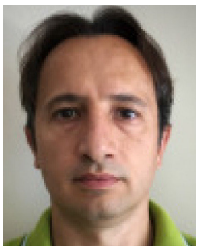
[63] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" *Transp. Res. A, Policy Pract.*, vol. 94, pp. 182–193, Dec. 2016.

[64] B. Efron and R. Tibshirani, *An Introduction to Bootstrap*. New York, NY, USA: Chapman & Hall, 1994.

[65] (Dreams4Cars H2020; CORDIS, European Commission). *Dream-Like Simulation Abilities for Automated Cars, Results*. Accessed: Oct. 1, 2020. [Online]. Available: https://cordis.europa.eu/project/id/731593/results

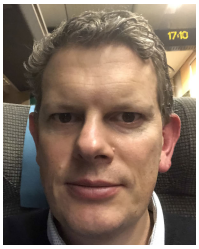
MAURO DA LIO (Member, IEEE) received the Laurea degree in mechanical engineering from the University of Padova, Italy, in 1986. He is currently a Full Professor of Mechanical Systems with the University of Trento, Italy. His earlier research activity was on modeling, simulation, and optimal control of mechanical multibody systems, in particular, vehicle and spacecraft dynamics. More recently, his focus shifted to the modeling 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 programme 6 and 7 projects (PReVENT, SAFERIDER, interactIVe, VERITAS, AdaptIVe, and No-Tremor). He is currently 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, working on the EU project Dreams4Cars. His main scientific interest is autonomous driving vehicles which constitutes the core topic of his Ph.D. research project.



FRANCESCO BIRAL was born in Italy, in 1972. He received the master's degree in mechanical engineering from the University of Padova, Italy, in 1997, and the Ph.D. degree in mechanism and machine theory from the University of Brescia, Italy, in 2000. He is currently an Associate Professor with the Department of Industrial Engineering, University of Trento. His research interests include symbolic and numerical multibody dynamics and optimization, constrained optimal control, mainly in the field of vehicle dynamics, with a special focus on intelligent transportation systems.

GASTONE PIETRO ROSATI PAPINI is currently a Researcher in advanced control systems applied to the field of autonomous driving with the Department of Industrial Engineering, University of Trento, and a Co-Founder of Cheros s.r.l., a University of Pisa spinout company that deals with renewable energies and ICT solutions. He was involved in several EU projects: VERITAS, Poly-Wec, and Dreams4Cars. He has over seven years' experience in applied mechanics, in particular, in the design of advanced control systems that also exploit machine learning techniques.



HENRIK SVENSSON received the Ph.D. degree from Linköping University, Sweden, in 2013. He is currently a Senior Lecturer with the University of Skövde, Sweden. He has a background in cognitive science. His research interest is focused on embodied theories of cognition, especially simulation/emulation theories of cognition. Another focus is on cognitive systems, in particular, the influence of bodily and situational factors in the interaction between artificial/natural agents, and how this may affect the design of autonomous agents.