Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Trajectory Planning for Autonomous Vehicles combining Nonlinear Optimal Control and Supervised Learning ⋆

Lukas Markolf * **Jan Eilbrecht** * **Olaf Stursberg** *

* *Control and System Theory, Department of Electrical Engineering and Computer Science, University of Kassel, Germany, (e-mail: {lukas.markolf,eilbrecht,stursberg}@uni-kassel.de).*

Abstract: This paper considers computationally efficient planning of reference trajectories for autonomous on-road vehicles in a cooperative setting. The basic element of the approach is the notion of so-called maneuvers, which allow to cast the nonlinear and non-convex planning task into a highly structured optimal control problem. This can be solved quite efficiently, but not fast enough for online operation when considering nonlinear vehicle models. Therefore, the approach proposed in this paper aims at approximating solutions using a supervised learning approach: First, training data are generated by solving optimal control problems and are then used to train a neural network. As is demonstrated for a cooperative overtaking maneuver, this approach shows good performance, while (contrasting approaches like reinforcement learning) requiring only low training effort.

Keywords: Nonlinear and optimal automotive control, automated driving, machine learning, intelligent control, neural networks

1. INTRODUCTION

An important problem in autonomous driving and similar applications (e.g. control of UAVs, rovers, or stationary robots) is to plan reference trajectories for the future states of a vehicle. This problem is difficult due to aspects such as nonlinear vehicle dynamics and collision avoidance constraints making the solution space non-convex. If cooperative settings are considered, i.e., scenarios in which multiple vehicles must adapt their plans to those of others, additional challenges as an increased dimensionality of the state space arise. Despite these challenges, it is required to determine reference trajectories very quickly during online operation, typically in around 10 ms.

Reflecting the practical importance of the problem, several solution approaches have been proposed within the last decade: While early work mostly focused on planning without dynamics (LaValle, 2006), more recent approaches address the problem within an optimal control setting: Methods to solve quite general problem formulations, e.g. in (Limebeer and Rao, 2015), can handle nonlinear dynamics subject to input and state constraints by using specialized numerical optimal control software but currently not quickly enough for online operation. Similarly complex problems can be addressed by sampling-based algorithms, e.g. RRT* in (hwan Jeon et al., 2013), but also there the computation times are relatively high, and only asymptotic guarantees exist with respect to completeness. Combinations of mixed-integer programming and model-predictive control (MPC) are limited to simpler problem instances relying on affine vehicle dynamics, but have found wide application both for aerial and on-road vehicles (Schouwenaars et al., 2001; Qian et al., 2016; Eilbrecht and Stursberg, 2018). These approaches provide guarantees for convergence, while being computationally efficient. Previous work of the authors is also based on this methodology, which was recently extended by a maneuver-based approach (Eilbrecht and Stursberg, 2018). Maneuvers, cf. Sec. 2.2, are modeled by hybrid automata, which allow to formulate highly structured optimization problems that can be solved efficiently. At the same time, it is possible to compute the set of initial states for which a solution to the planning problem exists, thus increasing the reliability of planning. A significant improvement in computational efficiency of both offline and online calculations was then obtained using approximate solutions in form of linear interpolation between optimal solutions (Eilbrecht and Stursberg, 2019).

In this paper, the previous work (Eilbrecht and Stursberg, 2019) is extended to obtain plans based on nonlinear instead of linear vehicle models, contributing to more realistic behavior. Despite the more complex problem class, computational efficiency is retained, relying on learning-based approximations of optimal solutions. Clearly, learning-based optimal control is no novelty per se. Typically, it is based on reinforcement learning which, however, may fail to converge to high-quality solutions if exploration and exploitation are not properly balanced, requiring much tuning effort. Another contribution of this paper on a conceptual level is to demonstrate that exploiting problem structure allows for efficient generation of training data, such that supervised learning can be used instead of reinforcement learning. Including knowledge on

⋆ Financial support by the German Research Foundation (DFG) within priority program (SPP) 1835 is gratefully acknowledged.

problem structure allows in addition to reduce the complexity of the approximation function.

The paper is structured as follows: while Sec. 2 formally introduces the problem and outlines the solution approach, Sec. 3 details the generation of training data by solving nonlinear optimal control problems. Using data to train the approximation structure is described in Sec. 4, and Sec. 5 demonstrates the application of the procedure in a cooperative overtaking maneuver.

2. OVERVIEW OF THE APPROACH

This section provides a general overview of the considered problem as well as of the proposed solution approach.

2.1 Problem setting

This work considers a setting in which a group of intelligent vehicles drives cooperatively on a road network. Let $\mathcal{G} \subset \mathbb{N}$ contain the identifiers of these vehicles and assume that each vehicle $i \in \mathcal{G}$ is given knowledge of the sequence of roads that lead it from its starting point to its destination. Let the dynamics of a vehicle i be given by:

$$\dot{\xi}^{(i)}(t) = f^{(i)}\left(\xi^{(i)}(t), \mu^{(i)}(t)\right),$$

depending on its state $\xi^{(i)}$ at the current time $t \in \mathbb{R}^{\geq 0}$, and on input values $\mu^{(i)}$. Assume that the inputs are determined by a given controller function $k^{(i)}$ based on the current state and a reference value $x^{(i)}_{\text{des}}(t)$:

$$\mu^{(i)}(t) = k^{(i)}\left(\xi^{(i)}(t), x^{(i)}_{\text{des}}(t)\right).$$

The task considered in this paper is to determine the reference trajectory $x^{(i)}_{\text{des}}(t)$ such that constraints for the states and inputs of the vehicle are not violated. While some of these, such as constraints on velocity or acceleration, stem from the dynamics of the vehicle, others result from the road topology or the need to avoid collisions with other vehicles. The latter implies that $x^{(i)}_{\text{des}}$ must depend on $x^{(j)}_{\text{des}}$, $j \in \mathcal{G}$, $j \neq i$. This problem is considered in a cooperative setting, meaning that vehicles can communicate information and adapt their behavior to that of others.

2.2 Solution Approach

Main elements of the general approach are *maneuvers*, used to encode sets of qualitatively similar trajectories such as merging, overtaking, lane keeping (Eilbrecht and Stursberg, 2018, 2019). They are modeled by hybrid automata, defined there as follows:

Definition 1. (Hybrid Automaton). A hybrid automaton is a tuple
$\texttt{HA} = (\mathcal{Q}, q_0, \mathbb{X}, \mathbb{U}, \text{inv}, \mathcal{X}_0, \mathcal{X}_\text{T}, \Theta, g, f)$ consisting of:

- a finite set of phases $\mathcal{Q}$ with initial phase $q_0 \in \mathcal{Q}$,
- a continuous state space $\mathbb{X} \subseteq \mathbb{R}^{n_x}$ with state vector $x \in \mathbb{X}$,
- a continuous input space $\mathbb{U} \subseteq \mathbb{R}^{n_u}$ with input vector $u \in \mathbb{U}$,
- a function $\text{inv} : \mathcal{Q} \to 2^{\mathbb{X}}$ called invariant and assigning to each phase $q \in \mathcal{Q}$ a set $\text{inv}(q) \subseteq \mathbb{X}$ in which the continuous state $x \in \mathbb{X}$ may evolve without changing the phase q,
- the sets $\mathcal{X}_0 \subseteq \text{inv}(q_0)$ and $\mathcal{X}_\text{T} \subseteq \mathbb{X}$ of initial and target states, respectively,
- a set of discrete transitions $\Theta \subseteq \mathcal{Q} \times \mathcal{Q}$,
- a guard function $g : \Theta \to 2^{\mathbb{X}}$ which assigns a guard set $g(\theta) \subseteq \text{inv}(q_i) \cap \text{inv}(q_j)$ to each transition $\theta = (q_i, q_j) \in \Theta$ with $\text{inv}(q_i) \cap \text{inv}(q_j) \neq \emptyset$,
- and a flow function $f : \mathcal{Q} \to (\mathbb{X} \times \mathbb{U} \to \mathbb{R}^{n_x})$.

When interpreting x and q as functions of real-valued time $t \in [t_0, t_f] \subset \mathbb{R}^{\geq 0}$, an admissible run of the automaton are trajectories $\left(\left(x(t)^\mathsf{T}\ q(t)^\mathsf{T}\right)^\mathsf{T}\right)_{t=t_0}^{t_f}$ complying to the following rules: Starting in the initial phase q_0 and initial continuous state x_0, the state $x(t)$ evolves according to the flow function $f(q_0)$, depending on the current state and input. A transition into a new phase q_j according to a transition $\theta \in \Theta$ is only possible if x is contained in the guard set $g(\theta)$. The evolution of the continuous state $x(t)$ in phase q_j is bounded to the condition $x(t) \in \text{inv}(q_j)$.

In the following, it is assumed that a target set $\mathcal{X}_\text{T}$ is defined and reachable from the initial set $\mathcal{X}_0$ by an admissible run. It is further assumed that all sets in the hybrid automaton are polyhedral. A maneuver is then defined as follows:

Definition 2. (Maneuver). A maneuver is a tuple $\texttt{M} = (\mathcal{C}, H_\text{plan}, \texttt{HA})$, consisting of a finite set $\mathcal{C} \subseteq \mathcal{G}$ of involved vehicles, a time horizon H_plan, and a hybrid automaton $\texttt{HA}$. The vehicles carry information about their current state and their preferences (e.g. regarding driving style, energy expenditure) in form of a cost function, while H_plan defines the time horizon over which a plan has to be made. The hybrid automaton encodes compliant vehicle behaviors (see more details in (Eilbrecht and Stursberg, 2018, 2019)).

A maneuver can specify behaviors of one or more vehicles. A collection of different maneuvers will be referred to as *maneuver library*, envisioned to be used within the following, hierarchical procedure: Based on the maneuver library, a high-level control algorithm determines a group $\mathcal{C}$ (termed *coalition*) of vehicles which are to perform a selected maneuver of a certain duration H_plan. For each coalition, a lower-level trajectory planner then determines reference trajectories for all cooperative vehicles.

In this work, the choice of a maneuver, its duration, and the corresponding coalition is assumed to be determined already (Eilbrecht and Stursberg, 2018, 2019), such that the remaining focus can be set on the problem of efficiently planning reference trajectories. These can be seen as solutions to finite-horizon hybrid optimal control problems:

Problem 3. (Hybrid Optimal Control Problem). Given a maneuver $\texttt{M}$ and a cost functional:

$$J(x_0, u(\cdot), H_\text{plan}) = \int_0^{H_\text{plan}} \|Cx - \bar{y}_q\|_Q + \|u\|_R \, dt, \quad (1)$$

determine an admissible run of $\texttt{HA}$ which minimizes J and satisfies:

$$x(H_\text{plan}) \in \mathcal{X}_\text{T}, \quad H_\text{plan} \in \mathbb{T}. \quad (2)$$

Here, $\mathbb{T} = [0, t_\text{max}]$ is a finite time interval, $\bar{y}_q \in \mathbb{R}^{n_y}$ allows to consider constant, location-dependent reference values for n_y state variables selected by $C \in \mathbb{R}^{n_y \times n_x}$, and $Q \geq 0$ and $R \geq 0$ are weighting matrices of suitable dimensions.

In theory, this problem can be solved by dynamic programming, requiring to solve the Hamilton-Jacobi-Bellman equation associated to the problem. The solution would be a so-called policy function $\pi : \mathbb{X} \times \mathbb{T} \to \mathbb{R}^{n_u}$, assigning to each state at time t the optimal input u^*:

$$u^*(x,t) = \pi(x,t). \tag{3}$$

While explicit solutions can be obtained only for very simple cases, more complex problems require numerical solution. These are, however, severely limited by the computational complexity. Therefore, the approach in this paper refrains from obtaining exact solutions, but rather approximates these.

Approximate dynamic programming, also known as adaptive or neuro-dynamic programming, addresses the computational complexity by using function approximators, such as feedforward artificial neural networks (Lewis and Vrabie, 2009; Liu et al., 2017; Bertsekas and Tsitsiklis, 1996). Generally, approximate dynamic programming methods alternate between improving the policy and improving the value function. While these approaches typically interlace the process of data generation and training of the approximation structure, here these steps are carried out separately: First, training data is generated by numerical solution of optimal control problems for selected initial states and horizons as detailed in Sec. 3. The data is then used within a supervised learning approach to train an approximation structure as detailed in Sec. 4.

Note that in some cases, this approach is not viable if point-wise computation of (3) by numerical optimal control is impossible, or too time-consuming (e.g. (Limebeer and Rao, 2015)) to generate a sufficient amount of training data. Thus, some approaches resort to approaches like reinforcement learning. The enabling aspect in the approach proposed here is the fact that the considered problem is highly structured due to the maneuvers.

3. NUMERICAL OPTIMAL CONTROL FOR GENERATION OF TRAINING DATA

To solve Problem 3 for selected initial states numerically, solvers such as $\mathbb{GPOPS}-\mathbb{II}$ (Patterson and Rao, 2014) can be used. To do so, Problem 3 can be reformulated into the following multiple-phase optimal control problem:

Problem 4. (Multiple-Phase Formulation). For each phase $q \in \{q_1, q_2, \ldots, q_P\}$ in a given phase sequence for a maneuver, find an initial time $t_0^{(q)}$, a final time $t_f^{(q)}$, an initial state $x_0^{(q)} := x^{(q)}(t_0^{(q)})$, a final state $x_f^{(q)} := x^{(q)}(t_f^{(q)})$, the trajectory $x^{(q)}(t)$ for the time interval $[t_0^{(q)}, t_f^{(q)}]$, and the controls $u^{(q)}(t)$ such that the cost functional:

$$J = \sum_{i=1}^{P} J^{(i)} \tag{4}$$

is minimized subject to the path constraints:

$$c_{\min}^{(q)} \leq c^{(q)}\left(x^{(q)}, u^{(q)}, t^{(q)}\right) \leq c_{\max}^{(q)}, \tag{5}$$

the event constraints:

$$b_{\min} \leq b\left(x_0^{(q_1)}, x_0^{(q_2)}, \ldots, x_0^{(q_P)}, t_0^{(q_1)}, t_0^{(q_2)}, \ldots, t_0^{(q_P)}, x_f^{(q_1)}, x_f^{(q_2)}, \ldots, x_f^{(q_P)}, t_f^{(q_1)}, t_f^{(q_2)}, \ldots, t_f^{(q_P)}\right) \leq b_{\max}, \tag{6}$$

and the dynamic constraints:

$$\dot{x}^{(q)}\left(t^{(q)}\right) = f^{(q)}\left(x^{(q)}\left(t^{(q)}\right), u^{(q)}\left(t^{(q)}\right), t^{(q)}\right). \tag{7}$$

The reformulation requires the following steps:

- consider the phases of the hybrid automaton as the phases of the multiple-phase optimal control problem;
- employ the polytopes $\mathbb{X}$, $\mathbb{U}$, and $\mathrm{inv}(q)$ of each phase $q \in \mathcal{Q}$ as path constraints;
- use the event constraints to define the polytopes $\mathcal{X}_0$, $\mathcal{X}_T$, and $g(\theta)$ for each transition $\theta \in \Theta$. Further, for each $\theta = (q_i, q_j) \in \Theta$, use the event constraints to enforce that $x_f^{(q_i)} - x_0^{(q_j)} = 0$ and $t_f^{(q_i)} - t_0^{(q_j)} = 0$. Additionally, specify the initial state x_0, the initial time t_0, and the bounds on the final time in the event constraints;
- define the dynamic constraints by choosing $f^{(q)} = f(q)$ for each $q \in \mathcal{Q}$;
- partition (1) according to (4).

A solver like $\mathbb{GPOPS}-\mathbb{II}$ employs a Legendre-Gauss-Radau quadrature method with orthogonal collocation to solve this problem and it returns the states, controls, and time values for each collocation point.

In this work, the training data set is defined as a collection of so-called *samples*. A sample consists of a point t in time, a state vector $x(t)$, and an input vector $u(t)$. It is generated for each collocation point resulting from numerical solution of Problem 4.

4. APPROXIMATING OPTIMAL SOLUTIONS BY NEURAL NETWORKS

This section details the approximation of the policy function, describing both its architecture and the computation of its parameters.

4.1 Approximation Architecture

In this work, feedforward artificial neural networks (Goodfellow et al., 2016; Sutton and Barto, 2018; Hagan and Menhaj, 1994) are used to approximate policy (3). Only networks with one hidden layer and one linear output layer are considered to keep the network architecture simple, see Fig. 1. This decision can be justified, theoretically, by the universal approximation theorem (Cybenko, 1989), stating that the networks described subsequently can approximate policy (3) with arbitrarily small error. Nevertheless, the choice of a suitable architecture and the determination of the parameters is a challenging task, as outlined in (Goodfellow et al., 2016).

The hidden layer and output layer each consist of a chosen number of units with input and output signals. A unit maps a weighted sum of its inputs to its output. The output of a unit is called *activation* and the underlying mapping *activation function*. This work focuses on networks where the activation functions of the output units are chosen to be identity functions, while the activation functions of the hidden units are selected as:

$$\tanh(z) = \frac{2}{1+\exp(-2z)} - 1. \tag{8}$$

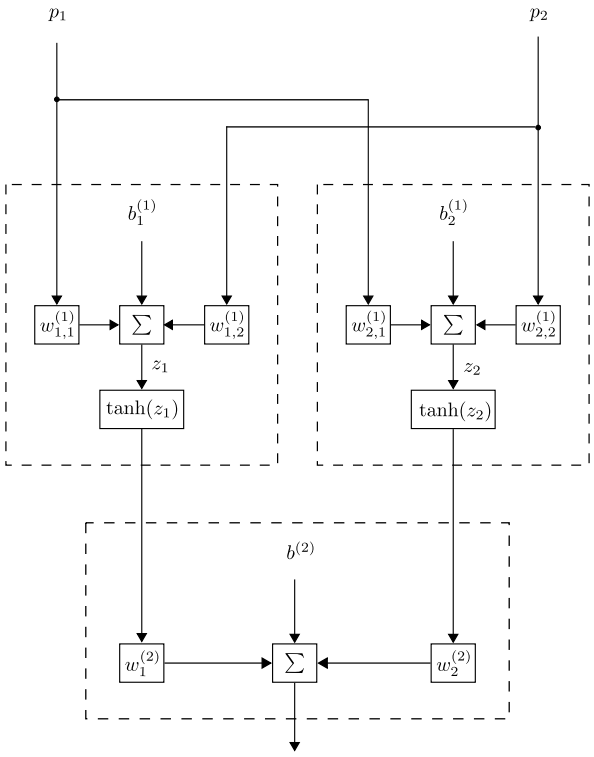


Fig. 1. Feedforward artificial neural network with two inputs, two hidden units and one output unit.

The approximated policy is defined as:

$$\hat{u} = \hat{\pi}(x,t,r) := \left[\sigma_1(p,\vartheta_1)\ \sigma_2(p,\vartheta_2)\ \dots\ \sigma_{n_u}(p,\vartheta_{n_u})\right]^\mathsf{T},$$
$$p := \left[x^\mathsf{T}\ t\right]^\mathsf{T}, \quad r := \left[\vartheta_1\ \vartheta_2\ \dots\ \vartheta_{n_u}\right], \tag{9}$$

where ϑ_i contains the tunable parameters of network $i \in \{1,2,\dots,n_u\}$ and $\sigma_i(p,\vartheta_i)$ describes the mapping from its input p to the i-th element of $\hat{u} = [\hat{u}_1, \hat{u}_2, \dots, \hat{u}_{n_u}]^\mathsf{T}$. In the remaining part of this section, the focus is exemplarily on one of the networks to describe the details. For simplicity, the index of the network is omitted in the rest of this section.

Let $n_p := n_x + 1$ denote the number of inputs of the network and n_L the number of units in the hidden layer. The mapping from the input to the output of the network is given by:

$$\sigma(p,\vartheta) = \left(\sum_{i=1}^{n_L} w_i^{(2)} \tanh\left(\left(\sum_{j=1}^{n_p} w_{i,j}^{(1)} p_j\right) + b_i^{(1)}\right)\right) + b^{(2)}, \tag{10}$$

where the parameter vector:

$$\vartheta = \left[w_{1,1}^{(1)}\ w_{1,2}^{(1)}\ \dots\ w_{n_L,n_p}^{(1)}\ b_1^{(1)}\ \dots\ b_{n_L}^{(1)}\ w_1^{(2)}\ \dots\ b^{(2)}\right] \tag{11}$$

consists of the weights $w_{i,j}^{(1)}, w_i^{(2)} \in \mathbb{R}$ and biases $b_j^{(1)}, b^{(2)} \in \mathbb{R}$.

4.2 Determination of the Parameter Vector

For each network, the number of parameters and their values are to be found within an offline procedure, such that the approximated policy is an adequate approximation of (3). Note that the number of parameters is specified by the chosen network, for which the number of hidden units is a matter of choice. In Sec. 5, the performance is improved ("*trained*") by using the neural net fitting tool of the deep learning toolbox within Matlab. The following part describes the fundamental aspects of training. The tool divides the training data into *training samples*, *validation samples*, and *test samples*. In the training procedure itself, only the training and validation samples are used to improve the performance. The test samples, on the other hand, serve as a measure of how good the network approximates samples that not contributed to the training. This property is known as *generalization*.

Here, mean squared errors quantify the performance of the training, validation, and test samples. Following the distinction above, these errors are termed *training errors*, *validation errors*, and *test errors*.

The Levenberg-Marquardt algorithm (Hagan and Menhaj, 1994) is a well-established optimization algorithm that can be used to adapt the parameter vector by iteratively minimizing the training error for a user-defined number of iterations (the *epochs*). In addition, the training procedure monitors the validation error in each iteration and terminates upon an increase, which would indicate so-called *overfitting*, indicating that too many units in the hidden layer are chosen. On the other hand, underfitting may occur if a too small number of units in the hidden layer are selected, see (Goodfellow et al., 2016).

5. EXAMPLE OF A COOPERATIVE OVERTAKING MANEUVER

5.1 Maneuver formulation

In order to demonstrate the efficacy of the proposed procedure, the cooperative overtaking maneuver sketched in Fig. 2 is addressed. This scenario has already been considered in prior work with solutions based on linear vehicle models and exact computation (Eilbrecht and Stursberg, 2018), or approximations thereof (Eilbrecht and Stursberg, 2019). In the maneuver, vehicle 1 is to overtake vehicle 2, what must be enabled by vehicle 2 and the oncoming vehicle 3. In addition, the vehicles 2 and 3 must keep their lanes during the maneuver.

Contrasting the earlier work, the vehicle dynamics are here based on nonlinear bicycle models (Liniger et al., 2015), as illustrated in Fig. 3. Referring to an earth-fixed coordinate system, longitudinal position and velocity of vehicle $i \in \{1,2,3\}$ are given by $p_x^{(i)}$ and $v_x^{(i)} := \dot{p}_x^{(i)}$, respectively. Analogously, $p_y^{(i)}$ and $v_y^{(i)} := \dot{p}_y^{(i)}$ describe the lateral components. Longitudinal and lateral velocities in a body-fixed coordinate system are denoted by $v_{\tilde{x}}^{(i)}$ and $v_{\tilde{y}}^{(i)}$. These coordinate systems are related by:

$$v_x^{(i)} = v_{\tilde{x}}^{(i)} \cos\left(\varphi^{(i)}\right) - v_{\tilde{y}}^{(i)} \sin\left(\varphi^{(i)}\right),$$
$$v_y^{(i)} = v_{\tilde{x}}^{(i)} \sin\left(\varphi^{(i)}\right) + v_{\tilde{y}}^{(i)} \cos\left(\varphi^{(i)}\right),$$

with yaw angle $\varphi^{(i)}$. Since the vehicles 2 and 3 must keep their lanes, it follows for $i \in \{2,3\}$ that $\varphi^{(i)} = 0$ and $v_y^{(i)} = 0$ during the maneuver, implying also $v_x^{(i)} = v_{\tilde{x}}^{(i)}$ and $v_y^{(i)} = v_{\tilde{y}}^{(i)}$. Consequently, it is sufficient to consider only

the longitudinal position and longitudinal velocity in the earth-fixed coordinate system for vehicle 2 and vehicle 3. The state vector of the cooperating group is chosen to:

$$x = \left[p_{\text{rel}}^{(1)}\ p_{\text{rel}}^{(2)}\ p_y^{(1)}\ v_x^{(1)}\ v_x^{(2)}\ v_x^{(3)}\ v_y^{(1)}\ v_{\tilde{x}}^{(1)}\ v_{\tilde{y}}^{(1)}\ \varphi^{(1)}\right]^\mathsf{T},$$

with $p_{\text{rel}}^{(1)} := p_x^{(2)} - p_x^{(1)}$, $p_{\text{rel}}^{(2)} := p_x^{(3)} - p_x^{(1)}$ denoting relative positions. As controlled inputs, the accelerations in the body-fixed frame: $a_{\tilde{x}}^{(i)} := \dot{v}_{\tilde{x}}^{(i)}$, $i \in \{1,2,3\}$, $a_{\tilde{y}}^{(1)} := \dot{v}_{\tilde{y}}^{(1)}$, and the yaw rate $\omega^{(1)} = \dot{\varphi}^{(1)}$ are chosen such that:

$$u = \left[a_{\tilde{x}}^{(1)}\ a_{\tilde{x}}^{(2)}\ a_{\tilde{x}}^{(3)}\ a_{\tilde{y}}^{(1)}\ \omega^{(1)}\right]^\mathsf{T}.$$

The dynamics is then defined as $\dot{x} = f(x,u)$ with:

$$f(x,u) = \begin{bmatrix} x_5 - x_4 \\ x_6 - x_4 \\ x_7 \\ (u_1 - u_5 x_9)\cos(x_{10}) - (u_4 + u_5 x_8)\sin(x_{10}) \\ u_2 \\ u_3 \\ (u_1 - u_5 x_9)\sin(x_{10}) + (u_4 + u_5 x_8)\cos(x_{10}) \\ u_1 \\ u_4 \\ u_5 \end{bmatrix}.$$

The overtaking maneuver is divided into three phases: In the first one, vehicle 1 is behind vehicle 2, in the second one, vehicle 1 overtakes and is on the left side of the road, and in the third phase, vehicle 1 drives in between vehicle 2 and the oncoming vehicle 3. The phases correspond to the set of locations $\mathcal{Q} = \{q_1, q_2, q_3\}$, where transitions are only allowed according to: $\Theta = \{\theta_{1,2} = (q_1, q_2), \theta_{2,3} = (q_2, q_3)\}$, what does not allow to fall back to a prior phase. Reference trajectories are prescribed for the states $p_y^{(1)}$, $v_x^{(1)}$, $v_x^{(2)}$, $v_x^{(3)}$, and $v_y^{(1)}$, such that $C = [0_{5\times2}\ I_{5\times5}\ 0_{5\times3}]$. This work considers $\bar{y}_1 = \bar{y}_3 = \left[p_y^{(2)}\ 20\ 20\ -20\ 0\right]$ and $\bar{y}_2 = \left[p_y^{(3)}\ 20\ 20\ -20\ 0\right]$ as reference values (in m resp. $\mathrm{m\,s^{-1}}$), and $Q = I_{5\times5}$, $R = I_{5\times5}$ as weight matrices. Safety distances in longitudinal and lateral direction are specified as $l_{x,\text{safe}} = 5$ and $l_{y,\text{safe}} = 5$, respectively.

5.2 Policy approximation

Problem 4 was established and solved for 2700 initial states randomly selected from the set $\mathcal{X}_0$. The time interval was specified to $\mathbb{T} = [0, 20]$. Overall, 2609 feasible solutions were found and lead to the generation of 316334 samples for the training data set. Fig. 4 illustrates the times that were needed to compute each of the 2609 solutions in a box plot diagram without and with outliers.

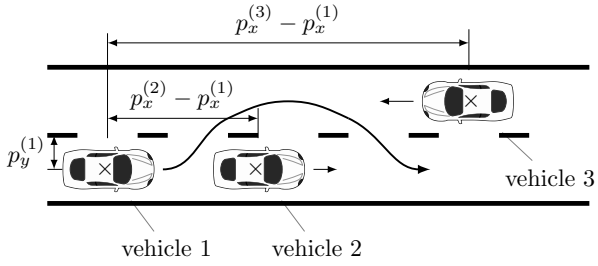


Fig. 2. Example Maneuver: overtaking with oncoming traffic.

For each network used to approximate the policy, 40 hidden units were chosen and 521 parameters were determined per network (480 weights and 41 biases) according to the training procedure described in Sec. 4. The same structure was used for each network. The training procedure of each network divided the samples randomly in 221434 samples for training, 47450 for validation, and 47450 for testing. Table 1 presents the obtained training, validation, and test errors, while Table 2 shows for each network the time and the number of epochs required by the training. The network index $i \in \{1, \ldots, 5\}$ in both tables refers to the network corresponding to the mapping $\sigma_i(p, \vartheta_i)$ (see Sec. 4). A number of epochs lower than 1000 (maximum number) indicates that a detected increase in the validation error caused the training to terminate. The times to evaluate the approximated policy for all samples is illustrated as a box plot diagram in Fig. 5.

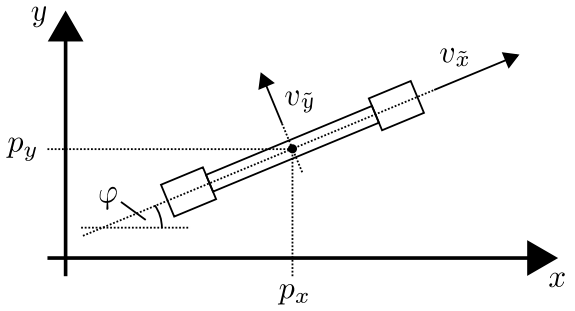


Fig. 3. Bicycle model illustrated within an earth-fixed coordinate system.

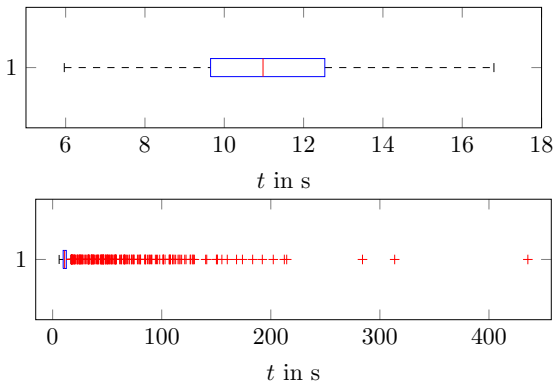


Fig. 4. Computation times for $\mathbb{GPOPS} - \mathbb{II}$ solutions in seconds ((a) without outliers, (b) with outliers).

Table 1. Training, validation and test errors.

Network index	Training error	Validation error	Test error
1	$3.7 \cdot 10^{-2}$	$3.8 \cdot 10^{-2}$	$3.8 \cdot 10^{-2}$
2	$2.5 \cdot 10^{-2}$	$2.8 \cdot 10^{-2}$	$2.7 \cdot 10^{-2}$
3	$1.7 \cdot 10^{-2}$	$1.8 \cdot 10^{-2}$	$1.7 \cdot 10^{-2}$
4	$9.6 \cdot 10^{-3}$	$1.1 \cdot 10^{-2}$	$1.0 \cdot 10^{-2}$
5	$4.1 \cdot 10^{-4}$	$4.1 \cdot 10^{-4}$	$3.9 \cdot 10^{-4}$

Table 2. Required training times and epochs.

Network index	1	2	3	4	5
Training time	823 s	701 s	559 s	605 s	693 s
Epochs	735	642	506	546	622

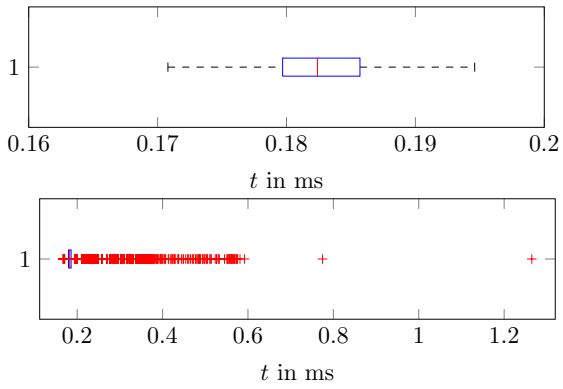


Fig. 5. Times to evaluate the approximated policy in milliseconds ((a) without outliers, (b) with outliers).

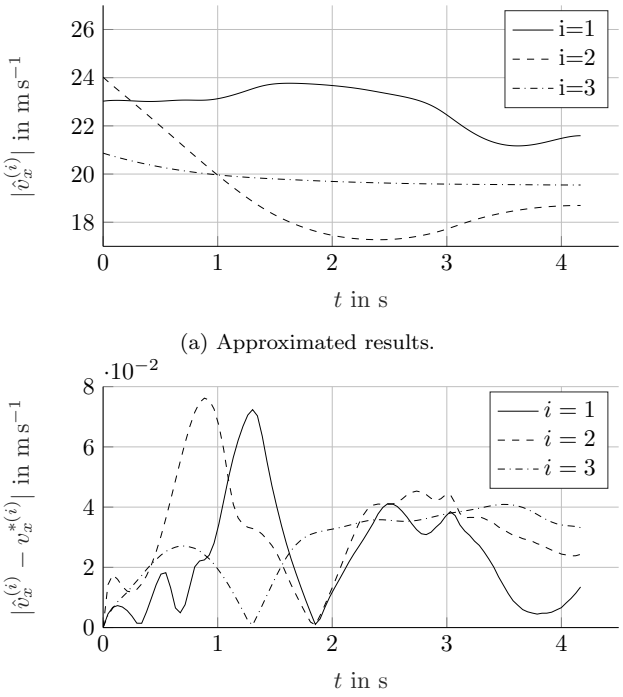
For all of the 2609 initial states for which an optimal solution could be obtained, the admissibility of the approximated solution was checked. This check verifies whether the approximated policy leads to a state sequence from x_0 to $\mathcal{X}_T$ within the time interval $\mathbb{T}$. If so, it is checked whether each attained state is an element of at least one invariant, and whether each control from the corresponding control trajectory is an element of $\mathbb{U}$. It turned out that the admissibility was obtained for 90% of the 2609 runs.

The solutions obtained by $\mathbb{GPOPS}-\mathbb{II}$ are interpreted as optimal policies ($*$) and those obtained by the networks as approximated policies ($\wedge$). Figs. 6-8 show approximated and optimal results obtained for an exemplary initial state. In detail, Fig. 6a illustrates the approximated absolute longitudinal velocities of the three vehicles involved in the overtaking maneuver. The absolute differences between the approximated and optimal longitudinal velocities are shown in Fig. 6b. Both, the optimal and the approximated lateral velocity of the overtaking vehicle are contained in Fig. 7. Finally, approximated positions of the vehicles and the corresponding deviations from the optimal ones are illustrated in Fig. 8. Note that no collision between vehicle 1 and vehicle 2 occurs, since they pass the same part of the road at different times.

6. CONCLUSIONS

This paper has proposed a novel approach to planning reference trajectories for cooperative autonomous vehicles with nonlinear dynamics. A central element is policy design by feedforward artificial neural networks. The policies are used by a trajectory planner to determine reference trajectories for all cooperative vehicles. It was demonstrated that the concept allows for efficient generation of training data by solving highly structured optimal control problems numerically. Even though the network design and training was kept simple, results very similar to the optimal ones have been obtained, as shown in Figs. 6-8 for an exemplary initial state.

The computation times in Fig. 4 illustrate, however, that the software is not suited for real-time application. Moreover, the outliers demonstrate that the computation times may be unpredictable high in rare cases. However, the solution times are low enough for the generation of training data, not at least due to the highly structured optimal



(a) Approximated results.

(b) Absolute differences between approximated and optimal results.

Fig. 6. Longitudinal velocities for an exemplary initial state.

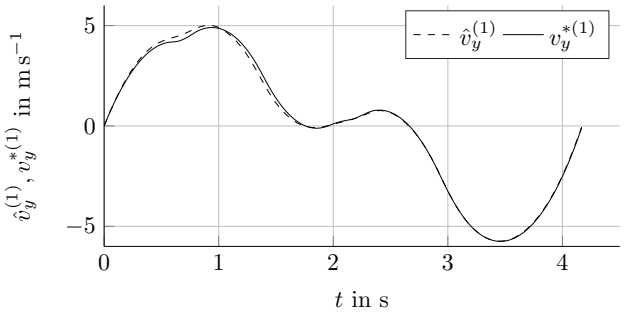
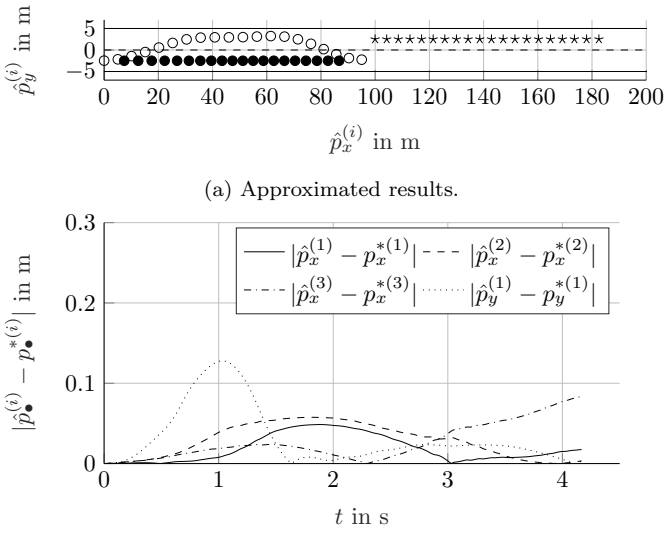


Fig. 7. Lateral velocities for an exemplary initial state.



(a) Approximated results.

(b) Absolute differences between approximated and optimal results.

Fig. 8. Positions for an exemplary initial state.

control problem obtained from the maneuver concept. In cases in which the optimization cannot find a solution, the use of termination criteria are an option to reduce the time effort. In contrast, Fig. 5 shows that the approximated policies can be evaluated very fast.

It has to be said, though, that the tuning of several parameters in the design and training of feedforward artificial neural networks is often a heuristic and iterative procedure. Thus, simple network structures and a well-established training procedure were used here to demonstrate the efficiency of the method.

In future work, benefits of more sophisticated network structures for the approximation quality will be evaluated.

REFERENCES

Bertsekas, D.P. and Tsitsiklis, J.N. (1996). *Neuro-dynamic programming*, volume 5. Athena Scientific Belmont, MA.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4), 303–314.

Eilbrecht, J. and Stursberg, O. (2018). Optimization-based maneuver automata for cooperative trajectory planning of autonomous vehicles. In *Proc. of the European Control Conference*, 82–88.

Eilbrecht, J. and Stursberg, O. (2019). Reducing computation times for planning of reference trajectories in cooperative autonomous driving. In *Proc. of the Intelligent Vehicle Symposium*, 114–120. IEEE.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. The MIT Press.

Hagan, M. and Menhaj, M. (1994). Training feedforward networks with the marquardt algorithm. *IEEE Transactions on Neural Networks*, 5(6), 989–993.

hwan Jeon, J., Cowlagi, R.V., Peters, S.C., Karaman, S., Frazzoli, E., Tsiotras, P., and Iagnemma, K. (2013). Optimal motion planning with the half-car dynamical model for autonomous high-speed driving. In *2013 American Control Conference*, 188–193. IEEE.

LaValle, S.M. (2006). *Planning Algorithms*. Cambridge University Press.

Lewis, F.L. and Vrabie, D. (2009). Reinforcement learning and adaptive dynamic programming for feedback control. *IEEE Circuits and Systems Magazine*, 9(3), 32–50.

Limebeer, D.J.N. and Rao, A.V. (2015). Faster, higher, and greener: Vehicular optimal control. *IEEE Control Systems*, 35(2), 36–56.

Liniger, A., Domahidi, A., and Morari, M. (2015). Optimization-based autonomous racing of 1:43 scale rc cars. *Optimal Control Applications and Methods*, 36(5), 628–647.

Liu, D., Wei, Q., Wang, D., Yang, X., and Li, H. (2017). *Adaptive dynamic programming with applications in optimal control*. Springer International Publishing.

Patterson, M.A. and Rao, A.V. (2014). GPOPS-II: A matlab software for solving multiple-phase optimal control problems using hp-adaptive gaussian quadrature collocation methods and sparse nonlinear programming. *ACM Transactions on Mathematical Software*, 41(1), 1–37.

Qian, X., Altché, F., Bender, P., Stiller, C., and de La Fortelle, A. (2016). Optimal trajectory planning for autonomous driving integrating logical constraints: An MIQP perspective. In *2016 19th Int. Conf. on Intelligent Transportation Systems*, 205–210. IEEE.

Schouwenaars, T., De Moor, B., Feron, E., and How, J. (2001). Mixed integer programming for multi-vehicle path planning. In *European Control Conf.*, 2603–2608. IEEE.

Sutton, R.S. and Barto, A.G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press.