Motion Primitives as the Action Space of Deep Q-Learning for Planning in Autonomous Driving

Tristan Schneider, Matheus V. A. Pedrosa, Timo P. Gros, Verena Wolf, and Kathrin Flaßkamp

***Abstract*— Motion planning for autonomous vehicles is commonly implemented via graph-search methods, which pose limitations to the model accuracy and environmental complexity that can be handled under real-time constraints. In contrast, reinforcement learning, specifically the deep Q-learning (DQL) algorithm, provides an interesting alternative for real-time solutions. Some approaches, such as the deep Q-network (DQN), model the RL-action space by quantizing the continuous control inputs. Here, we propose to use motion primitives, which encode continuous-time nonlinear system behavior as the action space. The novel methodology of motion primitives-DQL planning is evaluated in a numerical example using a single-track vehicle model and different planning scenarios. We show that our approach outperforms a state-of-the-art graph-search method in computation time and probability of reaching the goal.**

***Index Terms*— Motion primitives, reinforcement learning, autonomous driving, Markov decision process, optimal control.**

I. Introduction

ALGORITHMIC motion planning is a crucial ability for autonomous navigation. However, considering constraint environments, internal nonlinear system dynamics, and optimization criteria makes solving motion planning problems computationally challenging. Thus, it is a common approach to transform the continuous-time, continuous-state planning problem into a graph search problem to allow the application of fast graph-based search techniques. Within this area of research, [1] introduces motion planning by motion primitives. These primitives are trajectory segments admissible to the original nonlinear dynamical system model of, e.g., an autonomous vehicle. Quantizing the search space via motion primitives gives rise to finite representation via an automaton, in which the concatenation rules of the trajectory segments are encoded. The size of the automaton has to be chosen as a trade-off between accuracy and computational costs. On the one hand, a larger automaton provides more primitive candidates, so better chances to find admissible solutions if obstacle avoidance is crucial and closer approximations of drivable paths from initial points to desired final regions. On the other hand, the computation time of the graph search is a limiting factor of this approach (see, e.g., [2]).

Thus, replacing graph search with reinforcement learning (RL) is a promising approach for the motion planning with primitives technique. Speeding up the computation time, RL has a built-in collision avoidance from (multi-)sensor information, such as the light detection and ranging (LiDAR), while graph search methods need an additional verification step, for example. In this paper, we demonstrate that RL can provide sequences of motion primitives as approximately optimal motion plans in real-time independent of the automaton size.

This paper is built upon three main fields:

A. Motion Primitives

Many technological systems, e.g., mobile robots, cars, or helicopters behave basically invariant w.r.t. translations or rotations in space. This property leads to symmetries in dynamical system models. Motion planning with primitives exploits this property [1]. The symmetry property was exploited in that paper to build the (motion) primitives, originally developed in [1]. In mathematical terms, we consider Lie group structures operating on the dynamical systems' states and we represent trajectories via equivalence classes w.r.t. the corresponding Lie group actions [1], [3], [4].

B. Graph Search With Primitives

A* search is the most known best-first search, where the expansion of the graph's nodes is based on an evaluation function [5]. This function is a sum of two costs: 1) the cost from the initial node to the one being evaluated, plus 2) the estimated cost, given by a heuristic function, from the evaluated node to the goal. A famous extension of A* graph search is the Hybrid A* [6], [7], which associates a continuous state with a discrete grid cell. A*-based searches have also been extended to motion panning with primitives [8], [9], [10], [11], [12]. The A* and the Hybrid A* search rely on grid cells of the search space. Contrarily, the Optimized Primitives (Π*) search admits any continuous point in the state space [11]. Also, an optimization problem of reduced complexity makes an adjustment in the duration of some primitives. This allows the trajectory to end exactly at an arbitrary point (in the continuous state space), rather than in a region around it.

Manuscript received 14 November 2023; revised 23 May 2024; accepted 11 July 2024. This work was supported by the Deutsche Forschungsgemeinschaft (German Research Foundation) within the Priority Program SPP 1835 "Cooperative Interacting Automobiles" under Grant KO 1430/17-1. The Associate Editor for this article was H. H. Song. *(Corresponding author: Matheus V. A. Pedrosa.)*

Tristan Schneider, Matheus V. A. Pedrosa, and Kathrin Flaßkamp are with the Chair of Systems Modeling and Simulation, Systems Engineering, Saarland University, 66123 Saarbrücken, Germany (e-mail: tristanschneider2000@gmail.com; matheus.pedrosa@uni-saarland.de; kathrin.flasskamp@uni-saarland.de).

Timo P. Gros and Verena Wolf are with the Chair of Modeling and Simulation, Saarland Informatics Campus, Saarland University, 66123 Saarbrücken, Germany (e-mail: timopgros@cs.uni-saarland.de; wolf@cs.uni-saarland.de).

Digital Object Identifier 10.1109/TITS.2024.3436530

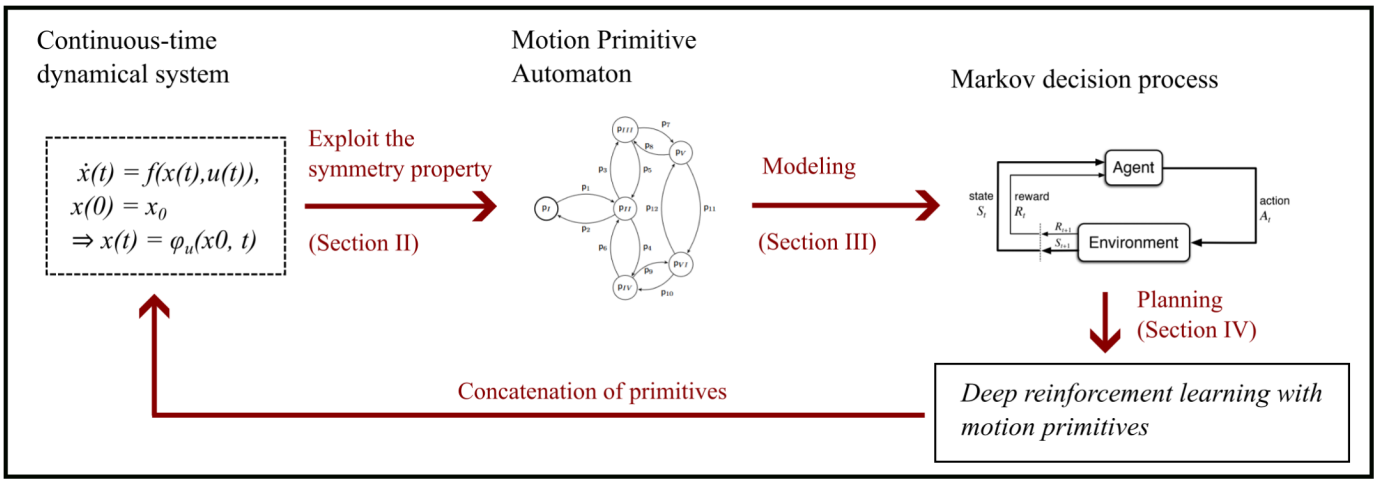

Fig. 1. Thematic outline of our proposed methodology.

C. Reinforcement Learning for Motion Planning

Prior work extensively addresses the application of RL in the context of a simple motion planning problem [13], [14], [15], [16]. Although there have been efforts to integrate RL with verification techniques [17], [18], [19], none of these approaches are capable of handling motion primitives, as they assume a discrete action space instead.

The main contribution of this work is the novel implementation of a RL algorithm for motion planning considering motion primitives as the action space. We show the benefits of this methodology in comparison with the state-of-the-art graph search algorithm for motion primitives, in terms of the success rate, the number of steps necessary for reaching the goal, and the execution time.

The remainder of this paper is organized as follows: In Section II, we summarize the theoretical framework that serves as the basis for our methodology; Section III describes the application of primitives as the action space on an RL; Section IV presents a simulation example for the RL and a comparison with the graph search approaches, together with the discussion; lastly, Section V shows the concluding remarks and future works. After the appendix, one can find the list of acronyms and symbols used in this paper.

II. Theoretical Background

In the following, we summarize the theoretical foundations, composed of the topics that ground the description and analysis of our proposed algorithm: we start introducing the dynamical system notation, and then we present the mathematical foundation to construct motion primitives. After defining a Markov decision process (MDP), we finish this section with the description of the deep Q-learning (DQL), which is a commonly used RL method. Figure 1 gives an outlook on the combined methodology we propose and evaluate within this paper.

A. Dynamical System

We assume the dynamical system is modeled by a nonlinear time-invariant ordinary differential equation

$$\dot{x}(t) = f(x(t), u(t)), \tag{1}$$

where $t \in \mathbb{R}_+$ is the time, $x \in \mathfrak{M}$ denotes the state as an element of the state manifold $\mathfrak{M}$, $u \in \mathcal{U}$ denotes the control u from a given set $\mathcal{U}$ and $f : \mathfrak{M} \times \mathcal{U} \to T\mathfrak{M}$ is the vector field in Equation (1).

For any appropriate control signal $u|_{[0,\mathrm{T}]}$ on a given time interval $[0, \mathrm{T}]$, $\mathrm{T} > 0$, $u(t) \in \mathcal{U}$ for all $t \in [0, \mathrm{T}]$, we further assume existence and uniqueness of solutions, i.e., trajectories, which can be described via the dynamical system's flow $\varphi_u : \mathfrak{M} \times \mathbb{R}_+ \to \mathfrak{M}$. That is,

$$x(t) = \varphi_u(x_0, t) \tag{2}$$

on $[0, T]$, $T \geq 0$, for a fixed control signal $u|_{[0,\mathrm{T}]}$ and initial state x_0. One could easily generalize to time intervals $[t_i, t_f]$, $0 \leq t_i \leq t_f$ since we do not consider explicitly time-dependent (i.e. non-autonomous) vector fields.

B. Motion Primitives

The existence of motion primitives, as they had been introduced by Frazzoli et al. [1], is tied to the symmetry properties of the system. Let $\mathcal{G}$ be a Lie group with an identity element e. Let a left action of the group $\mathcal{G}$ on the state manifold $\mathfrak{M}$ of (1) be represented by the left-invariant smooth map $\Psi : \mathcal{G} \times \mathfrak{M} \to \mathfrak{M}$, such that $\Psi(g_2, \Psi(g_1, x)) = \Psi(g_2 g_1, x)$, for $g_1, g_2 \in \mathcal{G}$. As an abbreviation, we define $\Psi_g(x) := \Psi(g, x)$ with $g \in \mathcal{G}$.

Definition 1 (Symmetry Group): The triple $(\mathcal{G}, \mathfrak{M}, \Psi)$ is a symmetry group for the dynamical system (1), if the property

$$\varphi_u(\Psi(g, x_0), t) = \Psi(g, \varphi_u(x_0, t)) \tag{3}$$

holds for all $(g, x_0, t) \in \mathcal{G} \times \mathfrak{M} \times [0, \mathrm{T}]$, $u \in \mathbb{R}^m$.

For example, many mechanical systems possess subgroups of $SE(n)$ as symmetry groups. It is homeomorphic to $\mathbb{R}^n \times SO(n)$, so, in other words, the symmetry consists of rotations and translations [4].

By exploiting the symmetry property, we can design motion primitives.

Definition 2 (Motion Primitive): Given a tuple $\mathsf{p} : t \in [0, \mathrm{T}] \to (x(t), u(t))$, consisting of a trajectory and its corresponding input signal both on time interval $[0, T]$, a motion

primitive is the class of tuples equivalent *to* p. *Here, two tuples*

$$\mathsf{p}_1 : t \in [t_{1,i}, t_{1,f}] \to (x_1(t), u_1(t)) \text{ and}$$
$$\mathsf{p}_2 : t \in [t_{2,i}, t_{2,f}] \to (x_2(t), u_2(t)),$$

are said to be equivalent*, if*

1) $t_{1,f} - t_{1,i} = t_{2,f} - t_{2,i}$ *and*
2) *there exists* $g \in \mathcal{G}$, $\varrho \in \mathbb{R}$ *such that* $(x_1(t), u_1(t)) = (\Psi_g(x_2(t-\varrho)), u_2(t-\varrho))$ $\forall t \in [t_{1,i}, t_{1,f}]$.

Further, we can *concatenate* two primitives

$$\mathsf{p}_a : t \in [0, \mathrm{T}_a] \to (x_a(t), u_a(t)) \text{ and}$$
$$\mathsf{p}_b : t \in [0, \mathrm{T}_b] \to (x_b(t), u_b(t)),$$

if there exists $g \in \mathcal{G}$ such that

$$x_a(\mathrm{T}_a) = \Psi_g\big(x_b(0)\big), \tag{4}$$

into a resulting primitive $\mathsf{p} := \mathsf{p}_a\mathsf{p}_b$ such that

$$\mathsf{p} : t \in [0, \mathrm{T}_a + \mathrm{T}_b] \to \begin{cases} \big(x_a(t), u_a(t)\big) & \text{if } t \in [0, \mathrm{T}_a] \\ \big(\Psi_g\big(x_b(t-\mathrm{T}_a)\big), u_b(t-\mathrm{T}_a)\big) & \text{if } t \in (\mathrm{T}_a, \mathrm{T}_a+\mathrm{T}_b]. \end{cases} \tag{5}$$

In [1], two types of primitives are introduced: *trim primitives* and *maneuvers*. While the first ones are defined as relative equilibria of the system, when control inputs are kept constant (or trimmed), the maneuvers are modeled as an arbitrarily controlled trajectory that connects pairs of trim primitives. In the following, we formalize these concepts.

Definition 3 (Trim Primitive): Let $(\mathcal{G}, \mathfrak{M}, \Psi)$ *be a symmetry group in the sense of Definition 1,* $\mathfrak{g}$ *be the Lie algebra of* $\mathcal{G}$ *with exponential map* $\exp : \mathfrak{g} \to \mathcal{G}$*, and consider a constant control input* $\bar{u} \in \mathcal{U}$*. A trajectory* $x : t \in [0, \tau] \to \varphi_{\bar{u}}(x_0, t)$ *on some time interval with* $\tau \geq 0$*, together with* $\bar{u}$*, constitutes a trim primitive if there exists a Lie algebra element* $\xi \in \mathfrak{g}$*, such that,* $\forall t \in [0, \tau]$*:*

$$\begin{aligned} x(t) &= \Psi_{\exp(\xi t)}(x_0), \\ u(t) &= \bar{u}. \end{aligned} \tag{6}$$

Thus, trim primitives can be generated via the symmetry action Ψ via the time-parameterized group orbit $t \in [0, \tau] \to \exp(\xi t) \in \mathcal{G}$. Therefore, it becomes unnecessary to evaluate the system's flow $\varphi_{\bar{u}}$, which might be unknown for nonlinear dynamical systems (1).

Note that, assuming unbounded existence properties of the dynamical system, the final time can be chosen arbitrarily with $\tau < \infty$. In fact, trims have originally been introduced with flexible duration [1].

We write $x \in \mathsf{q}$ for the state x being part of the trim primitive q, if $\tilde{\mathsf{q}} : t \in [0, \tau] \to (\Psi_{\exp(\xi t)}(x), u)$ is equivalent to q, meaning they belong to the same primitive.

Definition 4 (Maneuver): A maneuver is a primitive, as defined in Definition 2, that constitutes a link between two trim primitives. It is characterized by

1) *a fixed time duration* T,
2) *the control signal* $u : [0, \mathrm{T}] \to \mathcal{U}$,
3) $x(0)$ *and* $x(\mathrm{T})$ *both correspond to trim primitives.*

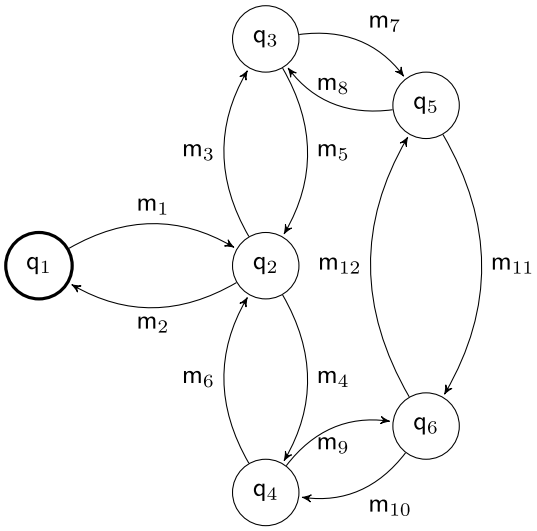

Fig. 2. A maneuver automaton with six trim primitives and 12 maneuvers.

Maneuvers can be computed with optimal control methods, e.g., minimizing the applied energy in the system or a distance to a target [2], [20], [21].

Typically, in systems with Lie group symmetries, an infinite amount of motion primitives can be found (unlike in systems with discrete, e.g. reflection or fixed-angle rotation, symmetries). Thus, for quantized representation, a finite amount of primitives has to be chosen. The choice of the trims can be done manually (e.g., [1], [4], [11]) or data-based [2]. As can be deduced from Definition 4, maneuvers are meant to be concatenated with trim primitives as defined in (5). This way, the concatenation structure between the finite set of motion primitives can be represented by a directed graph, which we call a *motion primitive automaton (MPA)*. In this graph, as illustrated in Figure 2, the trims are the vertices, and the maneuvers are the edges. They are represented in this paper, respectively, by $\mathsf{q} \in \mathsf{Q}$ and $\mathsf{m} \in \mathsf{M}$.

Definition 5 (Motion Primitive Automaton): A motion primitive automaton (MPA) is a 5-tuple $(\mathsf{Q}, \mathsf{M}, \mathsf{h}, \mathsf{q}_0, \mathsf{Q}_f)$ *composed by*:

- Q *is a finite set of trim primitives;*
- M *is a finite set of maneuvers;*
- $\mathsf{h} : \mathsf{Q} \times \mathsf{M} \to \mathsf{Q}$ *is the transition function defining the succeeding trim for a current trim and a concatenated maneuver*[1]*;*
- $\mathsf{q}_0 \in \mathsf{Q}$ *is the initial trim primitive;*
- $\mathsf{Q}_f \subseteq \mathsf{Q}$ *is the set of final trim primitives.*

The motion planning via primitives is done by a *motion plan*, which is a valid path within the graph representation of an MPA, starting in q_0 and ending in Q_f. This can be translated into a finite sequence of alternating trim and maneuver primitives.

Choosing any state $x_0 \in \mathfrak{M}$ such that $x_0 \in \mathsf{q}_0$ on the initial trim, the sequence can be rewritten as concatenated trajectory-control tuples in the sense of (5). Note that because

[1]More formally, $\mathsf{h}(\cdot,\cdot)$ is a partial function defined for every pair $(\mathsf{q}, \mathsf{m}) \in \mathsf{Q} \times \mathsf{M}$ iff $\exists \mathsf{qm}$, i.e., the maneuver m is concatenated with the trim q.

of the property (4), the path in the graph, i.e. the sequence of primitives, translates into a *continuous* trajectory.

A *planning problem* from initial state $x_0 \in \mathfrak{M}$ to some final state $x_f \in \mathfrak{M}$ can be formulated as finding an admissible path in the graph with $x_0 \in \mathsf{q}_0$ to $x_f \in \mathsf{q}_f$, such that $\mathsf{q}_f \in \mathsf{Q}_f$, and suitable durations of the visited trim states.[2] Potentially, the resulting solution shall minimize some given optimization criterion.

While many approaches in the literature solve the planning by graph-search methods (see discussion in Section I), we propose to use deep reinforcement learning (DRL), as it is later explained in detail in Section III. As a prerequisite, terminology of Markov decision processes is needed.

C. Markov Decision Processes

The underlying model of DRL is that of a (state-discrete) Markov decision process in discrete time. Let $\mathcal{D}(S)$ denote the set of probability distributions over S for any non-empty set S.

Definition 6 (Markov Decision Process): A Markov decision process is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mu \rangle$ *consisting of a finite set of* states $\mathcal{S}$*, a finite set of* actions $\mathcal{A}$*, a partial* transition probability function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightharpoonup \mathcal{D}(\mathcal{S})$*, and an* initial distribution $\mu \in \mathcal{D}(\mathcal{S})$*. We say that an action* $a \in \mathcal{A}$ *is* applicable *in state* $s \in \mathcal{S}$*, if* $\mathcal{T}(s, a)$ *is defined. We denote by* $\mathcal{A}(s) \subseteq \mathcal{A}$ *the set of actions applicable in* s*.*

In the context of RL, MDPs are associated with a *reward* function $\mathcal{R}$, which specifies a reward for each transition, i.e., $\mathcal{R} : \mathcal{S} \times A \times \mathcal{S} \to \mathbb{R}$.

An *action policy* gives the probability distribution for the non-deterministic choice of an action given a state. When the policy does not depend on the history of formerly visited states, only on the current one, we say it is *memoryless*. An action policy is *deterministic* if, in each state s, π selects an action with probability one. Then, we simply write $\pi(s)$ for the corresponding action.

Definition 7 (Action Policy): An action policy *is a function* $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ *such that* $\pi(s, \cdot)$ *is a probability distribution on* $\mathcal{A}$ *and, for all* $s \in \mathcal{S}$*,* $\pi(s, a) > 0$ *implies that* $a \in \mathcal{A}(s)$*.*

In the sequel, for a given MDP $\mathcal{M}$ and action policy π, we will write $S_0, S_1, S_2, \ldots$ for the states visited at times $t = 0, 1, 2, \ldots$. Let A_t be the action selected by policy π in state S_t and $R_{t+1} = \mathcal{R}(S_t, A_t, S_{t+1})$ the reward obtained when transitioning from S_t to S_{t+1} with action A_t. For finite-state MDPs, we consider the probability measure associated with these random variables being well defined and $\{S_t\}_{t \in \mathbb{N}_0}$ as a Markov chain with state space $\mathcal{S}$ induced by policy π. For further details, we refer to [22].

D. Deep Q-Learning

In the following, let

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{7}$$

denote the discounted, accumulated reward, also called *return*, from time t on, where $\gamma \in [0, 1]$ is a discount factor, and T is the final time step [23]. The discount factor determines the importance of short or long-term rewards. If $\gamma = 0$, the discounted return will be equal to the reward accumulated in one step only; if $\gamma = 1$ all future rewards will be worth the same; and if $\gamma \in (0, 1)$ the long term rewards will be less important than the short term ones.

Q-learning is a well-known algorithm to approximate action policies that maximize said accumulated reward [24]. For a fixed policy π, the so-called *action-value* or *q-value* $q_\pi(s, a)$ at time t is defined as the expected return G_t that is achieved by taking an action $a \in A(s)$ in state s and following the policy π afterwards, i.e.,

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a \right]. \end{aligned}$$

A policy π is optimal if it maximizes the expected return. We write $q_*(s, a)$ for the corresponding *optimal action-value*. Intuitively, the optimal action-value $q_*(s, a)$ is equal to the expected sum of the reward that we receive when taking action a from state s, and the (discounted) highest optimal action-value that we receive afterward. For optimal π, the Bellman optimality equation [23] gives

$$\begin{aligned} &q_*(s, a) \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma \cdot \max_{a'} q_* \left(S_{t+1}, a' \right) \,\middle|\, S_t = s, A_t = a \right]. \end{aligned}$$

Vice versa, one can evidently obtain the optimal policy if the optimal action values are known by selecting

$$\pi(s) = \operatorname{argmax}_{a \in A(s)} q_*(s, a).$$

By estimating the optimal q-values, one can obtain an approximation of an optimal policy. During tabular *Q-learning*, the action values are approximated separately for each state-action pair [24]. In the case of large state spaces, *DQL* can be used to replace the Q-table by a *neural network* as a function approximator [25]. Neural networks can learn low-dimensional feature representations and express complex non-linear relationships, such that DRL is based on training deep neural network to approximate optimal policies. Here, we consider an neural network with weights θ estimating the Q-value function as a *DQN* [26]. We denote this Q-value approximation by $Q_\theta(s, a)$ and optimize the network w.r.t. the target

$$\begin{aligned} &y_\theta(s, a) \\ &= \mathbb{E}_\theta \left[R_{t+1} + \gamma \cdot \max_{a'} Q_\theta(S_{t+1}, a') \mid S_t = s, A_t = a \right], \end{aligned} \tag{8}$$

where the expectation is taken over trajectories induced by the policy represented by the parameters θ. The corresponding loss function in iteration i of the learning process is

$$L(\theta_i) = \mathbb{E}_{\theta_i} \left[\left(y_{\theta'}(S_t, A_t) - Q_{\theta_i}(S_t, A_t) \right)^2 \right]. \tag{9}$$

Here, the so-called *fixed target* means that, in Equation (9), θ' does not depend on the current iteration's weights of

[2]Note that the final state x_f is a fixed position in $\mathfrak{M}$ while, at the same time, must have the configuration defined by $\mathsf{q}_f \in \mathsf{Q}_f$.

Algorithm 1 Deep Q-Learning Algorithm With Experience Replay [25]

Initialize replay memory D;
Initialize action-value function Q with random weights θ;
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$;
foreach *episode* **do**
  Initialize sequence $s_1 = \{\mathrm{x}_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$;
  foreach *step t of episode* **do**
    With probability ϵ, select a random action a_t, otherwise select $a_t = \mathrm{argmax}_a\, Q_\theta(\phi(s_t), a)$;
    Execute action a_t and observe reward r_t and the observed data x_{t+1};
    Set $s_{t+1} = s_t, a_t, \mathrm{x}_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$;
    Store transition $(\phi_t, a_t, r_t, \mathrm{x}_{t+1})$ in D;
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \mathrm{x}_{j+1})$ from D;
    if *episode terminates at step* $j+1$ **then**
      Set the target value $\mathrm{y}_j = r_j$;
    else
      Set $\mathrm{y}_j = r_j + \gamma \cdot \max_{a'} \hat{Q}_{\theta^-}(\phi_{j+1}, a')$;
    end
    Perform a gradient descent step on $\left(\mathrm{y}_j - Q_\theta(\phi_j, a_j)\right)^2$ with respect to the network parameters θ;
    Every C steps, reset $\hat{Q} = Q$;
  end
end

the (*local*) neural network θ_i, but on weights that were stored in earlier iterations (so-called *target* network), to avoid an unstable training procedure [25]. We approximate $\nabla L(\theta_i)$ and optimize the loss function by stochastic gradient descent. The pseudocode for the DQL, extracted from [25], is presented in Algorithm 1.

Stochastic gradient descent assumes independent and identically distributed samples. Therefore, we can store the samples in an *experience replay buffer* [25], such that, whenever a learning step is performed, we uniformly sample from this replay buffer to consider (approximately) uncorrelated tuples. Thus, the loss is given by

$$L(\theta_i) = \mathbb{E}\left[\left(r + \gamma \cdot \max_{a'} Q_{\theta'}(s', a') - Q_{\theta_i}(s, a)\right)^2\right], \quad (10)$$

where the expectation is taken over experiences $(s, a, r, s') \sim U(D)$. We generate our experience tuples by exploring the state space epsilon-greedily, i.e., during the Monte Carlo simulation we follow the policy that is implied by the current network weights with a chance of $(1-\epsilon)$ and otherwise choose a random action. We start with a high exploration coefficient $\epsilon = \epsilon_{\text{start}}$ and linearly decrease it, i.e., during some fraction of the total training iterations, ϵ is lowered until ϵ_{end} is reached. For the rest of the training, the exploration coefficient is constant. Common termination criteria for the learning process are fixing the number of episodes or using a threshold on the expected return achieved by the current policy.

III. Reinforcement Learning for Planning With Primitives

In order to solve the planning problem with RL, the MPA must be modeled as an MDP. Then, the planning can be solved by the DQN, in which primitives become the agent's actions. In this section, we describe how to model the integration of motion primitives into a DQN algorithm.

A. Modeling the Planning With MPA as an MDP

Recall the introduction of an MPA from Definition 5, which encodes the system's dynamical behavior, and of a *motion plan*, corresponding to alternating trims and maneuvers. Note that, within the MPA, there might exist multiple maneuvers outgoing from a common trim. However, there is a unique compatible successor trim following the choice of a maneuver (see Definition 4). Moreover, while the duration of a maneuver is fixed by definition, the trims can have arbitrary durations [1], [11]. It may be possible to let an RL algorithm choose the trims' duration, but this is not attempted in this work. Thus, from now on, we assume that all trims $\mathsf{q} \in \mathsf{Q}$ have fixed durations τ_{q}.

Corollary 1: Consider a finite set of trims Q *with fixed durations, together with a set of maneuvers* M *forming an MPA. Then, the planning problem becomes fully discrete. Moreover, we observe that* the planning problem boils down to finding a sequence of maneuvers *since a maneuver sequence implicitly defines the intermediate trims.*

This observation is key to formulating the Markov decision process, which will now be described step-by-step.

1) The MDP's States: Once we fix the duration of the trims within an MPA, for a given initial state $x_0 \in \mathsf{q}_0$, the set of reachable points, denoted by $\mathfrak{R}_{x_0}$, is discrete.

This defines the states $\mathcal{S}$ of the MDP as

$$\mathcal{S} = \mathfrak{R}_{x_0} \subset \mathfrak{M}. \quad (11)$$

2) The Action Space: Consider an MPA, with M being the set of maneuvers as presented in Definition 5. Then, we define the action space as

$$\mathcal{A} = \mathsf{M}.$$

However, given a current state $x \in \mathsf{q}$ for some trim q, there is typically only a subset of all maneuvers that could be selected for concatenation. So, we also define the trim-dependent *valid* action space.

Definition 8 (Valid Action Space): Let q *be a current trim primitive at the current step of the RL algorithm. We define the valid action space as*

$$\mathcal{A}_q^{\text{valid}} = \{\mathsf{m} \in \mathsf{M} \mid \exists\, \tilde{\mathsf{q}} \in \mathsf{Q},\ \text{s.t. } \mathsf{h}(\mathsf{q}, \mathsf{m}) = \tilde{\mathsf{q}}\} \subseteq \mathcal{A}, \quad (12)$$

with h being the MPA transition function as in Definition 5. Consequently, the number of valid actions might vary along a path of an MPA. This poses a challenge for the

application of DQL, since the Q-network requires a fixed output dimension. However, within DQL, there exists the concept of *invalid action masking* in order to overcome this issue [27], [28]. As we are using a value-based approach, for all invalid action $a \notin \mathcal{A}(s)$, we set $q_\pi(s,a) = -\infty$ and thus make sure that only valid actions are selected. Masking invalid actions has been shown to be crucial for the agent's performance [28], [29], [30].

3) The Transition Probability Function: The transition probability function $\mathcal{T}$ is defined as the probability distribution of switching to the next state, given an action $a \in \mathcal{A}^{\text{valid}}_{\mathsf{q}^-}$ applied to a current state $s \in \mathsf{q}^-$ with q^- denoting the current trim, and it is written as:

$$\mathcal{T}(s,a) := P(s'|s,a), \ \forall\, s' \in \mathcal{S}. \tag{13}$$

As idealized by [1] and followed by the relative literature, we use the distribution for a deterministic model. By choosing a valid maneuver as the action, we let the environment simulate not only this maneuver but also its unique successor trim in the same step. Thus,

$$P(s'|s,a=\mathsf{m}) = \begin{cases} 1, & \text{if } s' = \Psi_{\exp(\xi_{\mathsf{q}^+}\tau)g_\mathsf{m}}\big(x_\mathsf{m}(T)\big), \\ 0, & \text{otherwise,} \end{cases} \tag{14}$$

with g_m defined by

$$s = \Psi_{g_\mathsf{m}}\big(x_\mathsf{m}(0)\big), \tag{15}$$

where x_m is the (pre-computed/stored) trajectory of the maneuver $\mathsf{m} \in \mathsf{M}$,

$$\mathsf{m} : t \in [0, \mathrm{T}] \to (x_\mathsf{m}(t), u_\mathsf{m}(t)) \tag{16}$$

that corresponds to action $a \in \mathcal{A}^{\text{valid}}_{\mathsf{q}^-}$ and with the Lie algebra element ξ_{q^+} of the successor trim q^+.

Proposition 1: For $\mathcal{T}$ as defined above, the state s' corresponds to the endpoint of the simulation of maneuver m and its unique succeeding trim q^+.

Proof: Since m is valid for current state $s \in \mathcal{S}$, q^- is its preceding trim, with $s \in \mathsf{q}^-$. Therefore, u_m can be applied starting in s at $t = 0$, which would correspond to the controlled evolution $\varphi_{u_\mathsf{m}(t)}(s,t), \ t \in [0,T]$. However, since $x_\mathsf{m}(0)$ and s belong to the same trim, there also exists a $g_\mathsf{m} \in \mathcal{G}$ such that, alternatively, we can write $s = \Psi_{g_\mathsf{m}}\big(x_\mathsf{m}(0)\big)$ as required in (15), meaning g_m defines the symmetry shift for concatenating the precomputed maneuver trajectory. Let $\tilde{x}$ denote the endpoint of the applied (shifted) maneuver, so $\tilde{x} = \Psi_{g_\mathsf{m}}\big(x_\mathsf{m}(T)\big) = \varphi_{u_\mathsf{m}(T)}(s,T)$. Next, the succeeding trim q^+ with Lie algebra element ξ_{q^+} is applied for its defined duration τ to eventually end up in

$$\begin{aligned} x' &= \Psi_{\exp(\xi_{\mathsf{q}^+}\tau)}(\tilde{x}) \\ &= \Psi_{\exp(\xi_{\mathsf{q}^+}\tau)}(\Psi_{g_\mathsf{m}}\big(x_\mathsf{m}(T)\big)) \\ &= \Psi_{\exp(\xi_{\mathsf{q}^+}\tau)g_\mathsf{m}}\big(x_\mathsf{m}(T)\big) \\ &= s', \end{aligned}$$

as required for the transition function in (14). □

This means that the probability is one, if and only if s' equals the state after the maneuver p and its successor trim have been applied to s.

Remark 1: In a probabilistic motion planning setting, the transition probability function could consider stochastic disturbances or perturbations in the system, such that $P(s'|x,\mathsf{p}) > 0$ for s' in the vicinity of x'.

4) The Initial Distribution: The initial distribution μ can be selected as random, as it is usual when solving RL problems.

5) The Reward Function: The reward function $\mathcal{R}(s,a,s')$ is shaped to meet the planning problem's outcome: the solution time, the feasibility - i.e., arriving at a goal state while avoiding obstacles - or some other performance criterion to be optimized [8]. For instance, positive rewards could be given for s' reaching the goal, while the episode ends if there is a collision with an obstacle. A collision can be detected, at each episode, by verifying if the maneuver trajectory selected as the action a and its succeeding trim trajectory hit any obstacle.

B. A Unified RL Framework

Typically, RL methods rely either on a discrete action space or (exclusively) on a continuous one, with both having their advantages and disadvantages. By assuming the maneuver primitives as the discrete action space, we can reinterpret the actions as controlled continuous-time trajectories anytime, so that we take the best of both worlds.

The DQN algorithm only solves problems with discrete action spaces. Thus, usually, the actions may be defined by discretizing the control space and applying constant controls for the duration of a time step [31], [32]. As a drawback for this discretization, there is a trade-off between smoothness for the sequence of control inputs, by having enough discrete steps with the cost of a large discrete action space, and action selections becoming prohibitively expensive to evaluate [32]. As an alternative, other DRL algorithms can deal with continuous action space, such as the Deep Deterministic Policy Gradient (DDPG) [33], and the soft actor-critic [34]. However, controlling efficiently a continuous control space with DRL needs a sufficiently high amount of training time or depends on the quality of the demonstration's samples, if an imitation learning approach is chosen instead [35].

Alternatively, our method is based on discretization via motion primitives, which consist of a continuous sequence of control inputs and – via the dynamical system model and the symmetry exploitation – allow a reconstruction of continuous state trajectories at any time. Then, through the use of maneuvers, the RL action space is discrete but non-constant control laws can be applied, which allows the algorithm to produce a broader set of trajectories. Thus, our approach combines the flexibility of continuous actions with the problem abstraction given by (a sufficiently small amount of) discrete actions, making it possible to have solutions through DQN.

IV. Example: The Single-Track Vehicle Model

As an example to evaluate the proposed method, we chose to work with vehicle dynamics in an autonomous driving application. In the next subsections, we will, first, detail the DQL parameters. Then, we compare the goal reachability probability of the DQN to an A*-based graph search, showing that Q-learning for planning with MPA is a good alternative

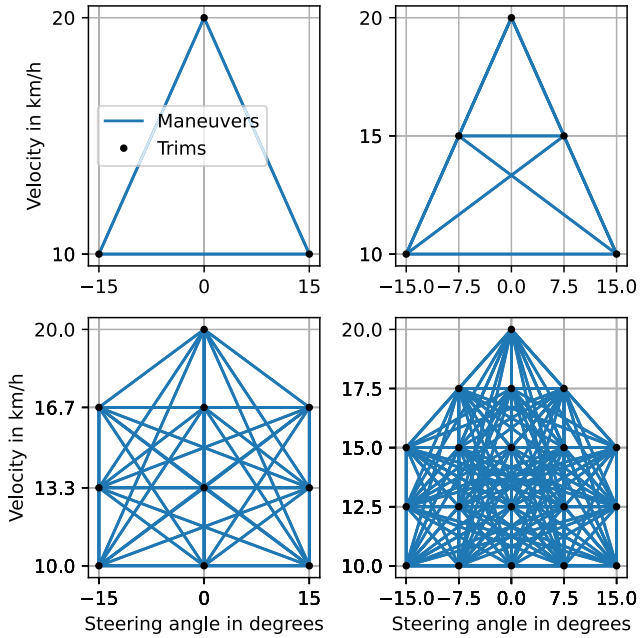

Fig. 3. The four motion primitive automata consisting of three, five, ten, and 19 trims, respectively, and connecting maneuvers according to the depicted network topology. Each blue edge corresponds to two maneuvers, one in each direction.

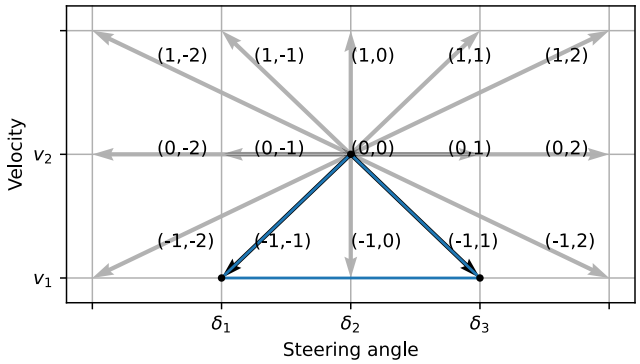

Fig. 4. The three-trim automaton as an extreme example of invalid actions: Among the fifteen potential actions $a \in \mathcal{A}$, only three are valid in the trim q defined by $v_2 = 20\,\mathrm{km\,h^{-1}}$ and $\delta_2 = 0°$, namely $(-1,-1)$, $(-1,1)$ which have a corresponding maneuver, and $(0,0)$, which corresponds to remaining in the same trim.

to the commonly used graph search method. Additionally, we will consider a particularly hard setting for the planning by changing the goal position. Lastly, different MPA sizes are compared, i.e., with different numbers of primitives.

A. Experimental Setup

The chosen vehicle dynamics is the single-track model [36]. This model describes the vehicle considering tire slip, i.e., understeer or oversteer effects when performing maneuvers on the edge of the physical limits. It is a more detailed model when compared with, for instance, the Reeds-Shepp's car [37] or the kinematic single-track model [36]. The single-track model is described in Appendix A.

The experiments were simulated in a version of the Cyber-Physical Mobility Lab, presented in [38], scaled up by a factor of 18 to obtain realistic dimensions for a real-world road network. The map can be seen in figures 5 and 6. We design two different validation scenarios that vary in the goal positions to be reached.

Our approach requires a constant duration of the trims to be chosen, which was set to 0.5 s. To obtain the maneuver trajectories, a cubic polynomial transition was applied for both velocity and steering angle, as it has previously been proposed in [11].

In order to study the influence of the number of available primitives, four different MPA were used. In Figure 3, their graphical representation is given. As it is derived in Appendix A, trims for the single-track vehicle model are simply characterized by the constant values (v, δ) for the steering angle and the velocity. Thus, we depict the MPA vertices on a two-dimensional grid and indicate the existence of (bidirectional) maneuvers by connecting edges.

Formally, each trim can be associated with two indices (i, j), corresponding to its constant velocity v_i, $i \in \{1, \ldots, n_v\}$ and its constant steering angle δ_j, $j \in \{1, \ldots, n_\delta\}$.

Further, a maneuver from trim (i, j) to trim (i', j') is represented by the tuple $(i' - i, j' - j)$. Assuming that the velocities are indexed in ascending order of their values, i.e. $v_1 < v_2 < \ldots < v_{n_v}$, and the same ordering applied to the steering angles, it ensures that the maneuver indices have a physical meaning. For example, all maneuvers represented by $(1, 0)$ correspond to an acceleration to the next velocity without a change in steering angle.

It is important to note that an arbitrary tuple $(\Delta i, \Delta j)$ with $\Delta i \in \{1 - n_v, \ldots n_v - 1\}$, $\Delta j \in \{1 - n_\delta, \ldots n_\delta - 1\}$ does not necessarily correspond to an existing maneuver. For instance, there cannot be an acceleration if the vehicle is currently in the fastest trim, as exemplified in Figure 4. Such invalid actions need to be masked out in the DQN algorithm, specifically in the rollout policy and in the calculation of the action value targets.

The reward function is given by $\mathcal{R}(s, a, s') = 100$, if s' is a goal state, and $\mathcal{R}(s, a, s') = 0$ otherwise. A state is considered to be in the goal if it lies within the circular region defined by $\|\begin{pmatrix} s_x & s_y \end{pmatrix}^\mathsf{T} - \begin{pmatrix} x_\mathrm{goal} & y_\mathrm{goal} \end{pmatrix}^\mathsf{T}\|_2 \leq r_\mathrm{goal} = 5\,\mathrm{m}$. Note that there are no negative rewards for collisions. They are unnecessary since an obtained optimal policy is guaranteed to also avoid crashes because they would end the episode before the goal could be reached. If the trajectory of the transition is colliding, i.e., at least part of the car leaves the road, the episode ends. The same happens if the maximum number of steps in an episode is exceeded or if the goal area is reached.

A DQN agent was trained for each combination of automaton and different scenarios, resulting in eight agents. The implementation of the DQN algorithm is taken from the python library *Stable Baselines 3* [39]. The hyperparameters are based on the values given by [40] for the *LunarLander-v2* environment, but some parameters were modified. They are given in Table I. For the Q-network, a multi-layer perceptron was used, containing two hidden layers of size 256. The achieved returns we report in the following were computed by using *deep statistical model checking* [41], [42] with the state-of-the-art statistical model checker MODES [43], [44],

TABLE I
DQN HYPERPARAMETERS

Parameter	Value
Batch size	128
Buffer size	500 000
Final exploration coefficient ϵ_{end}	0.01
Ratio of exploration phase to total training	0.5
Discount factor γ	0.9
Learning rate	0.000 63
Steps before learning begins	0
Target update interval	250
Training frequency	4

which is part of the MODEST TOOLSET [45], with a confidence of 95% that the error is at most 2 for the return, and at most 0.02 for the goal reachability probability.

We compared the performance of the DQN agents with an A*-based graph search algorithm described in Appendix B, which can be considered a simplified Π* algorithm presented in [11]. The heuristic was inflated by a factor such that we could compute (not necessarily optimal) solutions in the timeout of 10 s, avoiding searches that take an undetermined time to reach a solution. In the graph search algorithm, as in the DQN algorithm, the planning problem consists of finding a collision-free motion plan from a random starting pose to the goal area. Similarly to the proposed RL algorithm, the graph search is also set up to minimize the number of steps to the goal, where each step consists of one maneuver and its successor trim.[3]

We designed two test scenarios:

1) Scenario 1: a goal in the center of the map, as depicted in Figure 5;
2) Scenario 2: a goal that is placed in the upper center of the map, as shown in Figure 6.

Recall that we randomly choose initial positions for each episode. In the first scenario, the centered goal position can be reached from even the most distant initial positions by a path with one single turn only. The second one makes the agents' training more challenging, as for initial points that lie further apart from the goal, more complicated routes, with longer sequences of primitives, must be learned.

B. Results

In this subsection, we compare the performance of the DRL-trained policy with the one obtained by A*-planning. Depending on the vehicle's initial position, not every instance of the planning problem can be solved (due to, e.g., the MPA architecture and the features of the solvers). We evaluated the results by running simulations for 98% confidence interval of the return (7) and the goal reachability probability, i.e., how often the simulations were able to find the goal (in the total amount of experiments). The number of simulations run is presented in Table II.

Therefore, we measure the performance w.r.t. goal reachability probability. First, consider Figure 7, where we report the achieved goal reachability probability depending on the used number of trims. In Scenario 1, for the MPA with three trims, both approaches performed almost equally. In contrast,

[3]With the reward function being only a positive reward for reaching the goal and a discount factor below one, the RL algorithm optimizes for the minimal number of steps to the goal.

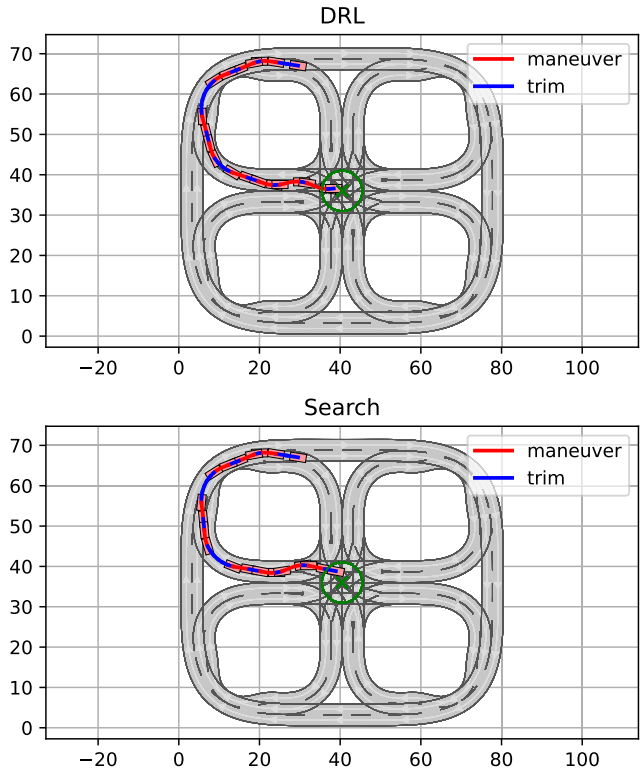

Fig. 5. Example solutions of the planning problem for Scenario 1 obtained by reinforcement learning (upper) and the search algorithm (lower).

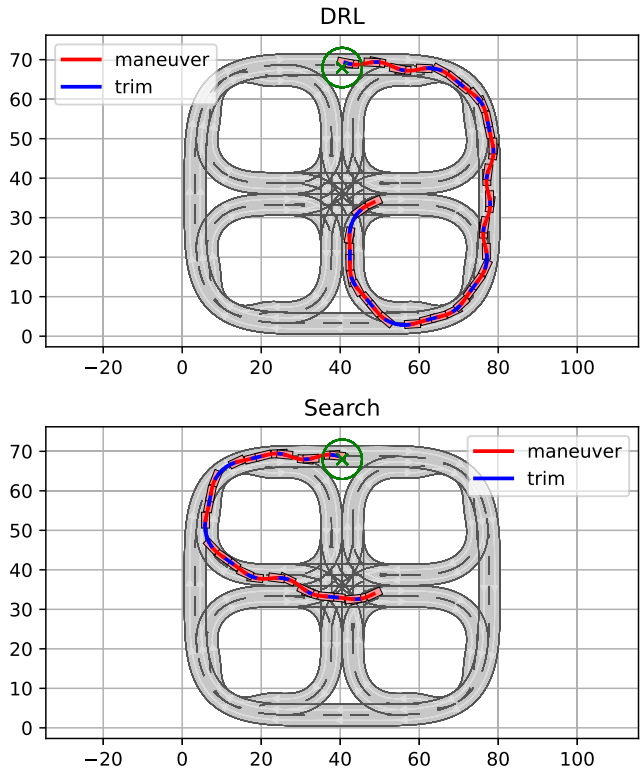

Fig. 6. Example solutions of the planning problem for Scenario 2 obtained by reinforcement learning (upper) and the search algorithm (lower).

TABLE II
NUMBER OF SIMULATIONS RUN FOR EACH SCENARIO AND MPA WITH 98% CONFIDENCE INTERVAL

Scenario	3 trims	5 trims	10 trims	19 trims
Scenario 1	970	890	790	820
Scenario 2	2250	1970	1560	1900

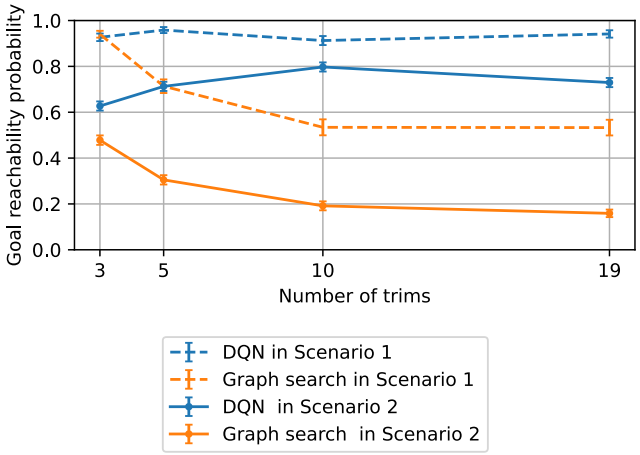

Fig. 7. Comparison of the goal reachability probabilities of DQN and the A*-search algorithm, for both scenarios.

for the MPA with five, ten, or 19 trims, A* regularly times out, i.e., it fails to find the goal in the maximum allowed time, while DQN's performance stays close to 100%. Similarly, for Scenario 2, DQN outperforms the search approach for the four motion primitive automata. The best result for DQN can be seen when using ten trims. The DQN for the MPA with 19 trims performs worse when compared to the smaller MPA. Presumably, DQN's performance suffers from the enlarged action space which is caused by the increased number of trims. In total, DQN is more likely to provide policies that reach the goal state than the A*-search does within the limited computation time.

Now, turning to Figure 8, we display the average number of steps of the trajectories from DQN and A* search, respectively, restricting to rollouts for which both approaches succeed in finding a solution. We observe that small automata, that provide few different trims only, lead to long sequences of alternating trim maneuver steps until reaching the goal. Switching from three to five, and subsequently to ten primitives, significantly reduces the needed steps to the goal, whereas solutions for ten and 19 trims behave similarly. This holds in both scenarios. This can be explained by the fact that larger automata provide a larger variety of trims and thus, are more likely to provide suitable short sequences for individual initial positions. As expected from the design of Scenario 1 versus Scenario 2, the centered goal position leads to shorter sequences. In all tests, the average steps of the DQN rollout is slightly below the average of the A* search solution steps.

Lastly, take into account Figure 9, which compares the average time to perform the search for the A* and the trained agent. To have a fair comparison, as in Figure 8, we consider just graph search simulations that were successful, i.e., reached the goal. The execution time of A* graph is, in general,

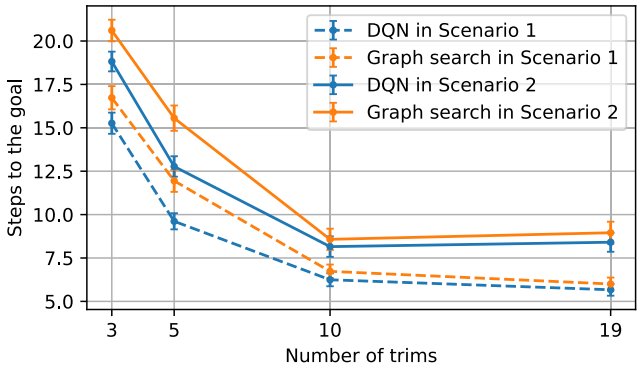

Fig. 8. Comparison between the average number of steps taken until reaching the goal by DQN and the search algorithm, for the two scenarios. Here, only rollouts that succeeded in reaching the goal in both DQN and search algorithms were considered to avoid the bias created by the fact that the search mostly solved the rollouts with the start close to the goal.

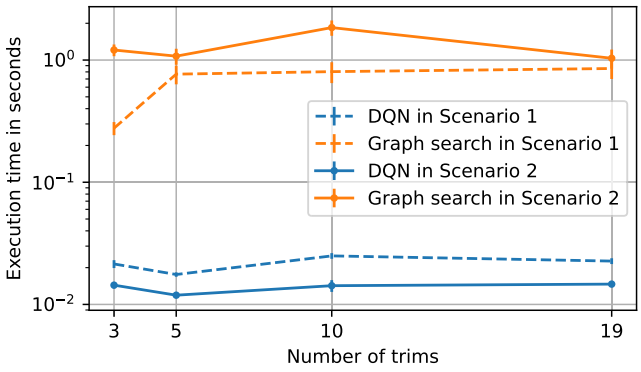

Fig. 9. Comparison of the average execution times of both algorithms, where only rollouts that reached the goal were considered.

sensitive to the test scenario and to the planning algorithm with its parameters, as previously observed and discussed in [2]. Thus, it was a priori expected to see differences compared to DQN, since this approach shifts the computationally heavy training phase to before the operation. Then, the agent could perform the planning on average, considering both scenarios, in 17.7 ms (st. dev. 4.7 ms, max. 25.0 ms, min. 11.9 ms). In comparison, the average time for the graph search was 0.98 s (st. dev. 0.45 s, max. 1.84 s, min. 0.28 s).

C. Discussion

To handle the non-constant action space, which occurs due to the varying number of valid maneuvers for the current trim, validity of maneuvers depending on the current trim, invalid action masking needs to be implemented in the DQN algorithm. This is not supported by many publicly available implementations such as the one provided by [39]. Therefore, the algorithm needed to be extended to take into account invalid actions, in particular in the target calculation of the Q-network (8), as well as in the rollout policy, both random and Q-network policy (lines 7 and 15 of Algorithm 1).

During the numerical experiments, it became apparent that the algorithm is very sensitive to changes in the hyperparameters. Interestingly, the discount factor also played an important role. A high value for this parameter, e.g. $\gamma = 0.99$, caused an unexpected phenomenon: the agent avoided reaching the

goal in order to keep the episode going for a longer time. This was likely caused by an approximation error of the action value, i.e., the return for reaching the goal directly was erroneously estimated to be lower than for reaching it later. We leave a methodological analysis of parameter sensitivity for further investigation, as it is not in the scope of this paper. Furthermore, it is noteworthy that getting the DQN algorithm to succeed in the Senario 2 was only possible once good hyperparameters in Scenario 1 had been found and transferred to Scenario 2.

The primitives-DQL can reach the goal more often than the state-of-the-art graph search method due to the computation time. As it is generally known, graph search methods may suffer from the curse of dimensionality. In our study, this was most likely the reason why the A* search has run into a computation timeout (of 10 s), therefore failing in reaching the goal. For both scenarios, independently from the MPA, Figure 8 indicates that DQN is acting close to the graph search. While a plain A* search would find optimal solutions, in our implementation containing the inflation factor we included for computational speed-up, this cannot be guaranteed.[4] However, the results show that not only is DQN able to reach the goal more often, but also indicates that it possibly does with almost as few steps as possible.

Lastly, we would like to emphasize that we had carefully chosen the road layout for the training and tests of the two approaches. The layout (cf. figures 5 and 6) represents typical urban types of urban street networks: cross intersections (four legs), T-intersections (three legs), and roundabouts [46]. Thus, the map is suitable for a generalization at a certain range. However, [46] describes a large variety of those street networks, as different angles of street intersections, for example, exist. Therefore, retraining our RL agents for different street layouts and routes will be necessary to apply our proposed method to a real-world scenario.

In summary, while the DQN approach is not guaranteed to provide optimal solutions, in our validation scenarios it produced robust results that outperforms the A* search in both computation time and reliability.

V. Conclusion and Future Works

In contrast to typical RL methods with discrete actions, where the control inputs are discretized, our method is based on discretization via motion primitives. These "building blocks" may consist of continuous sequences of control inputs. Thus, while we choose a finite representation of motion primitives to obtain a discrete action space, each action represents a segment of a trajectory with continuous control inputs. This allows the algorithm to produce a set of continuous trajectories with a discrete action space. We solved the planning problem via the DQN algorithm and compared it with the state-of-the-art Π* algorithm, an A*-based graph search.

We investigated four motion primitive automata with different sizes in two case scenarios. The presented results showed that, despite not having any optimal guarantees, the DQL provided solutions that outperformed the graph search in the probability of reaching the goal. Moreover, the graph search was performed with an inflation factor for the heuristic function for faster computation times, but the algorithm was not real-time capable, in contrast to those obtained from the DQL. As a further advantage of the RL algorithm, we observed that the solution times are practically invariant w.r.t. the size of the automaton.

Future works could perform a profound sensitivity analysis to better understand the parameter tuning. Also, scenarios with multiple vehicles, in which collision avoidance is crucial, should be studied. Finally, to validate the application of this methodology, a study can be conducted on real-world traffic scenarios, by incorporating real-world data and validating the approach in field tests with obstacles or unpredictable behaviors.

[4]Without the inflation factor, the computation times would increase, and very likely more simulations would be limited by the timeout parameter, being discarded.

Appendix

A. The Single-Track Vehicle Model

In this work, the vehicle dynamics are modeled by the single-track model, specifically, the version and parameters provided by the CommonRoad project [36], [47]. The state for this model is:

$$x = \begin{pmatrix} s_x & s_y & \psi & \dot{\psi} & v & \delta & \beta \end{pmatrix}^{\mathsf{T}}, \tag{17}$$

where $\begin{pmatrix} s_x & s_y \end{pmatrix}^{\mathsf{T}} \in \mathbb{R}^2$ are the positions of the center of gravity in the Cartesian plane, $\psi \in S^1$ is the vehicle orientation (where S^1 refers to the unit circle), $v \in \mathbb{R}$ is the longitudinal velocity, $\delta \in S^1$ is the steering angle and $\beta \in S^1$ is the slip angle. The control input is $u = \begin{pmatrix} u_{\dot{v}} & u_{\dot{\delta}} \end{pmatrix}^{\mathsf{T}} \in \mathbb{R}^2$, with $u_{\dot{v}}$ as the longitudinal acceleration and $u_{\dot{\delta}}$ as the velocity of the steering angle.

The equations that describe the single-track model are given by:

$$\begin{cases} \dot{s}_x(t) = v(t) \cdot \cos(\psi(t) + \beta(t)), \\ \dot{s}_y(t) = v(t) \cdot \sin(\psi(t) + \beta(t)), \\ \dot{\psi}(t) = \omega(t), \\ \dot{\omega}(t) = \dfrac{\mu M}{I_z L}\Big(l_{\mathrm{f}} \cdot \alpha_1 \cdot \delta(t) + (l_{\mathrm{r}} \cdot \alpha_2 - l_{\mathrm{f}} \cdot \alpha_1)\beta(t) \\ \qquad\qquad - (l_{\mathrm{f}}^2 \cdot \alpha_1 + l_{\mathrm{r}}^2 \cdot \alpha_2)\dfrac{\dot{\psi}(t)}{v(t)} \Big), \\ \dot{v}(t) = u_{\dot{v}}(t), \\ \dot{\delta}(t) = u_{\dot{\delta}}(t), \\ \dot{\beta}(t) = \dfrac{\mu}{L \cdot v(t)}\Big(\alpha_1 \cdot \delta(t) - (\alpha_2 + \alpha_1)\beta(t) \\ \qquad\qquad + (l_{\mathrm{r}} \cdot \alpha_2 - l_{\mathrm{f}} \cdot \alpha_1)\dfrac{\dot{\psi}(t)}{v(t)} \Big) - \dot{\psi}(t), \end{cases} \tag{18}$$

where

$$\begin{aligned} \alpha_1 &:= \alpha_1\big(u_{\dot{\delta}}(t)\big) = C_{\mathrm{f}}\big(g \cdot l_{\mathrm{r}} - h \cdot u_{\dot{\delta}}(t)\big), \\ \alpha_2 &:= \alpha_2\big(u_{\dot{\delta}}(t)\big) = C_{\mathrm{r}}\big(g \cdot l_{\mathrm{f}} - h \cdot u_{\dot{\delta}}(t)\big), \\ L &= l_{\mathrm{f}} + l_{\mathrm{r}}, \end{aligned} \tag{19}$$

for the parameters given in Table III.

TABLE III
SINGLE-TRACK MODEL'S PARAMETERS, TAKEN FROM [47]

Parameter	Symbol	Value
Distance from CG[#] to front axle	l_f	$0.883\,\mathrm{m}$
Distance from CG[#] to rear axle	l_r	$1.508\,\mathrm{m}$
Total vehicle mass	M	$1225\,\mathrm{kg}$
Moment of inertia about z axis	I_z	$1538\,\mathrm{kg\,m^2}$
Center of gravity height of M	h	$0.557\,\mathrm{m}$
Cornering stiffness coeff. (front, rear)	C_f, C_r	$20.89\,\mathrm{rad^{-1}}$
Friction coefficient	μ	1.048
Maximum steering velocity	$\dot{\delta}_\mathrm{max}$	$0.4\,\mathrm{rad\,s^{-1}}$
Switching velocity	v_s	$4.755\,\mathrm{m\,s^{-1}}$
Maximum acceleration	a_max	$11.5\,\mathrm{m\,s^{-2}}$

[#]CG stands for center of gravity.

Consider a rotation of an angle $\Delta\psi$ and a translation of Δs_x and Δs_y. The symmetry group for (18) can be given by

$$\mathcal{G} := \left\{ g \in \mathsf{SE}(7) : g = \begin{pmatrix} R & \Delta x \\ 0 & 1 \end{pmatrix} \right\}, \tag{20}$$

where

$$R = \begin{pmatrix} R_{\mathsf{SO}(3)} & 0 \\ 0 & I \end{pmatrix} \in \mathsf{SO}(7),$$
$$R_{\mathsf{SO}(3)} = \begin{pmatrix} \cos(\Delta\psi) & -\sin(\Delta\psi) & 0 \\ \sin(\Delta\psi) & \cos(\Delta\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \in \mathsf{SO}(3),$$
$$\Delta x = \begin{pmatrix} \Delta s_x & \Delta s_y & \Delta\psi & 0 \end{pmatrix}^\mathsf{T} \in \mathbb{R}^2 \times S^1 \times \{0\}^4,$$

where I and 0 are, respectively, the identity and a vector of zeros with appropriate dimensions. The proof can be found in [2].

The model's trim primitives are characterized by zero control inputs, a constant velocity v and constant steering angle δ. In these trajectories the curve described by the vehicle's center of mass (s_x, s_y) is a circular arc (or a straight line in the case of $\delta = 0$), while the yaw angle ψ changes at a constant rate and the other state variables, i.e., $\dot{\psi}$, v, δ and β, are constant [2].

B. A*-Based Search Algorithm

Various methods exist to solve motion planning problems. In previous work [11], we have presented a primitives-based search algorithm Π*, which is also capable of varying the coasting times of the trim primitives to reach exact goal positions. In this work, the achievable motion plans should be equal for both algorithms, for the sake of comparison. Therefore, here, we fix the coasting times for the search algorithm such that no continuous optimization is carried out.

This search algorithm finds the optimal solution to every feasible problem if an admissible heuristic function is provided. The heuristic function estimates the cost from the current node to the goal and it is called admissible if it never underestimates the actual cost, i.e., it provides a lower bound. To accelerate the exploration of nodes an inflated heuristic can be used, which is obtained by multiplying an admissible heuristic with an inflation factor $\eta > 1$. While the Π* algorithm would allow dynamic adjustments of η, we use a constant factor of $\eta = 3.5$ here.

Algorithm 2 Search Algorithm

Data: maneuver automaton, coasting time τ, initial trim, initial pose, goal region, heuristic function
Result: goal node with motion plan

```
start_node.state ← initial state;
start_node.motion_plan ← (·, (0));
start_node.g ← 0;
start_node.h ← h(start_node);
open_list ← {start_node};
while open_list not empty do
    node ← element with lowest f (= g + h) value in open_list;
    open_list ← open_list \ {node};
    if node in goal then
        return node;
    end
    open_list ← open_list ∪ expand_node(node);
end
Function expand_node(node)
    new_nodes ← ∅;
    foreach maneuver ∈ node.final_trim.successor_maneuvers do
        state ← maneuver.calc_end_state(initial_state = node.state);
        new_node.state ← maneuver.successor_trim.calc_flow(state, τ);
        new_node.motion_plan.maneuver_sequence ← new_node.motion_plan.maneuver_sequence ∪ (maneuver);
        new_node.motion_plan.coasting_times ← new_node.motion_plan.coasting_times ∪ (τ);
        if new_node is colliding then
            continue;
        end
        new_node.g ← node.g + cost(maneuver) + cost(maneuver.successor_trim);
        new_node.h ← h(new_node);
        new_nodes ← new_nodes ∪ {new_node};
    end
    return new_nodes;
end
```

The heuristic used here is given by $h = \eta h_0$ with the uninflated heuristic being

$$h_0 = \max\left(0, \frac{\| \begin{pmatrix} s_x & s_y \end{pmatrix}^\mathsf{T} - \begin{pmatrix} x_\mathrm{goal} & y_\mathrm{goal} \end{pmatrix}^\mathsf{T} \|_2 - r_\mathrm{goal}}{d_\mathrm{max}}\right), \tag{21}$$

where d_max is the maximum distance that can be covered in a single step (by the fastest maneuver with its successor trim). The pseudocode of the graph search, as used in this work, is given in Algorithm 2.

ACRONYMS

Π*	Optimized Primitives
DQL	deep Q-learning

DQN deep Q-network
DRL deep reinforcement learning
MDP Markov decision process
MPA motion primitive automaton
NN neural network
RL reinforcement learning

SYMBOLS

$:=$ Equality relationship, which is true by definition
x Dynamical system's state
u Dynamical system's input
f Dynamical system's vector field
φ Dynamical system's flow
t Time
$\mathfrak{M}$ Manifold of state's domain: $x \in \mathfrak{M}$
$\mathcal{U}$ Set of input's domain: $u \in \mathcal{U}$
T Time interval
$\mathcal{G}$ Lie group
Ψ Left action of the Lie group on a state manifold
p Motion primitive
$\mathfrak{g}$ Lie algebra of the Lie group $\mathcal{G}$
Q MPA's trim alphabet
M MPA's maneuver alphabet
h MPA's transition function
q_0 MPA's initial trim
Q_f MPA's set of final trims
$\mathcal{D}(S)$ The set of probability distributions over a non-empty set S
$\mathcal{M}$ Markov decision process (MDP)
$\mathcal{S}$ A finite set of MDP states
$\mathcal{A}$ A finite set of MDP actions
$\mathcal{T}$ A partial transition probability function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightharpoonup \mathcal{D}(\mathcal{S})$
μ An initial distribution $\mu \in \mathcal{D}(\mathcal{S})$
$\mathcal{R}$ Reward function $\mathcal{R} : \mathcal{S} \times A \times \mathcal{S} \to \mathbb{R}$
π Action policy
$\mathtt{x}$ DQN's observation data
$\mathtt{y}$ DQN's target value
ϕ DQN's preprocessed sequence
$P(A|B)$ Probability of A given B

REFERENCES

[1] E. Frazz, M. A. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE Trans. Robot.*, vol. 21, no. 6, pp. 1077–1091, Dec. 2005.

[2] M. V. A. Pedrosa, T. Schneider, and K. Flaßkamp, "Learning motion primitives automata for autonomous driving applications," *Math. Comput. Appl.*, vol. 27, no. 4, p. 54, Jun. 2022.

[3] E. Frazzoli, M. A. Dahleh, and E. Feron, "Robust hybrid control for autonomous vecontrol hicle motion planning," in *Proc. 39th IEEE Conf. Decis. Control*, vol. 1, Jul. 2000, pp. 821–826.

[4] K. Flaßkamp, S. Ober-Blöbaum, and K. Worthmann, "Symmetry and motion primitives in model predictive control," *Math. Control, Signals, Syst.*, vol. 31, no. 4, pp. 455–485, Dec. 2019.

[5] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed., Upper Saddle River, NJ, USA: Prentice-Hall, 2010.

[6] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," *Ann Arbor*, vol. 1001, no. 48105, pp. 18–80, 2008.

[7] J. Petereit, T. Emter, C. W. Frey, T. Kopfstedt, and A. Beutel, "Application of hybrid A* to an autonomous mobile robot for path planning in unstructured outdoor environments," in *Proc. ROBOTIK, 7th German Conf. Robot.*, May 2012, pp. 1–6.

[8] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[9] N. Richards, M. Sharma, and D. Ward, "A hybrid A*/automaton approach to on-line path planning with obstacle avoidance," in *Proc. AIAA 1st Intell. Syst. Tech. Conf.*, Sep. 2004, p. 6229.

[10] L. Lüttgens, B. Jurgelucks, H. Wernsing, S. Roy, C. Büskens, and K. Flaßkamp, "Autonomous navigation of ships by combining optimal trajectory planning with informed graph search," *Math. Comput. Model. Dyn. Syst.*, vol. 28, no. 1, pp. 1–27, 2022.

[11] M. V. A. Pedrosa, T. Schneider, and K. Flaßkamp, "Graph-based motion planning with primitives in a continuous state space search," in *Proc. 6th Int. Conf. Mech. Eng. Robot. Res. (ICMERR)*, Dec. 2021, pp. 30–39.

[12] P. Scheffe, M. V. A. Pedrosa, K. Flaßkamp, and B. Alrifaee, "Receding horizon control using graph search for multi-agent trajectory planning," *IEEE Trans. Control Syst. Technol.*, vol. 31, no. 3, pp. 1092–1105, May 2023.

[13] C. Baier et al., "Lab conditions for research on explainable automated decisions," in *Proc. 1st Int. Workshop*, 2020, p. 83.

[14] T. P. Gros, D. Höller, J. Hoffmann, and V. Wolf, "Tracking the race between deep reinforcement learning and imitation learning," in *Proc. 17th Int. Conf.*, 2020, pp. 11–17.

[15] T. P. Gros, D. Höller, J. Hoffmann, and V. Wolf, "Tracking the race between deep reinforcement learning and imitation learning—Extended version," 2020, *arXiv:2008.00766*.

[16] T. P. Gros, "Tracking the race: Analyzing racetrack agents trained with imitation learning and deep reinforcement learning," M.S. thesis, Dept. Comput. Sci., Saarland Univ., Saarbrucken, Germany, 2021.

[17] T. P. Gros, D. Höller, J. Hoffmann, M. Klauck, H. Meerkamp, and V. Wolf, "Dsmc evaluation stages: Fostering robust and safe behavior in deep reinforcement learning," in *Proc. 18th Int. Conf.*, 2021, pp. 197–216.

[18] T. P. Gros, H. Hermanns, J. Hoffmann, M. Klauck, M. A. Köhl, and V. Wolf, "Mogym: Using formal models for training and verifying decision-making agents," in *Proc. 34th Int. Conf*, 2022, pp. 430–443.

[19] T. P. Gros et al., "DSMC evaluation stages: Fostering robust and safe behavior in deep reinforcement learning—Extended version," *ACM Trans. Model. Comput. Simul.*, vol. 33, no. 4, pp. 1–28, Oct. 2023.

[20] K. Flaßkamp, S. Ober-Blöbaum, and M. Kobilarov, "Solving optimal control problems by exploiting inherent dynamical systems structures," *J. Nonlinear Sci.*, vol. 22, no. 4, pp. 599–629, Aug. 2012.

[21] K. Flaßkamp, S. Ober-Blöbaum, and S. Peitz, "Symmetry in optimal control: A multiobjective model predictive control approach," in *Studies in Systems, Decision and Control*. Cham, Switzerland: Springer, 2020, pp. 209–237.

[22] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ, USA: Wiley, 1994.

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

[24] A. G. Barto, S. J. Bradtke, and S. P. Singh, "Learning to act using real-time dynamic programming," *Artif. Intell.*, vol. 72, nos. 1–2, pp. 81–138, Jan. 1995.

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

[26] V. Mnih et al., "Playing Atari with deep reinforcement learning," *arXiv:1312.5602*, 2013.

[27] O. Vinyals et al., "StarCraft II: A new challenge for reinforcement learning," 2017, *arXiv:1708.04782*.

[28] A. M. Ali and L. Tirel, "Action masked deep reinforcement learning for controlling industrial assembly lines," in *Proc. IEEE World AI IoT Congr. (AIIoT)*, Jun. 2023, pp. 0797–0803.

[29] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," 2020, *arXiv:2006.14171*.

[30] A. Kanervisto, C. Scheller, and V. Hautamäki, "Action space shaping in deep reinforcement learning," in *Proc. IEEE Conf. Games (CoG)*, Sep. 2020, pp. 479–486.

[31] P. Wolf et al., "Learning how to drive in a real world simulation with deep Q-networks," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2017, pp. 244–250.

[32] B. R. Kiran et al., "Deep reinforcement learning for autonomous driving: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 6, pp. 4909–4926, Jun. 2022.

[33] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.

[34] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.

[35] G. Grandesso, E. Alboni, G. P. R. Papini, P. M. Wensing, and A. D. Prete, "CACTO: Continuous actor-critic with trajectory optimization—Towards global optimality," *IEEE Robot. Autom. Lett.*, vol. 8, no. 6, pp. 3318–3325, Jun. 2023.

[36] M. Althoff, M. Koschi, and S. Manzinger, "CommonRoad: Composable benchmarks for motion planning on roads," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2017, pp. 719–726.

[37] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific J. Math.*, vol. 145, no. 2, pp. 367–393, Oct. 1990, doi: 10.2140/pjm.1990.145.367.

[38] M. Kloock et al., "Cyber-physical mobility lab: An open-source platform for networked and autonomous vehicles," in *Proc. Eur. Control Conf. (ECC)*, 2021, pp. 1937–1944.

[39] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *J. Mach. Learn. Res.*, vol. 22, no. 268, pp. 1–8, 2021.

[40] A. Raffin. (2020). *RL Baselines3 Zoo*. [Online]. Available: https://github.com/DLR-RM/rl-baselines3-zoo

[41] T. P. Gros, H. Hermanns, J. Hoffmann, M. Klauck, and M. Steinmetz, "Deep statistical model checking," in *Proc. 15th Int. Federated Conf. Distrib. Comput. Tech.*, 2020, pp. 96–114.

[42] T. P. Gros, H. Hermanns, J. Hoffmann, M. Klauck, and M. Steinmetz, "Analyzing neural network behavior through deep statistical model checking," *Int. J. Softw. Tools Technol. Transf.*, vol. 25, no. 3, pp. 407–426, Jun. 2023.

[43] J. Bogdoll, L. M. Ferrer Fioriti, A. Hartmanns, and H. Hermanns, "Partial order methods for statistical model checking and simulation," in *Formal Techniques for Distributed Systems*. Berlin, Heidelberg: Springer, 2011, pp. 59–74.

[44] C. E. Budde, P. R. D'Argenio, A. Hartmanns, and S. Sedwards, "A statistical model checker for nondeterminism and rare events," in *Tools and Algorithms for the Construction and Analysis of Systems*. Cham, Switzerland: Springer, 2018, pp. 340–358.

[45] A. Hartmanns and H. Hermanns, "The modest toolset: An integrated environment for quantitative modelling and verification," in *Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer, 2014, pp. 593–598.

[46] K. Fitzpatrick, M. D. Wooldridge, and J. D. Blaschke, "Urban intersection design guide: Volume 1—Guidelines," Texas A&M Transp. Inst., TX, USA, Tech. Rep. FHWA/TX-05/0-4365-P2, 2005, vol. 1.

[47] M. Althoff. (2020). *Commonroad: Vehicle Models*. Accessed: May 29, 2023. [Online]. Available: https://commonroad.in.tum.de/tools/model-cost-functions

Tristan Schneider received the bachelor's degree in systems engineering in 2022. He is currently pursuing the master's degree in mechatronics and information technology, specializing in control engineering with Karlsruhe Institute of Technology (KIT). As a Student Research Assistant (2020–2023) with the Chair of Systems Modeling and Simulation, Saarland University, his work focused on autonomous vehicle modeling and motion planning. His bachelor's thesis compared graph search and reinforcement learning for motion planning using motion primitives.

Matheus V. A. Pedrosa received the bachelor's degree in computer engineering from the Federal University of Rio Grande do Norte, Natal, Brazil, in 2016, and the master's degree in automation and systems engineering from the Federal University of Santa Catarina, Florianópolis, Brazil, in 2018. He is currently pursuing the Ph.D. degree with the Chair of Systems Modeling and Simulation, Saarland University. His research interests include optimal control for nonlinear systems with motion primitives.

Timo P. Gros is currently pursuing the Ph.D. degree in computer science with the Chair of Modeling and Simulation and the Foundations of Artificial Intelligence Group, Saarland University. He is currently with German Research Center for Artificial Intelligence (DFKI) and is involved with the Centre for European Research in Trusted AI (CERTAIN). His research interests are in reinforcement learning, in particular its connections to other fields of research, such as planning, verification, and optimization.

Verena Wolf received the Ph.D. degree in computer science in 2008. She is currently the Scientific Director of German Research Center for Artificial Intelligence (DFKI). She has been a Full Professor with the Computer Science Department, Saarland University, since 2012. Her research interests include hybrid AI approaches that combine traditional mechanistic modeling and deep learning methods.

Kathrin Flaßkamp received the Diploma degree in technomathematik (applied mathematics with engineering) and the Ph.D. (Dr.rer.nat.) degree in mathematics from Paderborn University, Paderborn, Germany, in 2008 and 2013, respectively. She is currently a Full Professor in systems modeling and simulation with Saarland University, Saarbrücken, Germany. Her research interests include the field of modeling, simulation, optimization, and control, focusing on the development of numerical methods and on applications.