Learning to Drive Safely with Hybrid Options

Bram De Cooman(✉), Johan Suykens

Abstract. *Out of the many deep reinforcement learning approaches for autonomous driving, only few make use of the options (or skills) framework. That is surprising, as this framework is naturally suited for hierarchical control applications in general, and autonomous driving tasks in specific. Therefore, in this work the options framework is applied and tailored to autonomous driving tasks on highways. More specifically, we define dedicated options for longitudinal and lateral manoeuvres with embedded safety and comfort constraints. This way, prior domain knowledge can be incorporated into the learning process and the learned driving behaviour can be constrained more easily. We propose several setups for hierarchical control with options and derive practical algorithms following state-of-the-art reinforcement learning techniques. By separately selecting actions for longitudinal and lateral control, the introduced policies over combined and hybrid options obtain the same expressiveness and flexibility that human drivers have, while being easier to interpret than classical policies over continuous actions. Of all the investigated approaches, these flexible policies over hybrid options perform the best under varying traffic conditions, outperforming the baseline policies over actions.*

Keywords. *Reinforcement Learning, Autonomous Driving, Options, Skills, Hierarchical Control, Hybrid Actions*

1 Introduction

Due to its great potential to solve complex decision-making problems, Reinforcement Learning (RL) is a promising technique to train virtual drivers for autonomous driving applications [3, 13, 18, 21, 26, 42, 45]. An important design decision is the definition of the action space, defining how the virtual driver (agent) interacts with the environment. Most attempts so far have restricted their action spaces to be either fully discrete (with actions such as 'lane change left' and 'brake') [2, 25] or fully continuous (with actions such as 'turn the steering wheel by $x°$' or 'accelerate by $y\,\mathrm{m/s^2}$') [14, 34]. In this paper, we investigate some alternative approaches with options. The resulting hierarchical control architectures using combined or hybrid options have several benefits over the fully discrete or continuous approaches.

A classical RL setup with a discrete action space requires each action to have the same duration, corresponding to exactly one timestep in the underlying Markov Decision Process (MDP). An MDP with options, on the other hand, inherently supports different durations for each option. This makes the options approach especially useful for autonomous driving applications, in which a virtual driver needs access to both fast acceleration manoeuvres and slower lane change manoeuvres.

A drawback of purely discrete action spaces is that they lack flexibility and expressiveness, as, for example, the velocity of the ego vehicle can only be changed in fixed increments. This can be resolved using continuous action spaces. However, the extra flexibility of continuous actions also comes at a cost. Typically, the training time significantly increases and it becomes much harder to constrain the learned policies. Extra penalties could be introduced in the reward signal to try and constrain the vehicle's driving behaviour. However, this approach becomes increasingly more complex as the number of constraints grows. Moreover, determining good penalties for manoeuvres that take multiple timesteps to complete, is not a straightforward task either. In contrast, under the introduced options framework, we have the ability to constrain vehicle movements more reliably using dedicated options with embedded safety and comfort constraints for certain manoeuvres.

The proposed framework for hierarchical control with options has several other benefits. It has the ability to flexibly combine dedicated options for longitudinal and lateral control, for which already existing driving assistance systems (such as cruise control or lane assist) can be reused. This can accelerate the learning process, as individual driving manoeuvres no longer have to be learned from scratch. Additionally, it can improve the trust of the general public in the learned driving policies. In particular, the vehicle will never perform any other (weird) manoeuvres than the predefined ones, which can be based on automated features that are already widely used and trusted by human drivers. Finally, the obtained models are also more interpretable than their continuous counterparts, as it is immediately clear what kind of manoeuvre the ego vehicle is performing by looking at which option is active. Ultimately, the proposed hybrid policies with options share the same expressiveness and flexibility as human drivers, effectively combining the best of discrete and continuous approaches.

Contributions

There are two key contributions in this work. First, we propose a set of options and associated manoeuvres tailored to autonomous driving tasks. We show how safety measures applied to policies over continuous actions can be reapplied to the hierarchical control setup with options, resulting in safe and comfortable driving behaviour by design. Second, we introduce two novel hierarchical control architectures with combined options and hybrid options. Both support

B. De Cooman (✉ bram.decooman@esat.kuleuven.be) and J. Suykens (✉ johan.suykens@esat.kuleuven.be) are with STADIUS Center for Dynamical Systems, Signal Processing and Data Analytics, Department of Electrical Engineering (ESAT), KU Leuven, 3001 Leuven, Belgium.

the separate selection of actions for longitudinal and lateral control, leading to more flexible driving policies.

Organization

This paper is further organized as follows. Section 2 gives an introduction to reinforcement learning, the options framework, and the methods used throughout this work. The autonomous driving setup and simulation environment are briefly reviewed in Section 3. Afterwards, in Section 4 the practical algorithms for hierarchical control with options tailored to the autonomous driving task are introduced. The proposed methodology is evaluated and compared with other approaches in Section 5. Finally, the related work is discussed in Section 6 before the conclusion.

2 Preliminaries

The basic concepts of reinforcement learning and the options framework are briefly introduced here. For a more detailed overview, the reader is referred to Sutton et al. [35, 36].

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a machine learning technique that can be applied to solve sequential decision-making problems, in which an agent interacts with an environment.

Markov Decision Process

Formally, the environment can be described as a Markov Decision Process (MDP) [32] $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \iota, \tau, r, \gamma)$ consisting of state space $\mathcal{S}$, action space $\mathcal{A}$, initial state distribution ι, state transition distribution τ, reward function r and discount factor γ. At every timestep t, the agent is able to observe the environment's current state $\boldsymbol{s}_t \in \mathcal{S}$ and perform a certain action $\boldsymbol{a}_t \in \mathcal{A}$. This brings the environment to a new state $\boldsymbol{s}_{t+1}$ in the following timestep, according to the stochastic environment dynamics $\tau(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{a}_t)$. Simultaneously, the agent receives a scalar reward signal $r_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$, indicating how favorable the taken action and resulting state transition are. This process of observing a state (and reward) and executing an action is continuously repeated throughout an episode, and is started by sampling an initial state from $\iota(\boldsymbol{s}_0)$.

Policy and Objective

The agent's policy $\pi(\boldsymbol{a}_t \mid \boldsymbol{s}_t)$ describes the probability (density) of taking action $\boldsymbol{a}_t$ after perceiving $\boldsymbol{s}_t$. This policy can be improved by taking feedback from the reward signal into account. More specifically, the goal in reinforcement learning is to find the optimal policy π^*, maximizing the expected discounted return

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\iota,\pi,\tau}\left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})\right]. \tag{1}$$

The expectation is taken over $\boldsymbol{s}_0 \sim \iota(\cdot)$, $\boldsymbol{a}_t \sim \pi(\cdot \mid \boldsymbol{s}_t)$ and $\boldsymbol{s}_{t+1} \sim \tau(\cdot \mid \boldsymbol{s}_t, \boldsymbol{a}_t)$ for all timesteps. The discount factor $\gamma \in [0\,, 1)$ ensures the discounted return objective remains finite for bounded reward functions and trades off the importance of future versus immediate rewards. For deterministic policies, which we primarily work with in this paper, the notation $\boldsymbol{a} = \pi(\boldsymbol{s})$ is used.

Value Function

It is often helpful to consider the value functions of a certain policy π, defined as $v^\pi(\boldsymbol{s}) = \mathbb{E}_{\pi,\tau}[\sum_{k=0}^{\infty} \gamma^k r(\boldsymbol{s}_{t+k}, \boldsymbol{a}_{t+k}, \boldsymbol{s}_{t+k+1}) \mid \boldsymbol{s}_t = \boldsymbol{s}]$ and $q^\pi(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\pi,\tau}[\sum_{k=0}^{\infty} \gamma^k r(\boldsymbol{s}_{t+k}, \boldsymbol{a}_{t+k}, \boldsymbol{s}_{t+k+1}) \mid \boldsymbol{s}_t = \boldsymbol{s},\, \boldsymbol{a}_t = \boldsymbol{a}]$. These value functions aid in evaluating the respective policy, as they both provide the expected discounted return when following policy π starting from an arbitrary state $\boldsymbol{s}$ (and action $\boldsymbol{a}$ for q^π). Furthermore, the value functions for the optimal policy π^* satisfy the Bellman optimality equations $v^*(\boldsymbol{s}) = \max_{\boldsymbol{a}} \mathbb{E}_\tau[r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) + \gamma v^*(\boldsymbol{s}_{t+1}) | \boldsymbol{s}_t = \boldsymbol{s},\, \boldsymbol{a}_t = \boldsymbol{a}]$ and $q^*(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_\tau[r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) + \gamma \max_{\boldsymbol{a}'} q^*(\boldsymbol{s}_{t+1}, \boldsymbol{a}') \mid \boldsymbol{s}_t = \boldsymbol{s},\, \boldsymbol{a}_t = \boldsymbol{a}]$.

Parameterized Actions

Although the action space of an MDP is usually either continuous or discrete, more complex environments with hybrid action spaces can also be considered. In a *Parameterized Action MDP* (PAMDP) [23], the hybrid action space has a particular structure, consisting of a discrete set of actions $\mathcal{A}_{\mathrm{D}} = [D] = \{1, \ldots, D\}$, each parameterized by a continuous set of parameters $\mathcal{A}_{\mathrm{C},\boldsymbol{a}_{\mathrm{D}}} \subseteq \mathbb{R}^m$. More precisely, the parameterized action space can be defined as $\mathcal{A} = \{(\boldsymbol{a}_{\mathrm{D}}, \boldsymbol{a}_{\mathrm{C}}) \mid \boldsymbol{a}_{\mathrm{D}} \in \mathcal{A}_{\mathrm{D}}, \boldsymbol{a}_{\mathrm{C}} \in \mathcal{A}_{\mathrm{C},\boldsymbol{a}_{\mathrm{D}}}\}$. For such PAMDPs, the policy can be decomposed into a discrete and continuous part as $\pi(\boldsymbol{a} \mid \boldsymbol{s}) = \pi_{\mathrm{D}}(\boldsymbol{a}_{\mathrm{D}} \mid \boldsymbol{s})\, \pi_{\mathrm{C},\boldsymbol{a}_{\mathrm{D}}}(\boldsymbol{a}_{\mathrm{C}} \mid \boldsymbol{s})$. Sampling an action $\boldsymbol{a} = (\boldsymbol{a}_{\mathrm{D}}, \boldsymbol{a}_{\mathrm{C}})$ can then be done in two steps: first the discrete action $\boldsymbol{a}_{\mathrm{D}}$ is sampled from π_{D}, and afterwards the continuous parameters $\boldsymbol{a}_{\mathrm{C}}$ are sampled from the corresponding $\pi_{\mathrm{C},\boldsymbol{a}_{\mathrm{D}}}$.

Model-Free RL and Exploration

In the model-free RL setting used throughout this work, the environment dynamics and reward function (ι, τ, r) remain unknown to the agent. To find an optimal policy, the agent is thus required to explore the state-action space during training. For this purpose, a behavioural policy $\beta(\boldsymbol{a} \mid \boldsymbol{s})$ is deployed, for which the encountered experience tuples $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$ are collected in a replay buffer $\mathcal{B}$. Batches of experience tuples sampled from this replay buffer can then be used for optimizing the policy and other models. To sufficiently explore the state-action space when working with deterministic policies, ϵ-greedy behavioural policies are typically used for discrete action spaces, and Gaussian behavioural policies for continuous action spaces. Such an ϵ-greedy policy takes a random action with probability ϵ and the greedy action following the policy $\pi(\boldsymbol{s})$

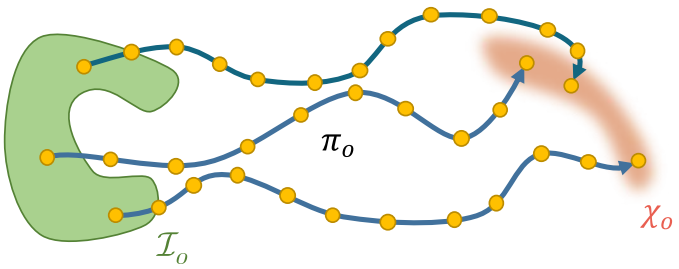

Figure 1: Schematic overview of an option acting in state space. Once in control, the option policy π_o can bring the agent from states in the initiation set $\mathcal{I}_o$ all the way to states with nonzero termination probabilities χ_o. For each intermediate state (yellow dots), the option policy provides the environment with suitable actions at every timestep to realize this transition.

otherwise, i.e. $\beta(\boldsymbol{a} \mid \boldsymbol{s}) = \epsilon|\mathcal{A}|^{-1} + (1-\epsilon)I_{\{\pi(\boldsymbol{s})\}}(\boldsymbol{a})$ where $I_{\mathcal{X}}(x)$ is the indicator function — equal to 1 when $x \in \mathcal{X}$ and 0 otherwise. The Gaussian policy, on the other hand, has the action taken by π as mean and σ^2 as variance, i.e. $\boldsymbol{a} \sim \mathcal{N}(\pi(\boldsymbol{s}), \sigma^2 I)$. The ϵ and σ parameters trade off the amount of exploration versus exploitation of the learned policy, and are typically attenuated throughout the training process.

2.2 Options

Instead of working directly with primitive actions $\boldsymbol{a}$, one can also work with *temporally extended* actions, commonly referred to as skills, options or templates [27, 36, 39]. While a primitive action is only 'active' for a single timestep, bringing the agent from one *state* to the next; options can last multiple timesteps, bringing the agent from one *region* in state space to another. Once an option is activated, and until it is terminated, the option policy takes over control and provides the environment with the necessary primitive actions at every timestep to complete this transition. Figure 1 visualizes this process.

Definition

More formally an option o can be defined by the triplet $(\mathcal{I}_o, \pi_o, \chi_o)$ with initiation set $\mathcal{I}_o \subset \mathcal{S}$, option policy $\pi_o(\boldsymbol{a}_t \mid \boldsymbol{s}_t)$ and termination condition $\chi_o(\boldsymbol{s}_{t+1})$. The initiation set determines for which regions in state space the option can be activated. Once the option is active, the agent samples actions from the option policy $\boldsymbol{a}_t \sim \pi_o(\cdot \mid \boldsymbol{s}_t)$ at every timestep until its termination. In general, the option's termination conditions χ_o are probabilistic. More precisely, for every state $\boldsymbol{s}_{t+1}$ the option terminates with probability $\chi_o(\boldsymbol{s}_{t+1})$ or remains active with probability $1 - \chi_o(\boldsymbol{s}_{t+1})$. In this work we specifically focus on *deterministic* options, which have both a deterministic policy $\boldsymbol{a}_t = \pi_o(\boldsymbol{s}_t)$ and deterministic termination conditions $\chi_o(\boldsymbol{s}_{t+1}) \in \{0, 1\}$. Such deterministic termination conditions effectively partition the state space, allowing us to define the termination set $\mathcal{T}_o = \{\boldsymbol{s}_{t+1} \in \mathcal{S} \mid \chi_o(\boldsymbol{s}_{t+1}) = 1\}$.

Remark that primitive actions $\boldsymbol{a}$ can be seen as a special case under this options framework. In particular, the option $(\mathcal{S}, I_{\{\boldsymbol{a}\}}, 1)$ is effectively equivalent to the primitive action $\boldsymbol{a}$, as it can be selected for all states $\mathcal{I} = \mathcal{S}$, has deterministic policy $\pi(\boldsymbol{s}_t) = \boldsymbol{a}$ for all states $\boldsymbol{s}_t$, and terminates after one timestep as $\chi(\boldsymbol{s}_{t+1}) = 1$ for all states $\boldsymbol{s}_{t+1}$ $(\mathcal{T} = \mathcal{S})$.

Hierarchical Control

Providing the agent with a set of options $\mathcal{O}$ to choose from, rather than the set of actions $\mathcal{A}$, naturally leads to a hierarchical control setup, as shown in Figure 3 and explained in Subsection 4.2. At the lowest level, the policy of the active option $o_t \in \mathcal{O}$ is now responsible for taking the appropriate actions at every timestep t, similar to the policy over actions in a classical setup. At the highest level, a novel *policy over options* or *master policy* $\pi_{\text{M}}(o_t \mid \boldsymbol{s}_t)$ is introduced, describing the probability of activating option o_t in state $\boldsymbol{s}_t$. This master policy can only select from the set of available options for a particular state, denoted by $\mathcal{O}^{\text{AV}}(\boldsymbol{s}) = \{o \in \mathcal{O} \mid \boldsymbol{s} \in \mathcal{I}_o\}$, which is fully determined by each of the option's initiation sets.

Remark that the policies at each level operate on different timescales. The active option's policy π_{o_t} operates on the finest timescale, selecting a new action at every timestep; whereas the policy over options π_{M} operates on a coarser timescale, only selecting a new option on termination of the previously active option. More precisely, $o_{t+1} \sim \pi_{\text{M}}(\cdot \mid \boldsymbol{s}_{t+1})$ when o_t terminates in $\boldsymbol{s}_{t+1}$, and $o_{t+1} = o_t$ otherwise. From the higher level's perspective, we are thus operating in a *Semi-MDP* [32], with the options providing the necessary link back and forth to the underlying MDP.

State-Option Value Function

Analogous to the state-action value functions for policies over actions, we can define the *state-option* value function for policies over options $q^{\pi_{\text{M}}}(\boldsymbol{s}, o) = \mathbb{E}_{\mathcal{O}, \pi_{\text{M}}, \tau}[\sum_{k=0}^{\infty} \gamma^k r(\boldsymbol{s}_{t+k}, \boldsymbol{a}_{t+k}, \boldsymbol{s}_{t+k+1}) \mid \boldsymbol{s}_t = \boldsymbol{s}, o_t = o]$. In this case, the expectation is taken over the probability distributions describing the active options $o_{t+1} \sim (1 - \chi_{o_t}(\boldsymbol{s}_{t+1}))I_{\{o_t\}}(\cdot) + \chi_{o_t}(\boldsymbol{s}_{t+1})\pi_{\text{M}}(\cdot \mid \boldsymbol{s}_{t+1})$, the selected actions $\boldsymbol{a}_t \sim \pi_{o_t}(\cdot \mid \boldsymbol{s}_t)$, and the state transitions $\boldsymbol{s}_{t+1} \sim \tau(\cdot \mid \boldsymbol{s}_t, \boldsymbol{a}_t)$ for all timesteps.

Sutton et al. [36] describe various strategies for finding the optimal master policy π^*_{M}. The simplest approach is *option-to-option* value learning, which is based on the optimal Bellman equation for the superimposed Semi-MDP, $q^*(\boldsymbol{s}, o) = \mathbb{E}_{\pi_o, \tau}\big[\sum_{k=0}^{d-1} \gamma^k r(\boldsymbol{s}_{t+k}, \boldsymbol{a}_{t+k}, \boldsymbol{s}_{t+k+1}) + \gamma^d \max_{o_{t+d} \in \mathcal{O}^{\text{AV}}(\boldsymbol{s}_{t+d})} q^*(\boldsymbol{s}_{t+d}, o_{t+d}) \;\big|\; \boldsymbol{s}_t = \boldsymbol{s}, o_t = o\big]$, with d the duration of o from $\boldsymbol{s}$. In other words, this method considers the execution of an option, from activation until termination, as an indivisible operation. In this work we specifically focus on the alternative *intra-option* value learning technique, which 'breaks up' the options in the finer timescale of the underlying MDP and also provides value updates for the intermediately visited states throughout an option's execution. When working with deterministic options, the optimal value function can then be writ-

ten down as $q^*(\boldsymbol{s}, o) = \mathbb{E}_\tau[r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) + \gamma u^*(\boldsymbol{s}_{t+1}, o_t) \mid \boldsymbol{s}_t = \boldsymbol{s}, o_t = o, \boldsymbol{a}_t = \pi_o(\boldsymbol{s}_t)]$ with

$$u^*(\boldsymbol{s}_{t+1}, o_t) = \begin{cases} q^*(\boldsymbol{s}_{t+1}, o_t) & \boldsymbol{s}_{t+1} \notin \mathcal{T}_{o_t} \\ \max\limits_{o_{t+1} \in \mathcal{O}^{\text{AV}}(\boldsymbol{s}_{t+1})} q^*(\boldsymbol{s}_{t+1}, o_{t+1}) & \boldsymbol{s}_{t+1} \in \mathcal{T}_{o_t} \end{cases}.$$

In Section 4 we show how a practical estimate of this optimal value function and the corresponding optimal master policy can be found using deep constrained Q-learning [16]. Following the intra-option technique instead of the option-to-option approach will turn out to be pivotal for supporting combined and hybrid options.

2.3 Methods

There are many ways to find an approximate optimal policy for (1). In this paper, we specifically limit ourselves to some recent model-free methods for learning deterministic policies. In particular, due to the discrete nature of option selection, we primarily focus on methods that support discrete ($\mathcal{A} = \mathcal{A}_{\text{D}} \subseteq \mathcal{O}$) or hybrid ($\mathcal{A} = \mathcal{A}_{\text{C}} \times \mathcal{A}_{\text{D}}$) action spaces. Policies over continuous action spaces ($\mathcal{A} = \mathcal{A}_{\text{C}}$) are only considered as a baseline to compare against in the conducted experiments. Figure 2 provides an overview of the different setups used throughout this work.

Actors and Critics

Two categories of methods can be immediately distinguished from this figure. On the one hand, *actor-critic* methods jointly train two deep neural networks: an actor $\mu(\boldsymbol{s}; \boldsymbol{\theta}_\mu)$ and a critic $q(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\theta}_q)$. The actor network represents a deterministic policy over *continuous* actions, $\boldsymbol{a}_{\text{C}} = \pi_{\text{C}}(\boldsymbol{s}) = \mu(\boldsymbol{s}; \boldsymbol{\theta}_\mu)$, approximating the optimal policy; whereas the critic network approximates the optimal value function. On the other hand, *value-based* methods only train one deep neural network, the Q-Network (critic) $q(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\theta}_q)$, which approximates the optimal value function. The optimal policy over *discrete* actions is not explicitly learned, but rather derived as the greedy policy for this learned value function, i.e. $\boldsymbol{a}_{\text{D}} = \pi_{\text{D}}(\boldsymbol{s}) = \arg\max_{\boldsymbol{a}} q(\boldsymbol{s}, \boldsymbol{a}; \boldsymbol{\theta}_q)$.

The critic network is typically trained by minimizing a squared temporal difference error, resulting in the critic loss

$$L_q(\boldsymbol{\theta}_q) = \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1}) \sim \mathcal{B}}\big[(q(\boldsymbol{s}_t, \boldsymbol{a}_t; \boldsymbol{\theta}_q) - y_t)^2\big], \quad (2)$$

where the target value y_t can be calculated in various ways, depending on the specific method. To improve the training stability, these target values are kept steady throughout training through the use of extra target networks. The parameters of these target networks are denoted with a prime, and are updated more slowly using Polyak averaging $\boldsymbol{\theta}'_i \leftarrow \tau \boldsymbol{\theta}_i + (1-\tau)\boldsymbol{\theta}'_i$. The actor network is trained by maximizing an expected discounted return estimate, usually involving an evaluation of the critic network. The specific definition of the actor loss $L_\mu(\boldsymbol{\theta}_\mu)$ depends on the method.

Although there are quite some similarities between the various approaches, there is an important difference in the architecture of the critic model, as illustrated in Figure 2. The continuous actions in the actor-critic setup are passed as an input to the critic model, which has a single output neuron corresponding to the value for the given state-action pair. On the other hand, in the value-based setup, only the state is passed as an input to the critic model, which has a dedicated output neuron for each discrete action to denote the value for the given state and respective action. When working with hybrid action spaces, a mixed critic architecture is used, which takes the continuous part of the action as an input (together with the state), and has separate output neurons for the discrete part of the action. Further details about each of the used methods are discussed in the remainder of this section.

Continuous Actions

For continuous action spaces, we work with actor-critic methods, such as 'Deep Deterministic Policy Gradient' (DDPG) [22] and 'Twin Delayed DDPG' (TD3) [12]. To ensure sufficient exploration of the state-action space, the Gaussian behavioural policy $\mathcal{N}(\pi(\boldsymbol{s}), \sigma^2 I)$ is deployed to populate the replay buffer. The critic network is trained by minimizing the critic loss (2) with target value $y_t = r_t + \gamma q(\boldsymbol{s}_{t+1}, \mu(\boldsymbol{s}_{t+1}; \boldsymbol{\theta}'_\mu); \boldsymbol{\theta}'_q)$ for the DDPG method. The actor network is updated simultaneously by minimizing the actor loss $L_\mu(\boldsymbol{\theta}_\mu) = -\mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{B}}[q(\boldsymbol{s}_t, \mu(\boldsymbol{s}_t; \boldsymbol{\theta}_\mu); \boldsymbol{\theta}_q)]$.

TD3 The performance and training stability of the original DDPG method can be further improved according to Fujimoto et al. [12]. In their proposed TD3 algorithm, extra twin networks are introduced, actor and target networks are updated less frequently than critic networks, and smoothed (regularized) target values are used. More precisely, two separate critic networks (the twins), denoted by parameters $\boldsymbol{\theta}_{q,1}$ and $\boldsymbol{\theta}_{q,2}$, are jointly trained using the same critic loss (2) but with altered target values $y_t = r_t + \gamma \min_{i \in \{1,2\}} q(\boldsymbol{s}_{t+1}, \mu(\boldsymbol{s}_{t+1}; \boldsymbol{\theta}'_\mu) + \epsilon_{\text{c}}; \boldsymbol{\theta}'_{q,i})$ with ϵ_{c} a noise sample from the clipped Gaussian distribution $\mathcal{N}(\boldsymbol{0}, \sigma_{\text{c}}^2 I)|_{[-c\,,\,c]}$.

Discrete Actions

For discrete action spaces, we resort to value-based approaches based on the popular 'Deep Q-Network' (DQN) [24]. To sufficiently explore the state-action space and fill the replay buffer, the ϵ-greedy behavioural policy is deployed. The model parameters can then be updated using samples from the replay buffer by minimizing the same critic loss (2) as before, although with a different calculation of the target values y_t. For the DQN method, the target values are calculated as $y_t = r_t + \gamma \max_{\boldsymbol{a}_{t+1}} q(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}; \boldsymbol{\theta}'_q)$.

Clipped Double DQN The performance of the original DQN method can be further improved by reducing overestimation bias [12, 41]. To this end, extra twin networks are introduced and the calculation of the target values is altered. More precisely, two separate Q-Networks (the twins), denoted by the parameters $\boldsymbol{\theta}_{q,1}$ and $\boldsymbol{\theta}_{q,2}$, are

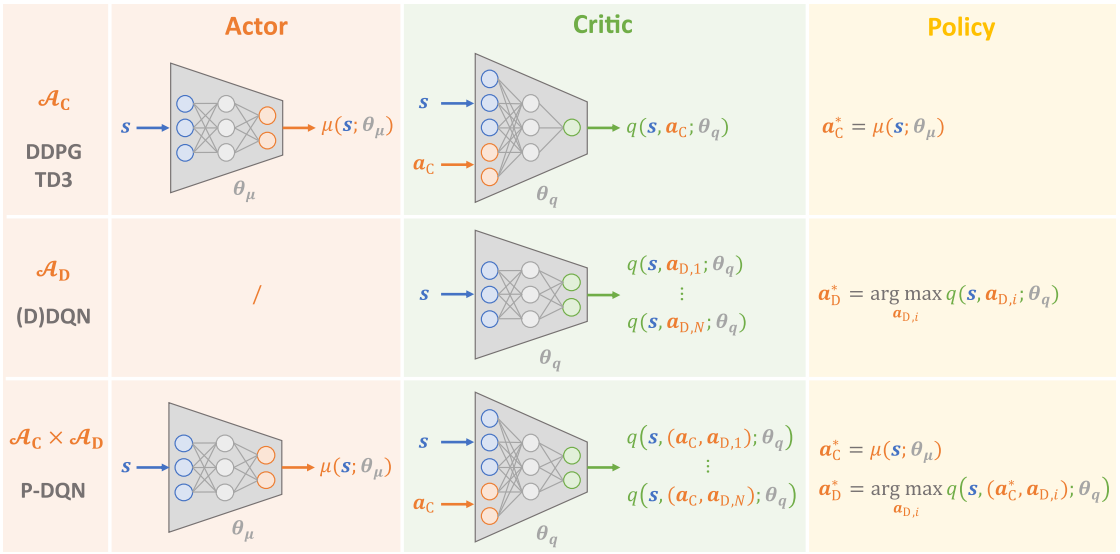

Figure 2: Overview of the model architectures and derived policies for the methods discussed in this paper. Top: actor-critic methods for policies over continuous actions $\boldsymbol{a}_\text{C} \in \mathcal{A}_\text{C}$. Middle: value-based methods for policies over discrete actions $\boldsymbol{a}_\text{D} \in \mathcal{A}_\text{D}$. Bottom: actor-critic methods for policies over hybrid actions $(\boldsymbol{a}_\text{C}, \boldsymbol{a}_\text{D}) \in \mathcal{A}_\text{C} \times \mathcal{A}_\text{D}$.

jointly trained. The policy is derived from the first twin, i.e. $\pi(\boldsymbol{s}) = \arg\max_{\boldsymbol{a}} q(\boldsymbol{s},\boldsymbol{a};\boldsymbol{\theta}_{q,1})$. The same loss function (2) is used to train each of the twins, but the target value is now calculated as $y_t = r_t + \gamma \min_{i\in\{1,2\}} q(\boldsymbol{s}_{t+1}, \pi(\boldsymbol{s}_{t+1});\boldsymbol{\theta}'_{q,i})$.

Constrained DQN Another DQN extension, the 'Constrained DQN' method, is proposed by Kalweit et al. [16] and aims to properly handle constrained (state-dependent) action spaces $\mathcal{A}(\boldsymbol{s}) \subset \mathcal{A}$. This is done by deriving the policy as the *pruned* greedy policy $\pi(\boldsymbol{s}) = \arg\max_{\boldsymbol{a}\in\mathcal{A}(\boldsymbol{s})} q(\boldsymbol{s},\boldsymbol{a};\boldsymbol{\theta}_q)$. Kalweit et al. [16] illustrate that this pruning is also necessary in the calculation of the target values, $y_t = r_t + \gamma \max_{\boldsymbol{a}_{t+1}\in\mathcal{A}(\boldsymbol{s}_{t+1})} q(\boldsymbol{s}_{t+1},\boldsymbol{a}_{t+1};\boldsymbol{\theta}'_q)$, as otherwise suboptimal policies are retrieved.

Hybrid Actions

There are various approaches to deal with hybrid action spaces, depending on the specific application [11, 23, 28]. In this work we exclusively train agents in PAMDPs which have the same continuous parameters for each discrete action, i.e. $\mathcal{A} = \mathcal{A}_\text{C} \times \mathcal{A}_\text{D}$. The 'Parameterized DQN' (P-DQN) [43] method was chosen to deal with such hybrid action spaces, as its architecture and methodology resemble the previously discussed TD3 and DQN methods the most.

Parameterized DQN In the P-DQN method, the hybrid policy can be split into a discrete and continuous part, $\boldsymbol{a} = \pi(\boldsymbol{s}) = (\pi_\text{C}(\boldsymbol{s}), \pi_\text{D}(\boldsymbol{s})) = (\boldsymbol{a}_\text{C}, \boldsymbol{a}_\text{D})$. Similar to the previously discussed actor-critic methods, two deep neural networks are jointly trained: a critic $q(\boldsymbol{s},(\boldsymbol{a}_\text{C},\boldsymbol{a}_\text{D});\boldsymbol{\theta}_q)$ for approximating the optimal value function and an actor $\mu(\boldsymbol{s};\boldsymbol{\theta}_\mu)$ for approximating the optimal policy over the continuous parameters. The policy over continuous parameters is thus explicitly learned through the actor model $\pi_\text{C}(\boldsymbol{s}) = \mu(\boldsymbol{s};\boldsymbol{\theta}_\mu)$, whereas the policy over discrete actions is implicitly derived from the learned critic model as $\pi_\text{D}(\boldsymbol{s}) = \arg\max_{\boldsymbol{a}_\text{D}} q(\boldsymbol{s},(\pi_\text{C}(\boldsymbol{s}),\boldsymbol{a}_\text{D});\boldsymbol{\theta}_q)$. To sufficiently explore the state-action space, an ϵ-greedy behavioural policy is used for the discrete actions, and a Gaussian behavioural policy for the continuous parameters. Once again, the same loss (2) can be used for optimizing the critic model, with target values calculated as $y_t = r_t + \gamma \max_{\boldsymbol{a}_{\text{D},t+1}\in\mathcal{A}_\text{D}} q(\boldsymbol{s}_{t+1},(\boldsymbol{a}_{\text{C},t+1},\boldsymbol{a}_{\text{D},t+1});\boldsymbol{\theta}'_q)$ and $\boldsymbol{a}_{\text{C},t+1} = \mu(\boldsymbol{s}_{t+1};\boldsymbol{\theta}'_\mu)$. The actor model is trained by minimizing $L_\mu(\boldsymbol{\theta}_\mu) = -\mathbb{E}_{\boldsymbol{s}_t\sim\mathcal{B}}[\sum_{\boldsymbol{a}_\text{D}\in\mathcal{A}_\text{D}} q(\boldsymbol{s}_t,(\mu(\boldsymbol{s}_t;\boldsymbol{\theta}_\mu),\boldsymbol{a}_\text{D});\boldsymbol{\theta}_q)]$.

3 Autonomous Driving Setup

The hybrid options framework introduced in the next section is applied to an autonomous driving problem on highways. A proprietary highway simulator is used for this purpose. In this section the most relevant components of this simulator and environment setup are briefly introduced.

3.1 Objective and Reward

The objective in this environment is to efficiently navigate highway traffic, while adhering to safety constraints (avoid crashes), driving regulations (keep right) and comfort constraints (smooth movements). A three lane highway with both straight and curved segments is considered for all experiments. Various vehicles are randomly distributed along the different available lanes and are assigned different target velocities at initialization. The virtual driver (agent) is in control of a single vehicle, assigned to drive the maximum allowed speed.

The objective and soft constraints are encoded in the

reward signal as a weighted sum of different penalties

$$r = \frac{w_{\mathrm{F}} r_{\mathrm{F}} + w_{\mathrm{V}} r_{\mathrm{V}} + w_{\mathrm{C}} r_{\mathrm{C}} + w_{\mathrm{R}} r_{\mathrm{R}}}{w_{\mathrm{F}} + w_{\mathrm{V}} + w_{\mathrm{C}} + w_{\mathrm{R}}}.$$

In order of appearance, these components give a penalty for insufficient following distance (r_{F}), inappropriate speed (r_{V}), drifting from lane center (r_{C}) and not keeping right (r_{R}). An extra penalty is given when the virtual driver collides with other vehicles or the highway boundaries.

3.2 Vehicle Dynamics and Policies

The kinematic bicycle model (KBM) [33] is used to update each vehicle's state, consisting of a global x and y position, heading angle ψ and speed v,

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v\cos(\psi + \zeta) \\ v\sin(\psi + \zeta) \\ \frac{v}{l_r}\sin\zeta \\ \frac{a}{\cos\zeta} \end{bmatrix} \quad \zeta = \arctan\left(\frac{l_r}{l_f + l_r}\tan\delta\right).$$

The inputs of this dynamical system are the steering angle δ and the longitudinal acceleration a. In this work, we do not further consider such *global* state vectors and low-level inputs; instead *projected* state vectors (into the road's frame of reference) and high-level actions are used. Henceforth, we denote the ego vehicle's projected longitudinal velocity (along the road) by v, and its projected lateral offset (with respect to the road edge) by d. Dedicated vehicle motion controllers aid in stabilizing the vehicle on the road and facilitate the driving task of the virtual driver. These controllers track the references (actions) $\boldsymbol{a}$, consisting of a desired relative longitudinal velocity Δv and relative lateral position Δd, set by the driving policies.

To take correct high level steering decisions, the virtual driver needs some extra information about other traffic participants in its neighbourhood. This information is gathered in the agent's observation vector $\boldsymbol{s}$, containing local information such as the vehicle's velocity components and its offset with respect to different lane centers, and relative information such as relative distances and velocities with respect to neighbouring traffic. The relative offsets with respect to the lane center of the current lane ($\Delta L = 0$) and nearest lanes on the left ($\Delta L = 1$) and right ($\Delta L = -1$) side of the ego vehicle are referred to as $c_{\Delta L}$. In case there is no subsequent lane on a particular side, $c_{\pm 1}$ is set equal to c_0.

Every vehicle is controlled by a policy, mapping observations $\boldsymbol{s}$ to suitable actions $\boldsymbol{a}$. The policy of the autonomous vehicle is learned using any of the described reinforcement learning methods in this paper. The policies of the other vehicles in the simulation environment are fixed beforehand. A mixture of vehicles equipped with a custom rule-based policy and a policy implementing the 'Intelligent Driver Model' (IDM) [40] and 'Minimizing Overall Braking Induced by Lane change' (MOBIL) [17] is used. Both policies try to mimic rudimentary human driving behaviour, although being fully deterministic.

3.3 Safety and Comfort constraints

The safety and comfort requirements are not encoded in the reward signal as soft constraints, but dealt with separately as hard constraints. For policies over continuous actions (directly providing setpoints for the motion controllers), safety is ensured through state-dependent action bounds as in De Cooman et al. [10]. These action bounds $\boldsymbol{a}_{\mathrm{L}}(\boldsymbol{s}) \leq \boldsymbol{a} \leq \boldsymbol{a}_{\mathrm{U}}(\boldsymbol{s})$ are derived from the braking criterion

$$\min\left(\Delta x, \Delta x + \frac{v_{\mathrm{L}}^2}{2b} - \frac{v_{\mathrm{F}}^2}{2b}\right) > \Delta x_{\mathrm{SAFE}}, \tag{3}$$

where Δx is the following distance between 2 vehicles, v_{L} and v_{F} are the velocities of the leading and following vehicle respectively, b is the maximum deceleration of both vehicles, and Δx_{SAFE} is a minimum safe distance to keep between both vehicles. Complying with this braking criterion ensures the following vehicle is always able to avoid a collision with the leading vehicle, under some reasonable worst-case assumptions [10]. To indicate that an action a is mapped to the safe interval defined by the action bounds, the following (clipping) notation is used $a|_{[a_{\mathrm{L}}, a_{\mathrm{U}}]} = \min[\max(a, a_{\mathrm{L}}), a_{\mathrm{U}}]$.

In this work, we employ the same braking criterion to enforce safety on policies using (hybrid) options. Moreover, under this options framework, additional comfort constraints (smooth lane changes) can be readily embedded in dedicated manoeuvre policies, as discussed in the next section.

4 Methodology

To restrict the driving behaviour of the virtual driver, we specifically predefine a set of options for longitudinal and lateral control of autonomous vehicles. Each option's policy automatically provides the necessary actions to perform certain manoeuvres over the course of multiple timesteps, while adhering to the given hard constraints by design. For example, safety of the learned driving policies can be ensured at the finest (MDP) timescale, by reusing the state-dependent action bounds for policies over continuous actions. The virtual driver can thus choose from a comprehensive collection of safe and comfortable manoeuvres (options) based on the current traffic situation, instead of selecting individual low-level actions. This makes the options framework especially suitable for autonomous driving applications, and naturally leads to a hierarchical control architecture, as visualized in Figure 3.

4.1 Safe Options for Autonomous Driving

We start with a concise description of the various predefined options tailored to autonomous driving tasks. A virtual driver typically needs to consider two high-level control decisions at every moment in time: longitudinal (velocity) control and lateral (position) control. For both of these control tasks, dedicated options are introduced and summarized in Table 1. The set of options acting on the longitudinal velocity is denoted by $\mathcal{O}_{\mathrm{V}} = \{o_{\mathrm{E}}, o_{\mathrm{M}}, o_{\mathrm{VD}}, o_{\mathrm{VI}}\}$, whereas

Table 1: Overview of the introduced options for autonomous driving.

Manoeuvre Option		**Target Velocity $\mathbf{v}_\mathrm{T}$**	**Target Offset $\mathbf{d}_\mathrm{T}$**	**Initiation Set $\mathcal{I}$**	**Termination Set $\mathcal{T}$**
Emergency	o_E	$0\vert_{[v+\Delta v_\mathrm{L}\,,\,v+\Delta v_\mathrm{U}]}$	$d\vert_{[d+\Delta d_\mathrm{L}\,,\,d+\Delta d_\mathrm{U}]}$	$\mathcal{S}$	$\mathcal{S}$
Maintain	o_M	v	d	$\{\boldsymbol{s} \in \mathcal{S} \mid \mathrm{Safe}[\boldsymbol{s}; v_\mathrm{T}, d_\mathrm{T}]\}$	$\mathcal{S}$
Velocity Change (−)	o_VD	$\left\lceil \frac{v}{\delta_v} - 1 \right\rceil \delta_v$	d	$\{\boldsymbol{s} \in \mathcal{S} \mid \mathrm{Safe}[\boldsymbol{s}; v_\mathrm{T}, d_\mathrm{T}]\}$	$\left\{ \boldsymbol{s} \in \mathcal{S} \;\middle\vert\; \begin{array}{r} \vert v_\mathrm{T} - v\vert < \epsilon_v \\ \vee \neg\mathrm{Safe}[\boldsymbol{s}; v_\mathrm{T}, d_\mathrm{T}] \end{array} \right\}$
Velocity Change (+)	o_VI	$\left\lfloor \frac{v}{\delta_v} + 1 \right\rfloor \delta_v$			
Lane Change (left)	o_LL	v	$d + c'_1$	$\left\{ \boldsymbol{s} \in \mathcal{S} \;\middle\vert\; \begin{array}{r} \vert d_\mathrm{T} - d\vert \geq \epsilon_d \wedge v \geq v_\mathrm{m} \\ \wedge\, \mathrm{Safe}[\boldsymbol{s}; v_\mathrm{T}, d_\mathrm{T}] \end{array} \right\}$	$\left\{ \boldsymbol{s} \in \mathcal{S} \;\middle\vert\; \begin{array}{r} \vert d_\mathrm{T} - d\vert < \epsilon_d \vee v < v_\mathrm{m} \\ \vee \neg\mathrm{Safe}[\boldsymbol{s}; v_\mathrm{T}, d_\mathrm{T}] \end{array} \right\}$
Lane Change (right)	o_LR		$d + c'_{-1}$		

the set of options acting on the lateral position is denoted by $\mathcal{O}_\mathrm{D} = \{o_\mathrm{E}, o_\mathrm{M}, o_\mathrm{LL}, o_\mathrm{LR}\}$. Once activated, the option's policy provides the vehicle motion controllers with the necessary actions to perform a certain predefined manoeuvre, bringing the ego vehicle from one region in state space to another over the course of multiple timesteps (e.g. a lane change manoeuvre). The virtual driver only needs to select another option after the active option terminates, either after successfully completing its associated manoeuvre, or after an abortion caused by other termination conditions (such as safety constraint violations). Although this manoeuvre is limited to either a longitudinal or lateral movement for the options introduced in this work, our methodology allows to readily combine multiple options (and their manoeuvres), as outlined in Subsection 4.2.

Option Targets

The effect of each option on the longitudinal velocity or lateral position is indicated by the target velocity v_T and target offset d_T. The associated option policy provides the necessary actions to achieve these targets from any initial state, while adhering to hard safety and comfort constraints during the manoeuvre's execution *by design*. For example, the following option policy could be used for this purpose, $\pi_o(\boldsymbol{s}) = \left[v_\mathrm{T} - v; d_\mathrm{T} - d\right] \vert_{[\boldsymbol{a}_\mathrm{L}(\boldsymbol{s})\,,\,\boldsymbol{a}_\mathrm{U}(\boldsymbol{s})]}$, as it takes safety constraints into account by enforcing the state-dependent action bounds (but disregards any other comfort constraints). Remark that for the Markov property to hold, it is essential to have fixed targets for all states, regardless of whether the option is already active or not in a particular state.

The option targets are also used to assess the safety of the option's associated manoeuvre. More precisely, the manoeuvre is considered to be safe for a certain state if the target velocity and position are within the state-dependent action bounds for that state, i.e. $\mathrm{Safe}[\boldsymbol{s}; v_\mathrm{T}, d_\mathrm{T}] = \Delta v_\mathrm{L} \leq v_\mathrm{T} - v \leq \Delta v_\mathrm{U} \wedge \Delta d_\mathrm{L} \leq d_\mathrm{T} - d \leq \Delta d_\mathrm{U}$. In our (model-free) setup, this effectively corresponds to verifying that the braking criterion (3) holds for all interpolated states between the current state and an estimated target state at the end of the manoeuvre. This estimated target state is simply derived from the current state by instantaneously moving the ego vehicle to the target position and immediately adopting the target velocity, while leaving all other state components unchanged. For model-based methods, the learned environment model could be leveraged instead to obtain more accurate state predictions, for which the braking criterion can be checked (or from which action bounds can be derived).

Option Safety

Safety of the various options is then ensured by extending the termination sets and restricting the initiation sets. More specifically, if the planned manoeuvre is considered unsafe in a certain state, that state is added to the termination set and removed from the initiation set. As a result, an option can only be activated if it will not bring the virtual driver to an unsafe state during the option's execution (according to the used prediction model); and it is terminated as soon as new predictions, obtained throughout the option's execution, become unsafe. Remark that the initiation and termination sets do not need to be fully constructed beforehand, rather the initiation and termination conditions can be evaluated for each individual state encountered by the virtual driver. By reevaluating the safety conditions at every timestep, the latest available state information is always used in deciding whether to activate or terminate an option.

Option Interruption

Most of the introduced options run until their targets are reached or their safety conditions are violated. To prevent unlimited execution and to ensure other options can be selected at any time, certain *primitive* options are also defined, which are automatically terminated after every time step ($\mathcal{T} = \mathcal{S}$). An alternative implementation could make the options *interruptable* instead [36], such an extension is however not further investigated in this work.

Overview

The various options from Table 1 are succinctly described below. Safety and comfort constraints are embedded in each option's policy, as well as through its termination and initiation sets.

Emergency Option To ensure that at least one option is always available, the emergency option o_E is introduced. This option tries to slow down the vehicle as fast as possible ($v_\mathrm{T} \to 0$) and tries to maintain the current lateral position ($d_\mathrm{T} \to d$), while adhering to the safety constraints. For any encountered unsafe states, the vehicle is moved towards

the nearest safe region, by clipping the target values using the state-dependent action bounds. Although it is the only option that is always available ($\mathcal{I} = \mathcal{S}$), it should be considered as a fallback for emergency situations; as generally, other, more suitable options should be available. Therefore, it is defined as a primitive option ($\mathcal{T} = \mathcal{S}$), ensuring this emergency manoeuvre can be interrupted at any time.

Maintain Option Additionally, we specify the primitive maintain option o_{M}, which maintains the current longitudinal velocity ($v_{\text{T}} = v$) and lateral position ($d_{\text{T}} = d$). This option is only available when it is safe to keep driving at the current velocity and position ($\text{Safe}[\boldsymbol{s}; v_{\text{T}}, d_{\text{T}}]$); and it is terminated after every time step ($\mathcal{T} = \mathcal{S}$), allowing other options to be selected at any moment.

Longitudinal Options For longitudinal control, we define two *velocity change* options, which decrease (o_{VD}) or increase (o_{VI}) the current velocity with fixed increments[1] $\delta_v = 2\,\text{m/s}$. These options provide the necessary actions to attain the target velocity v_{T} in a safe and comfortable manner. They can only be initiated when the target velocity is considered to be safe ($\text{Safe}[\boldsymbol{s}; v_{\text{T}}, d_{\text{T}}]$); and they terminate once the target velocity is reached ($|v_{\text{T}} - v| < \epsilon_v = 0.01\,\text{m/s}$) or when it is no longer safe to drive at the target velocity ($\neg\text{Safe}[\boldsymbol{s}; v_{\text{T}}, d_{\text{T}}]$).

Lateral Options For lateral control, we similarly define two *lane change* options, providing the necessary actions to steer the ego vehicle towards the nearest lane center to its left (o_{LL}) or right (o_{LR}) in approximately 5 s (for a complete lane change at maximum velocity). The relative offsets to the nearest lane center on the left and right of the ego vehicle are denoted by c'_1 and c'_{-1}. Both of these are often equal to their non-primed counterparts, unless the vehicle is not properly centered within its current lane and the current lane's center is the nearest in the respective direction (in which case $c'_{\pm 1} = c_0$)[1]. These options can only be activated when the car is not yet centered in the target lane ($|d_{\text{T}} - d| \geq \epsilon_d = 0.05\,\text{m}$), and the predicted target state is safe ($\text{Safe}[\boldsymbol{s}; v_{\text{T}}, d_{\text{T}}]$). Additionally, a minimum velocity condition ($v \geq v_{\text{m}} = 3\,\text{m/s}$) is enforced to ensure that the lane change manoeuvre can be completed in a reasonable timeframe, as otherwise the option could remain active indefinitely. The option terminates once aligned with the target lane's center ($|d_{\text{T}} - d| < \epsilon_d$), or once the minimum velocity is no longer attained ($v < v_{\text{m}}$), or when the predicted target state is no longer considered to be safe ($\neg\text{Safe}[\boldsymbol{s}; v_{\text{T}}, d_{\text{T}}]$).

4.2 Hierarchical Control with Options

The previously described options provide the virtual driver with several safe and comfortable manoeuvres for the control of autonomous vehicles. These options can be applied in various hierarchical control architectures. Figure 3 shows an overview of the different setups investigated in this work.

[1] This is necessary for the Markov property to hold.

Baseline

The different setups for hierarchical control with options are compared with other approaches using the baseline setup (Figure 3.a). In this baseline setup, a 'classical' policy over primitive actions directly interacts with the vehicle motion controllers through the continuous setpoints $\boldsymbol{a} = \begin{bmatrix}\Delta v; \Delta d\end{bmatrix}$. This policy can either be learned using standard reinforcement learning methods (such as TD3), or be predefined (e.g. following IDM and MOBIL).

Options

With the introduction of dedicated options, the virtual driver obtains the ability to select appropriate manoeuvres to efficiently navigate traffic. In other words, the problem becomes a decision-making task over a discrete set of options (manoeuvres) instead of over a continuous range of actions. We distinguish the higher level actions selected by the master policy, $\boldsymbol{a}_{\text{M}} = \pi_{\text{M}}(\boldsymbol{s})$, from the lower level actions that are passed to the vehicle motion controllers, $\boldsymbol{a} = \pi(\boldsymbol{s}, \boldsymbol{a}_{\text{M}}) = \begin{bmatrix}\Delta v; \Delta d\end{bmatrix}$.

In the most basic setup (Figure 3.b), only one option is activated at any time, selected from the complete set of options, $\boldsymbol{a}_{\text{M}} = o \in \mathcal{O}$ ($= \mathcal{O}_{\text{V}} \cup \mathcal{O}_{\text{D}}$). The selected option takes over control until termination and supplies the vehicle motion controllers with both longitudinal and lateral setpoints, $\begin{bmatrix}\Delta v; \Delta d\end{bmatrix} = \pi_o(\boldsymbol{s})$. The master policy π_{M} and its behavioural counterpart β_{M} only select a new option upon termination of the previously active one (or at the start of the episode).

Combined Options

A drawback of the basic options approach is that only a single manoeuvre can be performed at any given time. For example, the learned policies lack the ability to combine a lateral lane change manoeuvre with a longitudinal acceleration manoeuvre, which can be performed by the continuous baseline policies. To address this limitation, we also propose an extended hierarchical control architecture with support for *combined* options for longitudinal and lateral control (Figure 3.c). Such a setup can significantly increase the flexibility of the virtual driver, providing it with similar capabilities as the baseline models. In particular, two options are activated at any time, $\boldsymbol{a}_{\text{M}} = (o_{\text{V}}, o_{\text{D}}) \in \mathcal{O}_{\text{V}} \times \mathcal{O}_{\text{D}}$, one for longitudinal control o_{V} and one for lateral control o_{D}. The selected options take over control until termination, and each of them supplies a longitudinal or lateral setpoint to the vehicle motion controllers, $\Delta v = \begin{bmatrix}1 & 0\end{bmatrix} \pi_{o_{\text{V}}}(\boldsymbol{s})$ and $\Delta d = \begin{bmatrix}0 & 1\end{bmatrix} \pi_{o_{\text{D}}}(\boldsymbol{s})$. As soon as either option terminates, the (behavioural) master policy π_{M} (β_{M}) selects a new option for the respective direction of control, while the other non-terminating option remains active.

Hybrid Options

Finally, a hybrid setup (Figure 3.d) is proposed with the aim of combining the best of the continuous (baseline) and discrete (options) approaches. The resulting setup closely

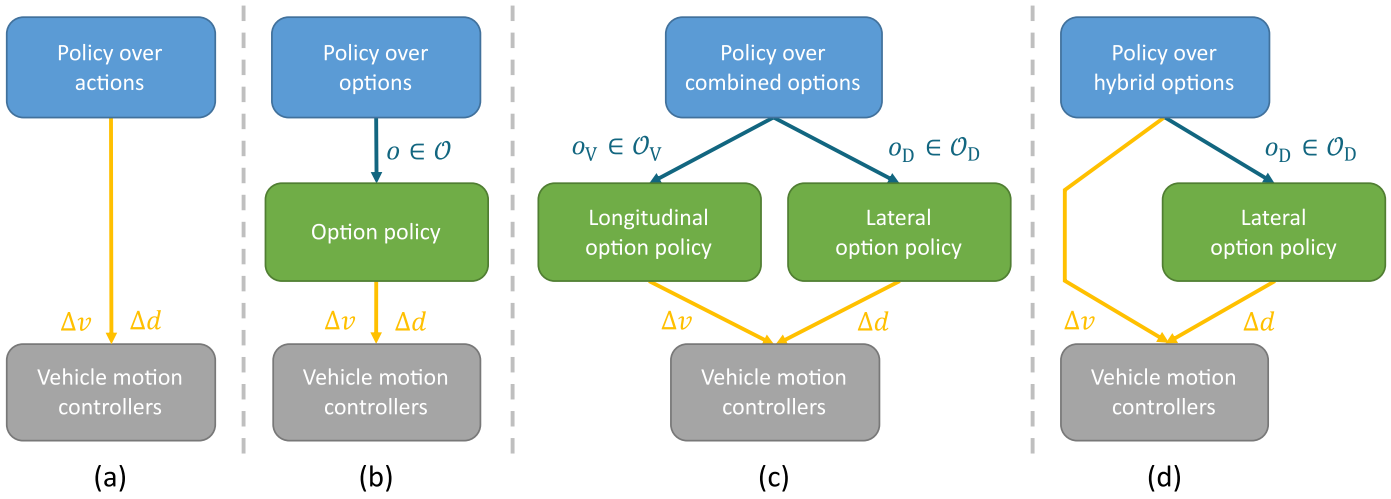

Figure 3: Schematic overview of the different autonomous driving policies used in this work. On the left (a) the baseline policy over primitive actions $(\Delta v, \Delta d)$, directly interfacing with the vehicle motion controllers. On the right (b+c+d) different configurations of policies over options, which select one (or two) new option(s) to execute after a previous one terminated. Once an option policy is in control, it provides the vehicle motion controllers with suitable actions to perform its associated manoeuvre and reach the targets v_{T} and d_{T} (see Figure 1 and Table 1).

aligns with the decision process of human drivers. More specifically, the hybrid master policy can select a continuous longitudinal velocity parameter, while picking a discrete lateral manoeuvre (option), $\boldsymbol{a}_{\mathrm{M}} = (\Delta v, o_{\mathrm{D}}) \in \mathbb{R} \times \mathcal{O}_{\mathrm{D}}$. The velocity setpoint Δv is passed directly to the vehicle motion controllers, while the selected option supplies the lateral position setpoint $\Delta d = \begin{bmatrix} 0 & 1 \end{bmatrix} \pi_{o_{\mathrm{D}}}(\boldsymbol{s})$. Remark that the continuous part of the (behavioural) master policy $\pi_{\mathrm{M,C}}$ $(\beta_{\mathrm{M,C}})$ selects a new velocity setpoint at every timestep, whereas the discrete part of the (behavioural) master policy $\pi_{\mathrm{M,D}}$ $(\beta_{\mathrm{M,D}})$ only selects a new option upon termination of the previously active one.

4.3 Implementation

We apply a combination of existing RL techniques to derive intra-option value learning and actor-critic algorithms for the various hierarchical control setups with options. The generic algorithm is summarized in Algorithm 1 and closely follows the training principles of state-of-the-art RL methods. More detailed implementations for each setup are provided in Appendix A.

Remark. *Although in Algorithm 1 a sample from the semi-Markov (behavioural) master policy is taken for every timestep, this is often not needed in practice. In particular, the discrete component of the policy should only select a new option upon termination of the active option.*

Algorithm 1 Hierarchical RL with Options (Generic)

Initialize critic (and actor) models $\boldsymbol{\theta}_{q,i}, \boldsymbol{\theta}'_{q,i}$ $(\boldsymbol{\theta}_\mu, \boldsymbol{\theta}'_\mu)$
$\mathcal{B} \leftarrow \emptyset$
for each episode **do**
  for each environment step t **do**
    $\boldsymbol{a}_{\mathrm{M},t} \sim \beta_{\mathrm{M}}(\cdot \mid \boldsymbol{s}_t, \boldsymbol{a}_{\mathrm{M},t-1})$ ▷ Select high level action
    $\boldsymbol{a}_t = \pi(\boldsymbol{s}, \boldsymbol{a}_{\mathrm{M},t})$ ▷ Obtain low level action
    $\boldsymbol{s}_{t+1} \sim \tau(\cdot \mid \boldsymbol{s}_t, \boldsymbol{a}_t)$
    $r_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{(\boldsymbol{s}_t, \boldsymbol{a}_{\mathrm{M},t}, r_t, \boldsymbol{s}_{t+1})\}$
  end for
  for each gradient step **do**
    Sample batch B from $\mathcal{B}$
    for all $(\boldsymbol{s}_t, \boldsymbol{a}_{\mathrm{M},t}, r_t, \boldsymbol{s}_{t+1}) \in B$ **do**
      $\boldsymbol{a}_{\mathrm{M},t+1} = \pi_{\mathrm{M}}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{\mathrm{M},t})$ ▷ Calculate targets
      $y_t = r_t + \gamma \min_{i \in \{1,2\}} q(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{\mathrm{M},t+1}; \boldsymbol{\theta}'_{q,i})$
    end for
    $\widehat{L}_q(\boldsymbol{\theta}_{q,i}; B) = \frac{1}{|B|} \sum_B (q(\boldsymbol{s}_t, \boldsymbol{a}_{\mathrm{M},t}; \boldsymbol{\theta}_{q,i}) - y_t)^2$
    $\widehat{L}_\mu(\boldsymbol{\theta}_\mu; B) = \frac{-1}{|B|} \sum_B \sum_{o_{\mathrm{D}} \in \mathcal{O}_{\mathrm{D}}^{\mathrm{AV}}(\boldsymbol{s}_t)} q(\boldsymbol{s}_t, (\mu(\boldsymbol{s}_t; \boldsymbol{\theta}_\mu), o_{\mathrm{D}}); \boldsymbol{\theta}_{q,1})$
    $\boldsymbol{\theta}_{q,i} \leftarrow \boldsymbol{\theta}_{q,i} - \eta_q \nabla_{\boldsymbol{\theta}_{q,i}} \widehat{L}_q(\boldsymbol{\theta}_{q,i}; B)$ ▷ Update critics
    $\boldsymbol{\theta}_\mu \leftarrow \boldsymbol{\theta}_\mu - \eta_\mu \nabla_{\boldsymbol{\theta}_\mu} \widehat{L}_\mu(\boldsymbol{\theta}_\mu; B)$ ▷ Update actor
    $\boldsymbol{\theta}'_m \leftarrow \tau \boldsymbol{\theta}_m + (1 - \tau) \boldsymbol{\theta}'_m$ ▷ Update target models
  end for
end for

Option Availability

At several points during training and deployment of the master policy over options, we need to know which options are available to the agent. The set $\mathcal{O}^{\mathrm{AV}}(\boldsymbol{s})$ provides this information at the time of selecting a new option, but that is insufficient. Since the options investigated in this work are uninterruptible, the only available option is often the currently active one. Therefore, we also define the closely related set of available options for a particular state $\boldsymbol{s}'$, *given that option o is active* upon reaching this state. This extended availability set additionally depends on the option's

termination sets, and is defined as

$$\mathcal{O}^{\text{AV}}(\boldsymbol{s}', o) = \begin{cases} \mathcal{O}^{\text{AV}}(\boldsymbol{s}') & \text{if } \boldsymbol{s}' \in \mathcal{T}_o \\ \{o\} & \text{otherwise} \end{cases}.$$

Similarly as for the initiation and termination sets, the availability sets do not need to be fully constructed beforehand, but can be evaluated 'on the go' for the encountered states and options. For very complex initiation or termination set conditions, the option availabilities for states $\boldsymbol{s}_t$ and $\boldsymbol{s}_{t+1}$ can also be stored in the replay buffer to speed up the computations (at the cost of increased memory usage).

Intra-Option Value Learning

The critic networks are trained using a variant of the clipped DDQN method [12], adapted for use with options by following the intra-option value learning technique. Additionally, the methodology of the constrained DQN method [16] is applied to enforce the constrained initiation and termination of the options. Ultimately, both critic networks (twins) can be trained by minimizing the critic loss (2) with target values $y_t = r_t + \gamma \min_{i\in\{1,2\}} q(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{\text{M},t+1}; \boldsymbol{\theta}'_{q,i})$ and $\boldsymbol{a}_{\text{M},t+1}$ obtained from the (constrained) master policy π_{M}.

By breaking up the options into their constituent actions for every timestep, the critic models can be updated from every individual interaction with the environment stored in the replay buffer (similar to classical RL approaches). Moreover, this allows to flexibly combine an action from the option's policy with an action from other policies (either over options or primitive actions) in every timestep, which is not possible in the option-to-option approach.

Remark. *Even though the option policies have to satisfy the Markov property in order to be able to use the intra-option learning rule, experiments suggested it also works well for option policies that do not always satisfy the Markov property.*

Options

The master policy over single options (Figure 3.b) is derived from the first critic twin as $\boldsymbol{a}'_{\text{M}} = \pi_{\text{M}}(\boldsymbol{s}', \boldsymbol{a}_{\text{M}}) = \arg\max_{o'\in\mathcal{O}^{\text{AV}}(\boldsymbol{s}',o)} q(\boldsymbol{s}', o'; \boldsymbol{\theta}_{q,1})$, or equivalently

$$o_{t+1} = \begin{cases} \arg\max_{o'\in\mathcal{O}^{\text{AV}}(\boldsymbol{s}_{t+1})} q(\boldsymbol{s}_{t+1}, o'; \boldsymbol{\theta}_{q,1}) & \boldsymbol{s}_{t+1} \in \mathcal{T}_{o_t} \\ o_t & \text{otherwise} \end{cases}.$$

The behavioural master policy β_{M} is the ϵ-greedy policy derived from π_{M}. Remark that the random selection must take into account the option availabilities, i.e. $\beta_{\text{M}}(\boldsymbol{a}'_{\text{M}} \mid \boldsymbol{s}', \boldsymbol{a}_{\text{M}}) = \epsilon|\mathcal{O}^{\text{AV}}(\boldsymbol{s}', o)|^{-1} I_{\mathcal{O}^{\text{AV}}(\boldsymbol{s}',o)}(\boldsymbol{a}'_{\text{M}}) + (1-\epsilon) I_{\{\pi_{\text{M}}(\boldsymbol{s}',\boldsymbol{a}_{\text{M}})\}}(\boldsymbol{a}'_{\text{M}})$.

Combined Options

When working with combined options (Figure 3.c), the high level action $\boldsymbol{a}_{\text{M}}$ is a two-dimensional discrete decision variable. In this case, the master policy can be derived from the first critic twin as $\boldsymbol{a}'_{\text{M}} = \pi_{\text{M}}(\boldsymbol{s}', \boldsymbol{a}_{\text{M}}) = \arg\max_{\boldsymbol{a}'_{\text{M}}\in\mathcal{O}^{\text{AV}}_{\text{V}}(\boldsymbol{s}',o_{\text{V}})\times\mathcal{O}^{\text{AV}}_{\text{D}}(\boldsymbol{s}',o_{\text{D}})} q(\boldsymbol{s}', \boldsymbol{a}'_{\text{M}}; \boldsymbol{\theta}_{q,1})$. This policy only selects a new option upon termination of any previously active one, which becomes more clear when rewriting it as

$$\boldsymbol{a}'_{\text{M}} = \begin{cases} \arg\max\limits_{o'_{\text{V}}\in\mathcal{O}^{\text{AV}}_{\text{V}}(\boldsymbol{s}'), o'_{\text{D}}\in\mathcal{O}^{\text{AV}}_{\text{D}}(\boldsymbol{s}')} q(\boldsymbol{s}, (o'_{\text{V}}, o'_{\text{D}}); \boldsymbol{\theta}_{q,1}) & \boldsymbol{s}' \in \mathcal{T}_{o_{\text{V}}} \cap \mathcal{T}_{o_{\text{D}}} \\ (\arg\max_{o'_{\text{V}}\in\mathcal{O}^{\text{AV}}_{\text{V}}(\boldsymbol{s}')} q(\boldsymbol{s}', (o'_{\text{V}}, o_{\text{D}}); \boldsymbol{\theta}_{q,1}), o_{\text{D}}) & \boldsymbol{s}' \in \mathcal{T}_{o_{\text{V}}} \setminus \mathcal{T}_{o_{\text{D}}} \\ (o_{\text{V}}, \arg\max_{o'_{\text{D}}\in\mathcal{O}^{\text{AV}}_{\text{D}}(\boldsymbol{s}')} q(\boldsymbol{s}', (o_{\text{V}}, o'_{\text{D}}); \boldsymbol{\theta}_{q,1})) & \boldsymbol{s}' \in \mathcal{T}_{o_{\text{D}}} \setminus \mathcal{T}_{o_{\text{V}}} \\ (o_{\text{V}}, o_{\text{D}}) & \text{otherwise} \end{cases}.$$

The behavioural master policy β_{M} is the ϵ-greedy policy derived from π_{M}, taking into account the option availabilities for the random selection, i.e. $\beta_{\text{M}}(\boldsymbol{a}'_{\text{M}} \mid \boldsymbol{s}', \boldsymbol{a}_{\text{M}}) = \epsilon|\mathcal{O}^{\text{AV}}_{\text{V}}(\boldsymbol{s}', o_{\text{V}}) \times \mathcal{O}^{\text{AV}}_{\text{D}}(\boldsymbol{s}', o_{\text{D}})|^{-1} I_{\mathcal{O}^{\text{AV}}_{\text{V}}(\boldsymbol{s}',o_{\text{V}})\times\mathcal{O}^{\text{AV}}_{\text{D}}(\boldsymbol{s}',o_{\text{D}})}(\boldsymbol{a}'_{\text{M}}) + (1-\epsilon) I_{\{\pi_{\text{M}}(\boldsymbol{s}',\boldsymbol{a}_{\text{M}})\}}(\boldsymbol{a}'_{\text{M}})$.

Hybrid Options

The implementation for hybrid options (Figure 3.d) is additionally based on the P-DQN method [43] and has a special (hybrid) structure for the critic networks (see Figure 2). Similar to classical TD3 (used in the baseline setup), the calculation of target values y_t is slightly altered to perform target policy smoothing [12]. Furthermore, the state-dependent action bounds methodology [10] is applied to ensure the selected continuous (velocity) parameters are safe.

In this setup, there is an extra actor model for selecting the continuous parameters, explicitly modeling the continuous part of the master policy. More precisely, the actor network provides the normalized parameter $\Delta\tilde{v} = \mu(\boldsymbol{s}; \boldsymbol{\theta}_\mu) \in [-1, 1]$, which is rescaled to $\Delta v = \pi_{\text{M,C}}(\boldsymbol{s}) = \sigma_{\text{pwl}}(\Delta\tilde{v}; \boldsymbol{s})$ using the piecewise linear rescaling function with state-dependent action bounds $[\Delta v_{\text{L}}(\boldsymbol{s}), \Delta v_{\text{U}}(\boldsymbol{s})]$ [10]. This actor model is trained by minimizing the actor loss $L_\mu(\boldsymbol{\theta}_\mu) = -\mathbb{E}_{\boldsymbol{s}_t\sim\mathcal{B}}[\sum_{o_{\text{D}}\in\mathcal{O}^{\text{AV}}_{\text{D}}(\boldsymbol{s}_t)} q(\boldsymbol{s}_t, (\mu(\boldsymbol{s}_t; \boldsymbol{\theta}_\mu), o_{\text{D}}); \boldsymbol{\theta}_{q,1})]$. The discrete part of the master policy is implicitly derived from the first critic model as $o'_{\text{D}} = \pi_{\text{M,D}}(\boldsymbol{s}', o_{\text{D}}) = \arg\max_{o'_{\text{D}}\in\mathcal{O}^{\text{AV}}_{\text{D}}(\boldsymbol{s}',o_{\text{D}})} q(\boldsymbol{s}', (\mu(\boldsymbol{s}'; \boldsymbol{\theta}_\mu), o'_{\text{D}}); \boldsymbol{\theta}_{q,1})$, or equivalently

$$o_{\text{D},t+1} = \begin{cases} \arg\max\limits_{o'_{\text{D}}\in\mathcal{O}^{\text{AV}}_{\text{D}}(\boldsymbol{s}_{t+1})} q(\boldsymbol{s}_{t+1}, (\Delta\tilde{v}_{t+1}, o'_{\text{D}}); \boldsymbol{\theta}_{q,1}) & \boldsymbol{s}_{t+1} \in \mathcal{T}_{o_{\text{D},t}} \\ o_{\text{D},t} & \text{otherwise} \end{cases}.$$

The behavioural master policy β_{M} can similarly be split in a continuous and discrete part. For the continuous part, the truncated Gaussian behavioural policy $\mathcal{N}^1_{-1}(\mu(\boldsymbol{s}; \boldsymbol{\theta}_\mu), \sigma^2_\epsilon)$ is used. For the discrete part, the ϵ-greedy policy derived from $\pi_{\text{M,D}}$ is used, taking into account each of the option availabilities for the random selection, i.e. $\beta_{\text{M,D}}(o'_{\text{D}} \mid \boldsymbol{s}', o_{\text{D}}) = \epsilon|\mathcal{O}^{\text{AV}}_{\text{D}}(\boldsymbol{s}', o_{\text{D}})|^{-1} I_{\mathcal{O}^{\text{AV}}_{\text{D}}(\boldsymbol{s}',o_{\text{D}})}(o'_{\text{D}}) + (1-\epsilon) I_{\{\pi_{\text{M,D}}(\boldsymbol{s}',o_{\text{D}})\}}(o'_{\text{D}})$.

5 Experiments

All presented methods for hierarchical control with options are compared with baseline policies in the simulated autonomous driving environment described in Section 3. More specifically, 4 different RL policies are trained: one baseline policy over continuous actions, and three master policies

over single, combined, and hybrid options (see Figure 3). The baseline policy over continuous actions is trained using STD3 [9] (for improved smoothness) with enforced state-dependent action bounds [10] (for safety). To make the comparisons fair, both the smoothness regularization and action bounds are also applied to the continuous component of the hybrid master policy, as illustrated in Algorithm 4. An overview of the used hyperparameters can be found in Appendix C.

For every configuration, $R = 10$ policies are trained, each using a different seed for initialization of the training process. Throughout training, $E = 10$ independent evaluation episodes are executed to obtain an estimate of the learned policy's average performance. Out of all R repeats, the best performing policy is selected. This selection is based on the average evaluation reward and only considers policies obtained after 2/3 of the training process has completed, which guarantees a good performance-smoothness trade-off for the policies over continuous actions. This best policy is separately evaluated on some additional benchmark scenarios for autonomous driving and extensively tested under varying traffic conditions.

5.1 Training and Evaluation

The average rewards for each of the RE evaluation episodes and for every configuration are aggregated and plotted (using exponential average smoothing with $\alpha = 0.1$) in Figure 4. It is clear the hybrid options approach outperforms the other policies, as it has both a higher average reward and lower variance between the different repeated experiments. Both the continuous and hybrid policies quickly converge, after approximately $2 \cdot 10^5$ training timesteps, whereas the single options approach converges much more slowly and might not have fully converged yet at the end of training. Looking at the wall clock time (using the same hardware for training), we observe the fastest training time for the continuous policies. Our implementations of the hierarchical approaches seem to be roughly 10% (single), 20% (combined) and 30% (hybrid) slower on average, although further optimizations could potentially accelerate training. Appendix B additionally visualizes the activity of options during evaluation, which underlines the increased interpretability of the option methods. Finally, it should be stressed that all trained policies are completely safe and cause no crashes (in the used simulation environment) both during training and evaluation.

5.2 Benchmark Results

The best performing policies for each configuration are separately evaluated on some typical driving tasks and scenarios. Figure 5 shows the observed driving behaviour for the simple scenario of overtaking one slower vehicle in front of the ego vehicle. From the lateral position plot, we can immediately see the benefit of the hierarchical policies over options. By using constrained option policies for lateral control, we obtain better alignment within the lane and much more desirable lane change manoeuvres, with more

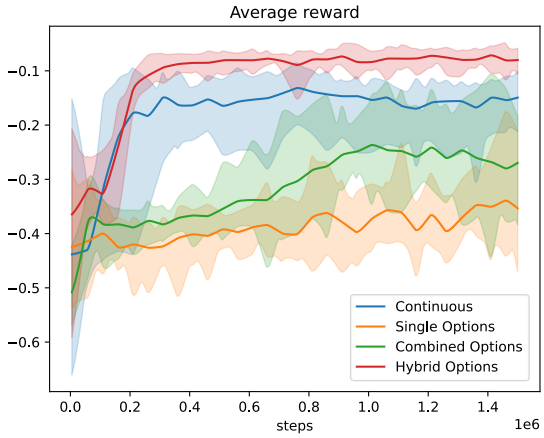

Figure 4: Summarized evaluation performance for each policy throughout training. Solid lines represent the mean value over 10 repeated experiments, the shaded areas denote the minimum and maximum values.

gradual transitions and without any overshoot. Although the smoothness regularization for the baseline policy over continuous actions already improves passenger comfort to some extent — by avoiding abrupt and oscillatory lateral movements — it is still insufficient to obtain the same kind of comfortable and consistent lane change manoeuvres as for the option approaches.

Looking carefully at the velocity plot, some interesting properties of the various option setups are revealed. The velocity profiles for the policies over single and combined options are less smooth and contain some 'discrete jumps', reducing passenger comfort. This phenomenon is caused by the particular definition of the options for longitudinal control, which modify the velocity in fixed increments δ_v to adhere to the Markov property. Some additional experiments with non-Markov options for longitudinal control (with more flexible velocity adaptation to avoid this inherent drawback), yielded promising initial results. Slightly harder to spot is the different operation of policies over single options compared to policies over combined options. The former policies can only perform one longitudinal *or* lateral manoeuvre at once, resulting in a constant velocity profile during the lane changes. The latter policies can combine both a longitudinal *and* lateral manoeuvre at the same time, resulting in slightly faster and more efficient movements on the road. Overall the policy over hybrid options scores the best, with the smoothest velocity and position profiles on this overtaking benchmark.

5.3 Test Results

Finally, the best performing policies for each configuration are also extensively tested under various traffic conditions, ranging from very calm to dense traffic scenarios, and compared with a conservative baseline policy following IDM and MOBIL. These test results are summarized in Figure 6, with additional plots provided in Appendix B.

Overall, all policies have very comparable performances and outperform the IDM/MOBIL policy in all circum-

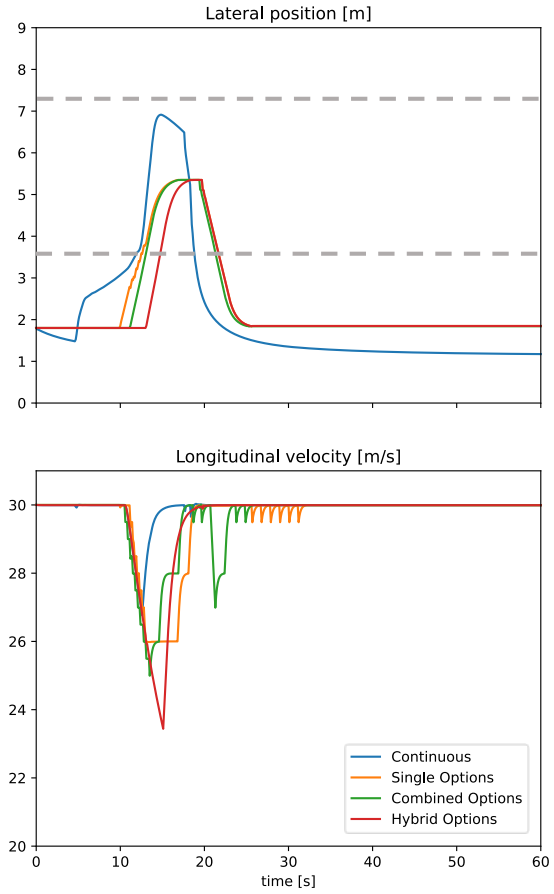

Figure 5: Lateral position and longitudinal velocity plots for each of the selected policies applied to the overtaking benchmark.

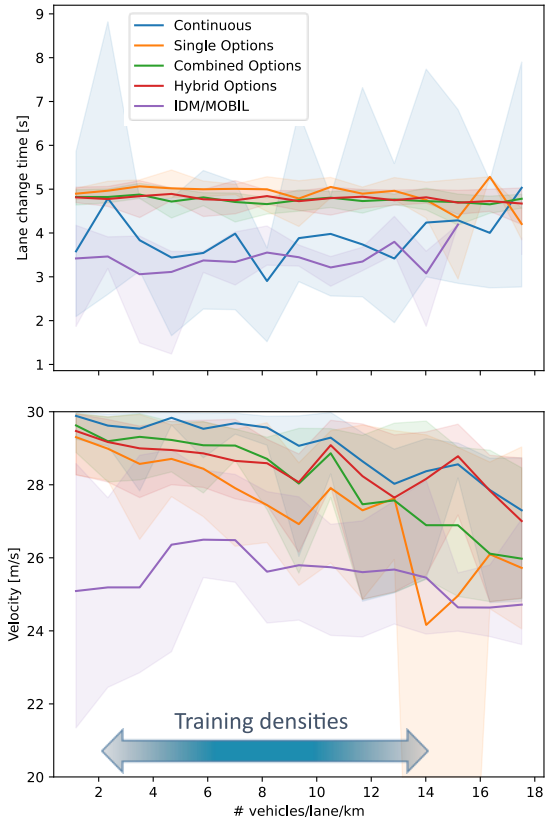

Figure 6: Evaluation of the selected policies in various traffic conditions and comparison with a conservative IDM/MOBIL policy. Solid lines represent the mean value over 10 episodes, the shaded areas denote the minimum and maximum values.

stances. The baseline policy over continuous actions and policy over hybrid options seem to outperform the other policies at higher traffic densities (more congestion), not seen during training. As expected, all hierarchical approaches with constrained options perform more reliable and consistent lane change manoeuvres. In particular, the lane change time is almost constant and takes around 5 s, complying with the intended comfort constraints. In contrast, the baseline policy over continuous actions exhibits highly variable lane change times, with lane change manoeuvres that are often too abrupt.

6 Related Work

Hierarchical and Hybrid Reinforcement Learning

Various reinforcement learning methods to deal with hierarchical and hybrid action spaces have been proposed, mainly differing in the specific structure of the action space or the considered application. Neunert et al. [28] provide a general algorithm for finding optimal hybrid policies with mixed continuous and discrete action spaces. In other works, the focus lies often on hybrid action spaces with a more particular structure, such as parameterized action spaces in PAMDPs [11, 23]. This is also the setting considered in this paper for the hybrid options. Additionally, many works on hierarchical reinforcement learning can be interpreted as finding policies over hybrid action spaces, with discrete action selections at the higher levels and continuous action selections at the lower levels (see for example Figure 3).

An overview of the broader field of hierarchical reinforcement learning is given by Pateria et al. [29] and Hutsebaut-Buysse et al. [15]. The options (or skills) framework considered in this work is only one specific subfield. The focus in most recent works lies on *learning* a suitable set of options from scratch [4, 44], or learning a policy that can achieve various subgoals [19, 20], or discovering useful skills from large datasets of interaction data [31, 37]. In contrast, in this work we specifically predefine a small set of options, tailored to the considered autonomous driving tasks, and *apply* them in various hierarchical control setups. From this perspective, our methodology is similar to the works of Neumann et al. [27] and Dalal et al. [7], which learn a master policy for selecting the parameters of predefined parameterized skills. Although we apply more recent deep RL techniques and apply intra-option learning, whereas these works and others [19, 38] typically perform Semi-MDP (option-to-option) updates.

Barreto et al. [5] and Peng et al. [30] introduce algorithms that allow the agent to execute a combination of learned

options. Both approaches are more generic than our proposed method for hierarchical control with combined options. More precisely, these methods compose new options acting on the whole (lower level) action space, whereas our approach specifically combines options acting on independent subsets of the action space (dedicated options for longitudinal and lateral control).

Reinforcement Learning for Autonomous Driving

Our methodology is most comparable with the work of Naveed et al. [26] and Wang et al. [42], which apply deep hierarchical RL to learn parameters for dedicated parameterized trajectory planners. A similar strategy is also followed by Chen et al. [6], who first train options for dedicated manoeuvres, and afterwards apply a policy gradient method to learn the master policy over options. Instead of working with dedicated manoeuvre options, many other works try to learn options with a fixed time horizon from large datasets or from interactions with the environment, which are afterwards used to train a master policy in the Semi-MDP setting [13, 21, 45]. An important difference with our work is that most of these methods use skills with a fixed time horizon and perform Semi-MDP (option-to-option) updates. In contrast, our options are predefined and have variable length, while the master policies are trained using intra-option updates. This is crucial to support combined or hybrid options, and allows to seamlessly combine slow lane change manoeuvres with fast acceleration manoeuvres.

7 Conclusion

This paper introduced several options tailored to the autonomous driving problem, and showed how they can be applied in various hierarchical control architectures. The resulting framework leads to a very natural representation of the problem, with dedicated options for longitudinal and lateral manoeuvres of varying durations, and separate decision-making modules operating at different timescales and levels in the control hierarchy. In such a setup the learned driving behaviour can also be constrained more easily, as safety and comfort constraints can be individually imposed on the options implementing the manoeuvres. As a result, even though option selection happens at coarser (Semi-MDP) timescales, we can still maintain strong safety guarantees at the finest (MDP) timescale. Furthermore, in the proposed framework, actions for longitudinal and lateral control can be selected separately, either by options or classical policies, as exemplified by the more flexible policies over combined or hybrid options. Ultimately, our contributions can lead to more interpretable, reliable, and trustworthy policies for autonomous driving.

Nevertheless, there are still many interesting open directions for future research. For example, proper support for interrupting options could be added. Some smaller experiments with interruption showed frequent abortions of lane change manoeuvres; hence, such an approach should add appropriate deliberation costs on option interruption. Another promising direction for future research would be a more complete implementation of intra-option learning. In our current implementation, the intra-option updates are only applied for active options. Truly updating the value estimates for all options that would take the same action as the active option, could thus further accelerate the learning process. Finally, it would be interesting to investigate parameterizable option policies for generic manoeuvres, such as a lane change manoeuvre with a variable duration parameter, similar to the parameterized trajectories of Wang et al. [42]. Concurrently, additional (more complex) value constraints [1] or dual policy constraints [8] could be imposed on the driving policies to further improve their performance and reliability.

Acknowledgments

This research was partially supported by Ford, under Ford Alliance Project KUL0076; and TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215. This research received funding from the Flemish Government (AI Research Program). Johan Suykens is also affiliated to Leuven.AI - KU Leuven institute for AI, B-3000, Leuven, Belgium. The research leading to these results has received funding from the European Research Council under the European Union's Horizon 2020 research and innovation program / ERC Advanced Grant E-DUALITY (787960). This paper reflects only the authors' views and the Union is not liable for any use that may be made of the contained information. The resources and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation - Flanders (FWO) and the Flemish Government.

References

[1] J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained Policy Optimization. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, volume 70, pages 22–31. PMLR, July 2017. URL https://proceedings.mlr.press/v70/achiam17a.html.

[2] A. Alizadeh, M. Moghadam, Y. Bicer, N. K. Ure, U. Yavas, and C. Kurtulus. Automated Lane Change Decision Making using Deep Reinforcement Learning in Dynamic and Uncertain Highway Environment. In *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019*, pages 1399–1404, 2019. ISBN 978-1-5386-7024-8. doi:10.1109/ITSC.2019.8917192.

[3] S. Aradi. Survey of Deep Reinforcement Learning for Motion Planning of Autonomous Vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23(2):740–759, Feb. 2022. ISSN 1558-0016. doi:10.1109/TITS.2020.3024655.

[4] P.-L. Bacon, J. Harb, and D. Precup. The Option-Critic Architecture. In *Proceedings of the 31st*

AAAI Conference on Artificial Intelligence, AAAI 2017, volume 31, pages 1726–1734, Feb. 2017. doi:10.1609/aaai.v31i1.10916.

[5] A. Barreto, D. Borsa, S. Hou, G. Comanici, E. Aygün, P. Hamel, D. Toyama, J. hunt, S. Mourad, D. Silver, and D. Precup. The Option Keyboard: Combining Skills in Reinforcement Learning. In *Advances in Neural Information Processing Systems 32, NeurIPS 2019*, volume 32, pages 13052–13062. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/hash/251c5ffd6b62cc21c446c963c76cf214-Abstract.html.

[6] J. Chen, Z. Wang, and M. Tomizuka. Deep Hierarchical Reinforcement Learning for Autonomous Driving with Distinct Behaviors. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1239–1244, June 2018. doi:10.1109/IVS.2018.8500368.

[7] M. Dalal, D. Pathak, and R. R. Salakhutdinov. Accelerating Robotic Reinforcement Learning via Parameterized Action Primitives. In *Advances in Neural Information Processing Systems*, volume 34, pages 21847–21859. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/hash/b6846b0186a035fcc76b1b1d26fd42fa-Abstract.html.

[8] B. De Cooman and J. Suykens. A Dual Perspective of Reinforcement Learning for Imposing Policy Constraints. *IEEE Transactions on Artificial Intelligence*, pages 1–14, Apr. 2024. ISSN 2691-4581. doi:10.1109/TAI.2025.3564898.

[9] B. De Cooman, J. Suykens, and A. Ortseifen. Improving temporal smoothness of deterministic reinforcement learning policies with continuous actions. In *33rd Benelux Conference on Artificial Intelligence, BNAIC 2021*, pages 217–240, Esch-sur-Alzette, Luxembourg, Nov. 2021. URL https://lirias.kuleuven.be/3635727.

[10] B. De Cooman, J. Suykens, and A. Ortseifen. Enforcing Hard State-Dependent Action Bounds on Deep Reinforcement Learning Policies. In G. Nicosia, V. Ojha, E. La Malfa, G. La Malfa, P. Pardalos, G. Di Fatta, G. Giuffrida, and R. Umeton, editors, *Proceedings of the 8th International Conference on Machine Learning, Optimization, and Data Science, LOD 2022*, volume 13811 of *Lecture Notes in Computer Science*, pages 193–218, Cham, Mar. 2023. Springer Nature Switzerland. ISBN 978-3-031-25891-6. doi:10.1007/978-3-031-25891-6_16.

[11] Z. Fan, R. Su, W. Zhang, and Y. Yu. Hybrid Actor-Critic Reinforcement Learning in Parameterized Action Space. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 2279–2285, Macao, China, Aug. 2019. International Joint Conferences on Artificial Intelligence Organization. ISBN 978-0-9992411-4-1. doi:10.24963/ijcai.2019/316.

[12] S. Fujimoto, H. Van Hoof, and D. Meger. Addressing Function Approximation Error in Actor-Critic Methods. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, volume 80, pages 1587–1596. PMLR, July 2018. URL https://proceedings.mlr.press/v80/fujimoto18a.html.

[13] Y. Gurses, K. Buyukdemirci, and Y. Yildiz. Developing Driving Strategies Efficiently: A Skill-Based Hierarchical Reinforcement Learning Approach. *IEEE Control Systems Letters*, 8:121–126, 2024. ISSN 2475-1456. doi:10.1109/LCSYS.2024.3349511.

[14] Z. Hou, W. Liu, and A. Knoll. Safe Reinforcement Learning for Autonomous Driving by Using Disturbance-Observer-Based Control Barrier Functions. *IEEE Transactions on Intelligent Vehicles*, 10(6):3782–3791, June 2025. ISSN 2379-8904. doi:10.1109/TIV.2024.3463468.

[15] M. Hutsebaut-Buysse, K. Mets, and S. Latré. Hierarchical Reinforcement Learning: A Survey and Open Research Challenges. *Machine Learning and Knowledge Extraction*, 4(1):172–221, Mar. 2022. ISSN 2504-4990. doi:10.3390/make4010009.

[16] G. Kalweit, M. Huegle, M. Werling, and J. Boedecker. Deep Constrained Q-learning, Sept. 2020.

[17] A. Kesting, M. Treiber, and D. Helbing. General Lane-Changing Model MOBIL for Car-Following Models. *Transportation Research Record*, 1999(1):86–94, Jan. 2007. ISSN 0361-1981. doi:10.3141/1999-10.

[18] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Sallab, S. Yogamani, and P. Perez. Deep Reinforcement Learning for Autonomous Driving: A Survey. *IEEE Transactions on Intelligent Transportation Systems*, 2021. doi:10.1109/TITS.2021.3054625.

[19] T. D. Kulkarni, K. R. Narasimhan, A. Saeedi, and J. B. Tenenbaum. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. In *Advances in Neural Information Processing Systems 29, NIPS 2016*, volume 29, pages 3675–3683. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/hash/f442d33fa06832082290ad8544a8da27-Abstract.html.

[20] A. Levy, G. Konidaris, R. Platt, and K. Saenko. Learning Multi-Level Hierarchies with Hindsight. In *7th International Conference on Learning Representations, ICLR 2019*, Sept. 2019. doi:10.48550/arXiv.1712.00948.

[21] Z. Li, F. Nie, Q. Sun, F. Da, and H. Zhao. Boosting Offline Reinforcement Learning for Autonomous Driving with Hierarchical Latent Skills. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 18362–18369, May 2024. doi:10.1109/ICRA57147.2024.10611197.

[22] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016*, 2016. doi:10.48550/arXiv.1509.02971.

[23] W. Masson, P. Ranchod, and G. Konidaris. Reinforcement Learning with Parameterized Actions. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence, AAAI 2016*, volume 30, Feb. 2016. doi:10.1609/aaai.v30i1.10226.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb. 2015. ISSN 1476-4687. doi:10.1038/nature14236.

[25] S. Nageshrao, H. E. Tseng, and D. Filev. Autonomous Highway Driving using Deep Reinforcement Learning. In *Proceedings of the 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 2326–2331, Oct. 2019. doi:10.1109/SMC.2019.8914621.

[26] K. B. Naveed, Z. Qiao, and J. M. Dolan. Trajectory Planning for Autonomous Vehicles Using Hierarchical Reinforcement Learning. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 601–606, Sept. 2021. doi:10.1109/ITSC48978.2021.9564634.

[27] G. Neumann, W. Maass, and J. Peters. Learning complex motions by sequencing simpler motion templates. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009*, ICML '09, pages 753–760, New York, NY, USA, June 2009. Association for Computing Machinery. ISBN 978-1-60558-516-1. doi:10.1145/1553374.1553471.

[28] M. Neunert, A. Abdolmaleki, M. Wulfmeier, T. Lampe, T. Springenberg, R. Hafner, F. Romano, J. Buchli, N. Heess, and M. Riedmiller. Continuous-Discrete Reinforcement Learning for Hybrid Control in Robotics. In *Proceedings of the Conference on Robot Learning*, volume 100, pages 735–751. PMLR, May 2020. URL https://proceedings.mlr.press/v100/neunert20a.html.

[29] S. Pateria, B. Subagdja, A.-h. Tan, and C. Quek. Hierarchical Reinforcement Learning: A Comprehensive Survey. *ACM Comput. Surv.*, 54(5):109:1–109:35, June 2021. ISSN 0360-0300. doi:10.1145/3453160.

[30] X. B. Peng, M. Chang, G. Zhang, P. Abbeel, and S. Levine. MCP: Learning Composable Hierarchical Control with Multiplicative Compositional Policies. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/hash/95192c98732387165bf8e396c0f2dad2-Abstract.html.

[31] K. Pertsch, Y. Lee, and J. Lim. Accelerating Reinforcement Learning with Learned Skill Priors. In *Proceedings of the 2020 Conference on Robot Learning*, volume 155, pages 188–204. PMLR, Oct. 2021. URL https://proceedings.mlr.press/v155/pertsch21a.html.

[32] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Jan. 1994. ISBN 978-1-118-62587-3. URL https://books.google.com/books?id=VvBjBAAAQBAJ.

[33] R. Rajamani. *Vehicle Dynamics and Control*. Mechanical Engineering Series. Springer US, Boston, MA, 2nd edition, 2012. ISBN 978-1-4614-1432-2. doi:10.1007/978-1-4614-1433-9.

[34] D. M. Saxena, S. Bae, A. Nakhaei, K. Fujimura, and M. Likhachev. Driving in Dense Traffic with Model-Free Reinforcement Learning. In *Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5385–5392, May 2020. doi:10.1109/ICRA40945.2020.9197132.

[35] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*, volume 258 of *A Bradford Book*. MIT Press, 2nd edition, Nov. 2018. ISBN 978-0-262-35270-3. URL https://books.google.be/books?id=uWV0DwAAQBAJ.

[36] R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, Aug. 1999. ISSN 0004-3702. doi:10.1016/S0004-3702(99)00052-1.

[37] D. Tanneberg, K. Ploeger, E. Rueckert, and J. Peters. SKID RAW: Skill Discovery From Raw Trajectories. *IEEE Robotics and Automation Letters*, 6(3):4696–4703, July 2021. ISSN 2377-3766. doi:10.1109/LRA.2021.3068891.

[38] C. Tessler, S. Givony, T. Zahavy, D. Mankowitz, and S. Mannor. A Deep Hierarchical Approach to Lifelong Learning in Minecraft. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence, AAAI 2017*, volume 31, pages 1553–1561, Feb. 2017. doi:10.1609/aaai.v31i1.10744.

[39] S. Thrun and A. Schwartz. Finding Structure in Reinforcement Learning. In *Advances in Neural Information Processing Systems 7, NIPS 1994*, volume 7, pages 385–392. MIT Press, 1994. URL https://proceedings.neurips.cc/paper/1994/hash/7ce3284b743aefde80ffd9aec500e085-Abstract.html.

[40] M. Treiber, A. Hennecke, and D. Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 62(2):1805–1824, Aug. 2000. doi:10.1103/PhysRevE.62.1805.

[41] H. Van Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence, AAAI 2016*, volume 30, pages 2094–2100, Mar. 2016. doi:10.1609/aaai.v30i1.10295.

[42] L. Wang, J. Liu, H. Shao, W. Wang, R. Chen, Y. Liu, and S. L. Waslander. Efficient Reinforcement Learning for Autonomous Driving with Parameterized Skills and Priors, May 2023.

[43] J. Xiong, Q. Wang, Z. Yang, P. Sun, L. Han, Y. Zheng, H. Fu, T. Zhang, J. Liu, and H. Liu. Parametrized Deep Q-Networks Learning: Reinforcement Learning with Discrete-Continuous Hybrid Action Space, Oct. 2018.

[44] S. Zhang and S. Whiteson. DAC: The Double Actor-Critic Architecture for Learning Options. In *Advances in Neural Information Processing Systems 32, NeurIPS 2019*, volume 32, pages 2012–2022. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/hash/4f284803bd0966cc24fa8683a34afc6e-Abstract.html.

[45] T. Zhou, L. Wang, R. Chen, W. Wang, and Y. Liu. Accelerating Reinforcement Learning for Autonomous Driving Using Task-Agnostic and Ego-Centric Motion Skills. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11289–11296, Oct. 2023. doi:10.1109/IROS55552.2023.10341449.

Appendices

A Algorithms

Below we provide the concrete implementations of the generic Algorithm 1 for hierarchical reinforcement learning with single options (Algorithm 2), combined options (Algorithm 3) and hybrid options (Algorithm 4).

Algorithm 2 Hierarchical RL with Options

Initialize critic models $\boldsymbol{\theta}_{q,i}, \boldsymbol{\theta}'_{q,i}$
$\mathcal{B} \leftarrow \emptyset$
for each episode **do**
  for each environment step t **do**
    if $t = 0$ or $\boldsymbol{s}_t \in \mathcal{T}_{o_{t-1}}$ **then** ▷ Activate new option
      with probability ϵ **do** ▷ ϵ-greedy β_{M}
        $o_t \sim \mathrm{U}[\mathcal{O}^{\mathrm{AV}}(\boldsymbol{s}_t)]$
      else
        $o_t = \arg\max_{o \in \mathcal{O}^{\mathrm{AV}}(\boldsymbol{s}_t)} q(\boldsymbol{s}_t, o; \boldsymbol{\theta}_{q,1})$
      end
    else
      $o_t = o_{t-1}$
    end if
    $\boldsymbol{a}_t = \pi_{o_t}(\boldsymbol{s})$
    $\boldsymbol{s}_{t+1} \sim \tau(\cdot \mid \boldsymbol{s}_t, \boldsymbol{a}_t)$
    $r_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{(\boldsymbol{s}_t, o_t, r_t, \boldsymbol{s}_{t+1})\}$
  end for
  for each gradient step **do**
    Sample batch B from $\mathcal{B}$
    for all $(\boldsymbol{s}_t, o_t, r_t, \boldsymbol{s}_{t+1}) \in B$ **do**
      if $\boldsymbol{s}_{t+1} \in \mathcal{T}_{o_t}$ **then** ▷ Determine next active option
        $o_{t+1} = \arg\max_{o' \in \mathcal{O}^{\mathrm{AV}}(\boldsymbol{s}_{t+1})} q(\boldsymbol{s}_{t+1}, o'; \boldsymbol{\theta}_{q,1})$
      else
        $o_{t+1} = o_t$
      end if
      $y_t = r_t + \gamma \min_{i \in \{1,2\}} q(\boldsymbol{s}_{t+1}, o_{t+1}; \boldsymbol{\theta}'_{q,i})$
    end for
    $\widehat{L}_q(\boldsymbol{\theta}_{q,i}; B) = \frac{1}{|B|} \sum_B \big(q(\boldsymbol{s}_t, o_t; \boldsymbol{\theta}_{q,i}) - y_t\big)^2$
    $\boldsymbol{\theta}_{q,i} \leftarrow \boldsymbol{\theta}_{q,i} - \eta_q \nabla_{\boldsymbol{\theta}_{q,i}} \widehat{L}_q(\boldsymbol{\theta}_{q,i}; B)$
    $\boldsymbol{\theta}'_m \leftarrow \tau \boldsymbol{\theta}_m + (1-\tau)\boldsymbol{\theta}'_m$
  end for
end for

Algorithm 3 Hierarchical RL with Combined Options

Initialize critic models $\boldsymbol{\theta}_{q,i}, \boldsymbol{\theta}'_{q,i}$
$\mathcal{B} \leftarrow \emptyset$
for each episode **do**
  for each environment step t **do**
    if $t = 0$ or $\boldsymbol{s}_t \in \mathcal{T}_{o_{\mathrm{V},t-1}} \cap \mathcal{T}_{o_{\mathrm{D},t-1}}$ **then** ▷ ϵ-greedy β_{M}
      with probability ϵ **do**
        $(o_{\mathrm{V},t}, o_{\mathrm{D},t}) \sim \mathrm{U}[\mathcal{O}^{\mathrm{AV}}_{\mathrm{V}}(\boldsymbol{s}_t) \times \mathcal{O}^{\mathrm{AV}}_{\mathrm{D}}(\boldsymbol{s}_t)]$
      else
        $(o_{\mathrm{V},t}, o_{\mathrm{D},t}) = \underset{o_{\mathrm{V}} \in \mathcal{O}^{\mathrm{AV}}_{\mathrm{V}}(\boldsymbol{s}_t), o_{\mathrm{D}} \in \mathcal{O}^{\mathrm{AV}}_{\mathrm{D}}(\boldsymbol{s}_t)}{\arg\max} q(\boldsymbol{s}_t, (o_{\mathrm{V}}, o_{\mathrm{D}}); \boldsymbol{\theta}_{q,1})$
      end
    else if $\boldsymbol{s}_t \in \mathcal{T}_{o_{\mathrm{V},t-1}}$ **then**
      $o_{\mathrm{D},t} = o_{\mathrm{D},t-1}$
      with probability ϵ **do**
        $o_{\mathrm{V},t} \sim \mathrm{U}[\mathcal{O}^{\mathrm{AV}}_{\mathrm{V}}(\boldsymbol{s}_t)]$
      else
        $o_{\mathrm{V},t} = \arg\max_{o_{\mathrm{V}} \in \mathcal{O}^{\mathrm{AV}}_{\mathrm{V}}(\boldsymbol{s}_t)} q(\boldsymbol{s}_t, (o_{\mathrm{V}}, o_{\mathrm{D},t}); \boldsymbol{\theta}_{q,1})$
      end
    else if $\boldsymbol{s}_t \in \mathcal{T}_{o_{\mathrm{D},t-1}}$ **then**
      $o_{\mathrm{V},t} = o_{\mathrm{V},t-1}$
      with probability ϵ **do**
        $o_{\mathrm{D},t} \sim \mathrm{U}[\mathcal{O}^{\mathrm{AV}}_{\mathrm{D}}(\boldsymbol{s}_t)]$
      else
        $o_{\mathrm{D},t} = \arg\max_{o_{\mathrm{D}} \in \mathcal{O}^{\mathrm{AV}}_{\mathrm{D}}(\boldsymbol{s}_t)} q(\boldsymbol{s}_t, (o_{\mathrm{V},t}, o_{\mathrm{D}}); \boldsymbol{\theta}_{q,1})$
      end
    else
      $(o_{\mathrm{V},t}, o_{\mathrm{D},t}) = (o_{\mathrm{V},t-1}, o_{\mathrm{D},t-1})$
    end if
    $\boldsymbol{a}_t = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \pi_{o_{\mathrm{V},t}}(\boldsymbol{s}) + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \pi_{o_{\mathrm{D},t}}(\boldsymbol{s})$
    $\boldsymbol{s}_{t+1} \sim \tau(\cdot \mid \boldsymbol{s}_t, \boldsymbol{a}_t)$
    $r_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{(\boldsymbol{s}_t, (o_{\mathrm{V},t}, o_{\mathrm{D},t}), r_t, \boldsymbol{s}_{t+1})\}$
  end for
  for each gradient step **do**
    Sample batch B from $\mathcal{B}$
    for all $(\boldsymbol{s}_t, (o_{\mathrm{V},t}, o_{\mathrm{D},t}), r_t, \boldsymbol{s}_{t+1}) \in B$ **do**
      if $\boldsymbol{s}_{t+1} \in \mathcal{T}_{o_{\mathrm{V},t}} \cap \mathcal{T}_{o_{\mathrm{D},t}}$ **then** ▷ π_{M}
        $(o_{\mathrm{V},t+1}, o_{\mathrm{D},t+1}) =$
        $\underset{o'_{\mathrm{V}} \in \mathcal{O}^{\mathrm{AV}}_{\mathrm{V}}(\boldsymbol{s}_{t+1}), o'_{\mathrm{D}} \in \mathcal{O}^{\mathrm{AV}}_{\mathrm{D}}(\boldsymbol{s}_{t+1})}{\arg\max} q(\boldsymbol{s}_{t+1}, (o'_{\mathrm{V}}, o'_{\mathrm{D}}); \boldsymbol{\theta}_{q,1})$
      else if $\boldsymbol{s}_{t+1} \in \mathcal{T}_{o_{\mathrm{V},t}}$ **then**
        $o_{\mathrm{D},t+1} = o_{\mathrm{D},t}$
        $o_{\mathrm{V},t+1} = \arg\max_{o'_{\mathrm{V}} \in \mathcal{O}^{\mathrm{AV}}_{\mathrm{V}}(\boldsymbol{s}_{t+1})} q(\boldsymbol{s}_{t+1}, (o'_{\mathrm{V}}, o_{\mathrm{D},t+1}); \boldsymbol{\theta}_{q,1})$
      else if $\boldsymbol{s}_{t+1} \in \mathcal{T}_{o_{\mathrm{D},t}}$ **then**
        $o_{\mathrm{V},t+1} = o_{\mathrm{V},t}$
        $o_{\mathrm{D},t+1} = \arg\max_{o'_{\mathrm{D}} \in \mathcal{O}^{\mathrm{AV}}_{\mathrm{D}}(\boldsymbol{s}_{t+1})} q(\boldsymbol{s}_{t+1}, (o_{\mathrm{V},t+1}, o'_{\mathrm{D}}); \boldsymbol{\theta}_{q,1})$
      else
        $(o_{\mathrm{V},t+1}, o_{\mathrm{D},t+1}) = (o_{\mathrm{V},t}, o_{\mathrm{D},t})$
      end if
      $y_t = r_t + \gamma \min_{i \in \{1,2\}} q(\boldsymbol{s}_{t+1}, (o_{\mathrm{V},t+1}, o_{\mathrm{D},t+1}); \boldsymbol{\theta}'_{q,i})$
    end for
    $\widehat{L}_q(\boldsymbol{\theta}_{q,i}; B) = \frac{1}{|B|} \sum_B \big(q(\boldsymbol{s}_t, (o_{\mathrm{V},t}, o_{\mathrm{D},t}); \boldsymbol{\theta}_{q,i}) - y_t\big)^2$
    $\boldsymbol{\theta}_{q,i} \leftarrow \boldsymbol{\theta}_{q,i} - \eta_q \nabla_{\boldsymbol{\theta}_{q,i}} \widehat{L}_q(\boldsymbol{\theta}_{q,i}; B)$
    $\boldsymbol{\theta}'_m \leftarrow \tau \boldsymbol{\theta}_m + (1-\tau)\boldsymbol{\theta}'_m$
  end for
end for

Algorithm 4 Hierarchical RL with Hybrid Options

Initialize actor and critic models $\boldsymbol{\theta}_\mu, \boldsymbol{\theta}'_\mu, \boldsymbol{\theta}_{q,i}, \boldsymbol{\theta}'_{q,i}$
$\mathcal{B} \leftarrow \emptyset$
for each episode **do**
  for each environment step t **do**
    $\Delta\tilde{v}_t \sim \mathcal{N}^1_{-1}(\mu(\boldsymbol{s}_t;\boldsymbol{\theta}_\mu);\sigma^2_\epsilon)$ ▷ Truncated Gaussian $\beta_{\text{M,C}}$
    if $t = 0$ or $\boldsymbol{s}_t \in \mathcal{T}_{o_{\text{D},t-1}}$ **then**
      with probability ϵ **do** ▷ ϵ-greedy $\beta_{\text{M,D}}$
        $o_{\text{D},t} \sim \text{U}[\mathcal{O}^{\text{AV}}_{\text{D}}(\boldsymbol{s}_t)]$
      else
        $o_{\text{D},t} = \arg\max_{o_{\text{D}} \in \mathcal{O}^{\text{AV}}_{\text{D}}(\boldsymbol{s}_t)} q(\boldsymbol{s}_t, (\Delta\tilde{v}_t, o_{\text{D}}); \boldsymbol{\theta}_{q,1})$
      end
    else
      $o_{\text{D},t} = o_{\text{D},t-1}$
    end if
    $\Delta v_t = \sigma_{\text{pwl}}(\Delta\tilde{v}_t; \boldsymbol{s}_t)$ ▷ State-dependent bounds [10]
    $\Delta d_t = \begin{bmatrix}0 & 1\end{bmatrix} \pi_{o_{\text{D},t}}(\boldsymbol{s}_t)$
    $\boldsymbol{a}_t = \begin{bmatrix}\Delta v_t; \Delta d_t\end{bmatrix}$
    $\boldsymbol{s}_{t+1} \sim \tau(\cdot \mid \boldsymbol{s}_t, \boldsymbol{a}_t)$
    $r_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{(\boldsymbol{s}_t, (\Delta\tilde{v}_t, o_{\text{D},t}), r_t, \boldsymbol{s}_{t+1})\}$
  end for
  for each gradient step **do**
    Sample batch B from $\mathcal{B}$
    for all $(\boldsymbol{s}_t, (\Delta\tilde{v}_t, o_{\text{D},t}), r_t, \boldsymbol{s}_{t+1}) \in B$ **do**
      $\Delta\tilde{v}_{t+1} = \mu(\boldsymbol{s}_{t+1}; \boldsymbol{\theta}'_\mu)$ ▷ $\pi'_{\text{M,C}}$
      $\epsilon_{\text{c}} \sim \mathcal{N}(0, \sigma^2_{\text{c}})|_{[-c,\,c]}$ ▷ Target policy smoothing [12]
      $\Delta\tilde{v}_{t+1} \leftarrow \Delta\tilde{v}_{t+1} + \epsilon_{\text{c}}$
      if $\boldsymbol{s}_{t+1} \in \mathcal{T}_{o_{\text{D},t}}$ **then** ▷ $\pi_{\text{M,D}}$
        $o_{\text{D},t+1} = \arg\max_{o'_{\text{D}} \in \mathcal{O}^{\text{AV}}_{\text{D}}(\boldsymbol{s}_{t+1})} q(\boldsymbol{s}_{t+1}, (\Delta\tilde{v}_{t+1}, o'_{\text{D}}); \boldsymbol{\theta}_{q,1})$
      else
        $o_{\text{D},t+1} = o_{\text{D},t}$
      end if
      $y_t = r_t + \gamma \min_{i\in\{1,2\}} q(\boldsymbol{s}_{t+1}, (\Delta\tilde{v}_{t+1}, o_{\text{D},t+1}); \boldsymbol{\theta}'_{q,i})$
    end for
    $\widehat{L}_q(\boldsymbol{\theta}_{q,i}; B) = \frac{1}{|B|}\sum_B \big(q(\boldsymbol{s}_t, (\Delta\tilde{v}_t, o_{\text{D},t}); \boldsymbol{\theta}_{q,i}) - y_t\big)^2$
    $\widehat{L}_\mu(\boldsymbol{\theta}_\mu; B) = -\frac{1}{|B|}\sum_B \sum_{o_{\text{D}} \in \mathcal{O}^{\text{AV}}_{\text{D}}(\boldsymbol{s}_t)} q(\boldsymbol{s}_t, (\mu(\boldsymbol{s}_t;\boldsymbol{\theta}_\mu), o_{\text{D}}); \boldsymbol{\theta}_{q,1})$
    $\widehat{R}_{\text{s}}(\boldsymbol{\theta}_\mu; B) = \frac{1}{|B|}\sum_B \big(\mu(\boldsymbol{s}_{t+1};\boldsymbol{\theta}_\mu) - \Delta\tilde{v}_t\big)^2$
    ▷ Smoothness regularization [9]
    $\boldsymbol{\theta}_{q,i} \leftarrow \boldsymbol{\theta}_{q,i} - \eta_q \nabla_{\boldsymbol{\theta}_{q,i}} \widehat{L}_q(\boldsymbol{\theta}_{q,i}; B)$
    $\boldsymbol{\theta}_\mu \leftarrow \boldsymbol{\theta}_\mu - \eta_\mu \nabla_{\boldsymbol{\theta}_\mu} \big(\widehat{L}_\mu(\boldsymbol{\theta}_\mu; B) + \lambda_{\text{s}} \widehat{R}_{\text{s}}(\boldsymbol{\theta}_\mu; B)\big)$
    $\boldsymbol{\theta}'_m \leftarrow \tau\boldsymbol{\theta}_m + (1-\tau)\boldsymbol{\theta}'_m$
  end for
end for

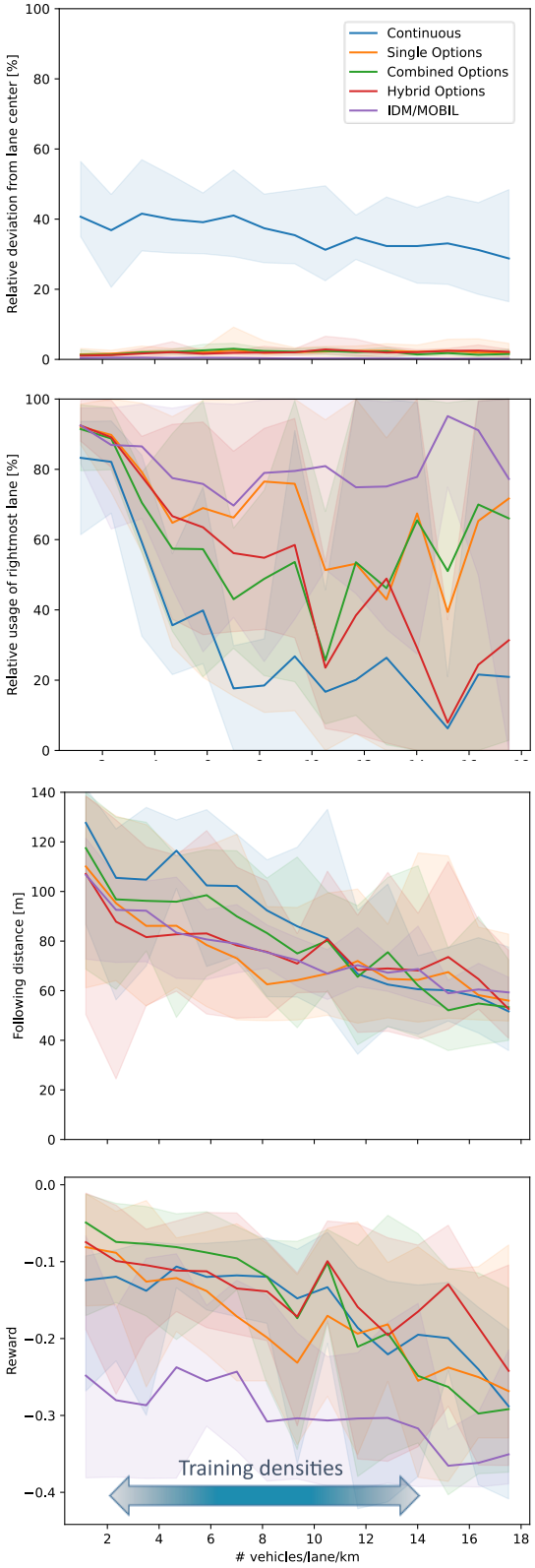

Figure 7: Evaluation of the selected policies in various traffic conditions and comparison with a conservative IDM/MOBIL policy. Solid lines represent the mean value over 10 episodes, the shaded areas denote the minimum and maximum values.

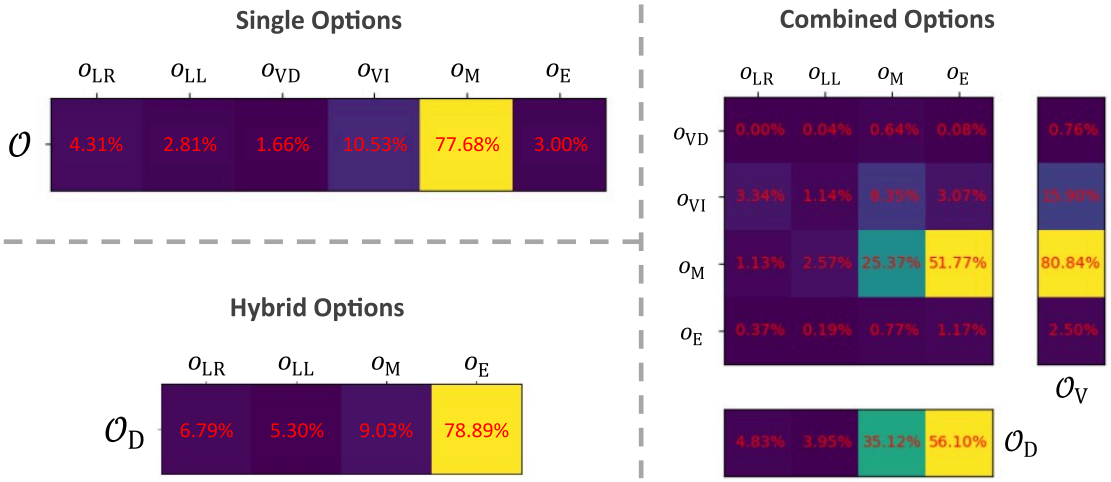

Figure 8: Option activity for the selected policies over options during $E = 10$ evaluation episodes. The percentages denote the fraction of time spent with a certain option active.

B Experimental Results

Some additional test metrics for the selected policies are shown in Figure 7, which further supports the observations made in Section 5. In particular, the rewards and following distances of all policies are quite similar and roughly follow the same trends. In more dense traffic conditions, the rightmost (slower) lane is used less frequently by the baseline policy over continuous actions and master policy over hybrid options. This confirms their slight outperformance compared to other approaches in more congested traffic situations. Finally, due to the use of constrained options for lateral control, the hierarchical setups achieve better alignment within the lane with almost no deviation from lane center.

Figure 8 shows the activity of options during evaluation of the selected (best performing) policies. These plots can be used to analyze the learned driving behaviour in some more detail, and thus increase the interpretability of the learned models as compared to the continuous baseline models. For example, we can observe that the policy over combined options effectively uses the flexibility of combining longitudinal and lateral manoeuvres, as it actively chooses to alter the vehicle's velocity during lane changes $\sim 50\%$ of the time. Performing such an analysis or determining which manoeuvres are being performed, is much harder for policies over continuous actions. In fact, only the immediate reference signals selected by a policy over continuous actions are known at any moment in time. In contrast, for the policies over options the exact manoeuvre selected by the master policy is known at every moment in time (and can be determined for any possible traffic situation on the road).

Remark that for lateral control, the maintain and emergency options both provide the same action to maintain the lateral position (unless this would be unsafe). It seems the selected master policies have learned a slight preference for the (lateral) emergency option, which is unintended. This can be easily resolved by adding a small penalty to the reward signal for activating this emergency option. Results for experiments with such a reward modification are however not shown in this work to make a fair comparison between the different approaches easier (using the exact same reward for all experiments).

C Hyperparameters

Table 2 below shows the used hyperparameters for the performed experiments.

Table 2: Overview of the used hyperparameters.

Hyperparameters		**Values**
Maximum episode length (truncation)	T	5000
Total training episodes		300
Warmup timesteps		6400
Replay buffer size	$\lvert\mathcal{B}\rvert$	1 000 000
Batch size	$\lvert B\rvert$	64
Discount factor	γ	0.99
Learning rates (strides)	η_q	$5 \cdot 10^{-4}$ (1)
	η_μ	$5 \cdot 10^{-4}$ (2)
Target networks averaging constant (stride)	τ	$1 \cdot 10^{-3}$ (2)
Network architecture – hidden dimensions	(critic)	64×32
	(actor)	$32 \times 16 \times 8$