2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

Amortized Q-learning with Model-based Action Proposals for Autonomous Driving on Highways

Branka Mirchevska[1,2], Maria Hügle[1], Gabriel Kalweit[1], Moritz Werling[2], Joschka Boedecker[1,3]

***Abstract*— Well-established optimization-based methods can guarantee an optimal trajectory for a short optimization horizon, typically no longer than a few seconds. As a result, choosing the optimal trajectory for this short horizon may still result in a sub-optimal long-term solution. At the same time, the resulting short-term trajectories allow for effective, comfortable and provable safe maneuvers in a dynamic traffic environment. In this work, we address the question of how to ensure an optimal long-term driving strategy, while keeping the benefits of classical trajectory planning. We introduce a Reinforcement Learning based approach that coupled with a trajectory planner, learns an optimal long-term decision-making strategy for driving on highways. By online generating locally optimal maneuvers as actions, we balance between the infinite low-level continuous action space, and the limited flexibility of a fixed number of predefined standard lane-change actions. We evaluated our method on realistic scenarios in the open-source traffic simulator SUMO and were able to achieve better performance than the 4 benchmark approaches we compared against, including a random action selecting agent, greedy agent, high-level, discrete actions agent and an IDM-based SUMO-controlled agent.**

I. INTRODUCTION

Autonomous vehicles have to make safe and reliable decisions in highly dynamic and complex traffic environments. So far, the approaches proposed for tackling this challenge can be divided into three broad categories: rule-based systems, optimization-based (optimal control) systems and data-driven i.e machine learning-based systems. Rule-based systems [15], [24], [2], offer the advantage of greater control over their actions. However, defining a consistent set of rules that works in all possible situations under noisy observations is a difficult and error-prone procedure. Optimal control-based systems, like [29], [17], [3] and [4], which rely on constrained optimization and Model Predictive Control, encode the vehicle dynamics model directly in the planning module and are able to generate feasible and comfortable trajectories. They also typically offer sound solutions with mathematically backed up safety guarantees. However, due to the short optimization horizon, these approaches are not capable of making farsighted, globally optimal decisions [16]. Alternatively, purely machine learning-based approaches, while offering better generalization to unseen situations and learning from data, introduce safety concerns and reduce the transparency of the behavior of the system. Among machine learning methods, Reinforcement Learning (RL) has become increasingly popular in autonomous driving applications, due to the notable success in many application areas [14, 19, 20]. However, when applying RL to autonomous driving, an important challenge is to determine the level of control the agent should have over the vehicle. Most common are the discrete actions and continuous actions settings. In the discrete actions case (e.g. [1, 6, 31] and also [13, 7, 8, 9, 10]), the agent can choose from actions such as keep lane, lane-change to the left or lane-change to the right or fixed accelerating/decelerating steps. While the small and fixed action set leads to fast learning progress, the lane-change maneuvers are usually with fixed execution duration, resulting in a sub-optimal, unnatural behavior in tight situations. On the other end of the spectrum, in the continuous actions setting, the agent learns to influence either the low-level steering wheel and gas/brake pedal control signals directly, or some kind of an abstraction thereof such as curvature or acceleration [27, 25, 26]. This endows the agent with maximal freedom of choice, but since the set of actions is infinite and the choices that the agent needs to make are typically very granular, it will need large amounts of good action sequences among the learning samples, slowing down learning considerably. In addition, the maximization of the action-value function in continuous off-policy RL methods is problematic. To address that, in [30], the maximization step is simplified by a learned proposal distribution which is used to sample the continuous action space. The approximate maximum is then taken as the maximizing action from the proposed sample-set.

[1]Dept. of Computer Science, University of Freiburg, Germany.
{hueglem,kalweitg,jboedeck}@cs.uni-freiburg.de
[2]BMW Group, Unterschleissheim, Germany.
{Branka.Mirchevska,Moritz.Werling}@bmw.de
[3]Cluster of Excellence BrainLinks-BrainTools, Freiburg, Germany.

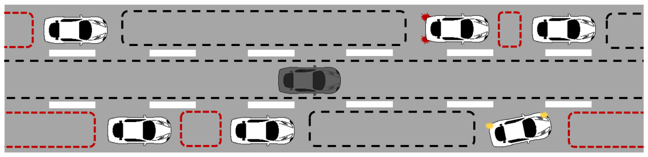
Fig. 1: RL environment: RL agent (black), surrounding vehicles (white), reachable gaps (black-dashed), unreachable gaps (red-dashed)

Inspired by this approach, we use model-based action proposals for balancing out the trade-off between continuous and discrete actions discussed above (see Figure 1 for an overview). Since the actions that we are proposing, are finite but described by continuous values, they can be seen as a compromise between the discrete and the continuous representations. This way, we combine the advantages of both sides of the spectrum, while ensuring safety. For this purpose, we introduce the notion of gaps, as model-based action proposals for the agent.

A gap is a space between two surrounding vehicles on the same lane. We define the actions as the set of all reachable gaps in the vicinity of the agent. An embedded trajectory planner plans locally optimal trajectories which satisfy all physical and safety constraints, based on which the set of reachable gaps is created. After the RL agent chooses a gap from the reachable set of gaps, the best low-level maneuver towards that gap is executed. Since, the number of gaps the agent can choose from is finite, but the features describing each gap are continuous, we allow for a sufficient level of control over the vehicle, while ensuring safety and keeping the action space and the learning times manageable.

We introduce *Amortized Q-learning with Model-based Action Proposals* (AQL-MAP), an *Amortized Q-learning* approach that replaces the learned proposal distribution by model-based action estimation. The term "amortized" in this case stems from amortizing the cost of maximization over the whole continuous action space by proposing a finite set of reachable actions to maximize over. Even though the set of proposed actions is finite, it covers all safe and feasible maneuvers the agent can execute in a given state. We implement the approach in the Semi-MDP Options-framework [21], [18] in order to speed up learning and stabilize the performance. Moreover, since for autonomous driving in real traffic, on-policy, online learning is out of the question, the training data usually needs to be pre-collected from fleets of human-operated vehicles. With that in mind, we train our approach in offline and off-policy fashion, on fixed batch of training data. Our main contributions are the following:

1) Introducing gaps as an action abstraction, described by continuous features which incorporate all possible maneuvers that the vehicle can safely perform.
2) Coupling RL with an optimization based trajectory planner for estimating the current set of reachable gaps and generating feasible and safe trajectories towards them.
3) Analysis of the achieved performance by evaluating the novel agent against 4 benchmark agents in realistic highway scenarios, defined in the SUMO simulation environment [11]. The benchmark agents include: random action selecting agent, greedy agent, IDM-based SUMO-controlled agent [23] and a high-level, discrete action selecting agent.

II. REINFORCEMENT LEARNING BACKGROUND

Reinforcement learning is concerned with the problem of learning from interaction with the environment. Formally we define a RL problem as a Markov Decision Process, consisting of: set of states $S = \{s_1, s_2, ..., s_n\}$, set of actions $A = \{a_1, a_2, ..., a_m\}$, transition function T: $T(s, a, s') = P(s(t+1) = s'|s(t) = s, a(t) = a)$ and a reward function $r(s, a)$. In a state s_t, at a time-step t, following a policy π, a RL agent executes an action a_t over the environment. Then, at a time-step $t+1$, it ends up in some state s_{t+1} and receives a reward r_{t+1}. We define $R(s_t) = \sum_{t'>=t} \gamma^{t'-t} r_{t'}$ as the expected discounted

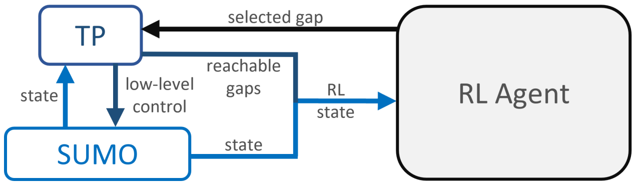

Fig. 2: Diagram of the information flow between SUMO, TP and the RL agent

long-term return, where $\gamma \in [0, 1]$ is the discount factor. The objective of the agent is to optimize its policy π in order to maximize R. One way of finding the optimal policy is by Q-learning [28], where the agent aims to learn the optimal action-value function $Q^*(s, a)$, defined as: $Q^*(s, a) = \max_\pi \mathbb{E}[R_t|s_t = s, a_t = a, \pi]$.

Since we implement our approach in the Options framework, here we briefly introduce it. Options are a generalization of primitive actions to include temporary extended courses of actions. An option is a triple $o = (I, \pi, \beta)$, where $I \subset S$ is the set of states that an option can start in (initiation set), $\pi : S \times A \to [0, 1]$ is the policy and $\beta : S^+ \to [0, 1]$ is the termination condition (S^+ is the set of all states including the terminal state if there is one). An example of an option would be the task of changing a lane, which consists of multiple primitive actions like: activating the turn signal, turning the steering wheel, stepping on the gas pedal etc. A set of options defined over an MDP constitutes a semi-Markov decision process (SMDP). Options can take variable number of steps. Given a fixed set of options O_s at state s, at a time-step t the agent chooses an option o_t. When the option terminates at time-step $t+k$, the agent is to select a new one (unlike the standard MDP setting when the agent always chooses a new primitive action at $t+1$). Using options can potentially speed up and stabilize learning on large problems. For more details refer to [21] and [22].

III. GENERAL APPROACH

The final approach consists of two main components: the trajectory planning and the learning algorithm. For generating the set of action proposals for each time-step, we adapted the sampling-based optimal control approach for trajectory planning and generation from [29]. For approximating the optimal Q-function, we use DQN-based function approximator Q, with parameters θ, in an offline, batch learning way. Namely, each training iteration, we sub-sample a mini-batch M of transition samples (s, a, r, s'), from a pre-collected training data buffer R. Given $M = (s_i, a_i, s_{i+1}, r_i)$, we optimize for a the loss function: $L(\theta) = \frac{1}{b}\sum_i (Q(s_i, a_i|\theta) - y_i)^2$, where $y_i = r_i + \gamma \max_a Q'(s_{i+1}, a_i|\theta)$ are the targets. Q' is the target network with parameters θ' which is updated every iteration by a soft update policy with parameter $\tau \in [0, 1]$. In order to reduce the overestimation of the Q-values we use two target networks, as suggested in [5]. Furthermore, we build on the DeepSet-Q approach from [7], which allows us to support variable number of inputs for our RL architecture.

A. AQL-MAP Framework

In Figure 2 the complete framework of interaction between the simulation environment (SUMO), the trajectory planner (TP) and the RL agent is shown. In each time-step, first SUMO sends the current environment state to TP. Based on it, TP plans locally optimal trajectories and creates the set of reachable gaps. The current environment state and the set of reachable gaps construct the RL state which is sent to the RL agent. After the RL agent has chosen a gap to fit into, it sends its decision to TP. Then, TP sends to SUMO, the low-level control signals necessary for reaching the selected gap. SUMO executes the controls and sends the next environment state to TP to begin a new cycle of the same process.

Using this framework, we collect the training data transitions. In RL state s_t, given the initial state of the RL agent: $s_t^{rl} = \{\text{pos}_t, \text{speed}_t, \text{accel}_t\}$, in time-step $t = 0$ considering the surrounding vehicles in the sensor range of the RL agent, the set of all surrounding gaps $G_t = \{g_{t,0}; g_{t,1}; ...; g_{t,n}\}$ is computed. Then, the TP generates all possible trajectories starting from state s_t^{rl}: $T_t = \{tr_{t,0}; tr_{t,1}; ...; tr_{t,m}\}$. By examining which gaps are reachable by the generated trajectories, we compute the set of reachable gaps: $RG_t = \{rg_{t,0}; rg_{t,1}; ...; rg_{t,k}\}$. This set is proposed to the RL agent to choose from. The RL agent then randomly chooses a gap $g_t \in RG$, upon which the TP returns the best trajectory leading to g_t. The simulator starts executing the trajectory, the RL agent arrives in state s_{t+1} and receives a reward from the environment r_t. Finally, the transition (s_t, g_t, s_{t+1}, r_t) is stored to the fixed batch $\mathcal{R}$.

B. Trajectory Planning

We plan locally optimal trajectories to each gap if reachable within a 6 s horizon, that satisfy physical and safety constraints. This is done based on the collision checking module of the TP, as well as some predefined vehicle dynamics constraints like maximum/minimum acceleration and maximal allowed speed. First, we compute all gaps in the surroundings of the agent. Then, we check which ones are safely reachable by at least one trajectory. In this manner, we define the set of reachable gaps, that are proposed to the RL-agent as actions (Figure 1). Once the agent selects a target gap, out of all trajectories that safely reach the gap, the TP outputs the best one. The best trajectory is selected based on a set of costs and weights taking into consideration the speed, the comfort, as well as how well the end points of the trajectory fit inside the gap. More precisely, we are minimizing the cumulative longitudinal and lateral jerk of the trajectory for comfort. In lateral direction, we penalize the lateral distance to the target lane center. In longitudinal direction, we predict constant velocity trajectories for all relevant[1] surrounding vehicles, so that we can maintain safe distance between them and the RL agent. This is done by calculating the time-to-collision and time-headway values for the planned trajectory, relative to the relevant surrounding vehicles. For a host vehicle v_h, a reference vehicle v_r, we define time-to-collision and time-headway as: $\text{ttc} = |v_{h,\text{pos}} - v_{r,\text{pos}}|/(v_{f,\text{speed}} - v_{l,\text{speed}})$ and $\text{thw} = |v_{h,\text{pos}} - v_{r,\text{pos}}|/v_{f,\text{speed}}$, where $v_{h,\text{pos}}$ and $v_{r,\text{pos}}$ are the longitudinal positions of the host and the reference vehicle and, $v_{\text{l, speed}}$ and $v_{\text{f, speed}}$ refer to whichever vehicle is leading/following between the host and the reference vehicle in the time of the calculation.

For more details on the trajectory generation, refer to [29].

[1]For e.g. if we are examining a keep-lane trajectory, only the vehicles on that lane are considered relevant.

C. Model-based Action Proposals Algorithm

We use DQN for the state-action value function approximation. For that purpose, we initialize a network $Q_{\mathcal{AQL}}$, with parameters θ and a target network $Q'_{\mathcal{AQL}}$, with parameters θ'. $Q_{\mathcal{AQL}}$ consists of the modules ϕ, ρ and Q, shown in Figure 3. The fully connected networks ϕ transform each of the dynamic input instances $\{s_{i,1}^{\text{dyn}}, s_{i,2}^{\text{dyn}}, ..., s_{i,j}^{\text{dyn}}\}$, describing the surrounding vehicles of the RL agent, into the representation $\phi(s_i^{\text{dyn}})$ as in [7]. Then a pooling operation (sum) is applied over the output, ultimately making the Q function permutation invariant wrt. its input. This is what enables handling dynamic number of input objects [32]. The output of the pooling operation is processed by the fully connected network ρ, providing the final reconstruction of the dynamic inputs. Besides the dynamic input instances, we also have a static part of the state s_i^{static} consisting of features describing the RL agent. In order to inform the RL agent about the available gaps, we include the features describing the reachable gaps. Finally we combine the final representation of the dynamic inputs $\rho(\sum_i \phi(s_i^{\text{dyn}}))$, with as many pairs $(s_i^{\text{static}}, s_i^{\text{gap}})$ as we have available gaps in time-step i. Then we propagate each triple $(\rho(\sum_i \phi(s_i^{\text{dyn}})), s_i^{\text{static}}, s_i^{\text{gap}})$ through the Q module separately, to finally get the state-action value estimation q for each proposed gap. In the end, we acquire the final policy by maximizing over the q-value estimations of all proposed gaps, and choosing the one with the maximal q-value. For details refer to Algorithm 1. Building upon [7], the parts marked with red in lines 11 and 12 in Algorithm 1, are the ones that we introduce for the description of the gap and for enabling variable number of actions in each time-step. Once we have a trained agent, we can apply it on a new driving scenario. The gap selection steps of a trained agent are summarized in Algorithm 2.

IV. TRAINING

We apply the AQL-MAP approach to teach an agent to drive on a simulated highway, in a smooth and safe way,

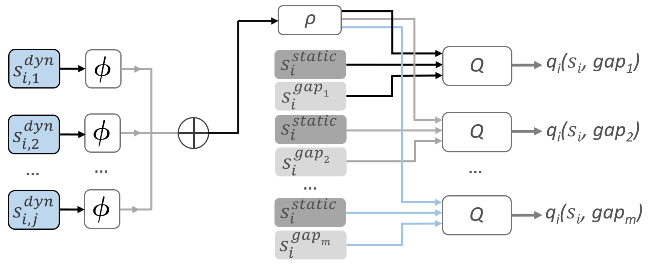

Fig. 3: Amortized Q-learning with Model-based Action Proposals network architecture

Algorithm 1: Fixed Batch AQL-MAP

```
1  initialize $Q_{\mathcal{AQL}} = (Q, \phi, \rho)$, $Q'_{\mathcal{AQL}} = (Q', \phi', \rho')$
2  set replay buffer $\mathcal{R}$
3  for training iteration $ti = 1, 2, \dots$ do
4      get mini-batch $M = (s_i, g_i, s_{i+1}, r_{i+1})$ from $\mathcal{R}$,
5      where $s_i = (S_i^{\text{dyn}}, s_i^{\text{static}})$, $s_{i+1} = (S_{i+1}^{\text{dyn}}, s_{i+1}^{\text{static}})$
6      foreach transition do
7          foreach object $s_{i+1}^j$ in $S_{i+1}^{dyn}$ do
8              $(\phi'_{i+1})^j = \phi'\left(s_{i+1}^j\right)$
9          end
10         $\rho'_{i+1} = \rho'\left(\sum_j (\phi'_{i+1})^j\right)$
11         $y_i = r_{i+1} + \gamma \max_a Q'(\rho'_{i+1}, (s_{i+1}^{\text{static}}, s_{i+1}^{\text{gap}}), g)$,
12         where: $g = s_{i+1}^{\text{gap}} \in \{s_{i+1}^{\text{gap}_1}, s_{i+1}^{\text{gap}_2}, \dots, s_{i+1}^{\text{gap}_m}\}$
13     end
14     perform a gradient step on loss:
           $$\frac{1}{b}\sum_i \left(Q_{\mathcal{AQL}}(s_i, a_i) - y_i\right)^2$$
15     update target network by:
           $$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$$
16 end
```

Algorithm 2: AQL-MAP Learned Agent Gap Selection

```
Input: RL agent's state $s_t^{rl} = \{\text{pos}_t, \text{speed}_t, \text{accel}_t\}$,
       RL state $s_t = (s_t^{\text{dyn}}, s_t^{\text{static}})$ in time-step $t = 0$
       Learned model $Q_{\mathcal{AQL}} = (Q, \phi, \rho)$
1  while episode not finished do
2      get the reachable gap proposals from the TP:
3      $s_t^{\text{gap}} = \{s_t^{\text{gap}_1}, s_t^{\text{gap}_2}, \dots, s_t^{\text{gap}_m}\}$
4      compute the dynamic state representation as in:
           $\text{state\_rep}_t^{\text{dyn}} = \rho\left(\sum(\phi_t(s_t^{\text{dyn}}))\right)$
5      $\text{Q\_values}_t = \{\}$
6      foreach gap $g_t^m = s_t^{gap_m}$ in $s_t^{gap}$ do
7          $\text{Q\_values}_t.\text{insert}(Q_{\mathcal{AQL}}(\text{state\_rep}_t^{\text{dyn}}, s_t^{\text{static}}, s_t^{gap}))$
8      end
9      choose gap $g_t = \text{argmax}(\text{Q\_values}_t)$
10 end
```

while fulfilling the desired speed requirement. In this section, we discuss the training pipeline and its components.

A. Options Framework

At a time-step t the RL agent, chooses a gap g (an option) from the set of reachable gaps. Unlike the classical RL framework, where the RL agent chooses a new action in fixed time-intervals, here, the duration of the intervals is arbitrary. Every second, if the gap is not reached and it is still available, the RL agent is not required to choose again. If the initially chosen option is not available anymore, we can interrupt it and choose a new, reachable one.

B. State space

The RL state consists of features describing the RL agent, its surrounding vehicles and gaps. For the RL agent we use:

- absolute velocity, $v_{\text{RL}} \in \mathbb{R}_{\geq 0}$,
- left lane valid flag, indicating whether there is a lane left of the RL agent, $\text{ll}_{\text{valid}} \in \{0, 1\}$,
- analogously for the right lane: $\text{rl}_{\text{valid}} \in \{0, 1\}$.

For describing the vehicles j surrounding the RL agent, we consider all vehicles in sensor range sr of $80\,\text{m}$ ahead and behind. We describe them with the following features:

- relative distance $d_{\text{rel},j}$, defined as $(\text{pl}_j - \text{pl}_{\text{RL}})/\text{sr}$, where pl_{RL} and pl_j are the longitudinal positions of the RL agent and the considered vehicle j respectively,
- relative velocity $v_{\text{rel},j}$, defined as $(v_j - v_{\text{RL}})/v_{\text{des,RL}}$, where v_j is the absolute velocity of the vehicle j, and $v_{\text{des, RL}}$ is the user-defined desired velocity the RL agent is aiming to achieve,
- relative lane $\text{lane}_{\text{rel},j}$, defined as $\text{lane}_{\text{ind},j} - \text{lane}_{\text{ind},RL}$, where ind $\in \{0, 1, 2\}$, represents the right, middle and left lane indices.

Additionally, we include the features for every reachable gap in the surroundings, described below.

C. Action space

The action set consists of all reachable gaps RG, described in Section III-A. For description of the gaps we use:

- relative distance $d_{\text{rel},g}$, relative velocity $v_{\text{rel},g}$ and relative lane $\text{lane}_{\text{rel},g}$, defined the same way as in Section IV-B for the surrounding vehicles, here in terms of gaps,
- gap length, len_g, defined as $\text{pl}_{\text{lv}} - \text{pl}_{\text{fv}}$, where pl_{lv} and pl_{fv} are the longitudinal positions of the leading and the following vehicles forming the gap g,
- gap association flag, $\text{af}_g \in \{0, 1\}$, which is 0 if the currently chosen gap is the same as the one chosen at the previous time-step, and 1 otherwise.

D. Reward function

For a desired velocity $v_{\text{des, RL}}$, given $\delta_{\text{vel}} = |v_s^{\text{RL}} - v_{\text{des, RL}}|$, the reward function $r : S \times A \to \mathbb{R}$, is defined as:

$$r(s,a) = \begin{cases} 1 - \delta_{\text{vel}} + p_{\text{ac}}, & \text{if } v_s^{\text{RL}} < v_{\text{des, RL}} \\ 1 + p_{\text{ac}}, & \text{otherwise,} \end{cases} \quad (1)$$

where p_{ac} is a slight penalty for choosing an action that differs from the one chosen in the previous time-step, which encourages smoother driving. We use $p_{\text{ac}} = -0.01$, chosen empirically by conducting a wide range of experiments.

E. Training data details

We collected two training data sets, one for the high-level agent, where the RL agent chooses an action every second, and one for the options-selecting agent, which chooses a new option in intervals with varying size (depending on when a selected option terminates). However, apart from the action/option choosing frequency, we made sure that they are collected in the same way, so everything we describe below holds for both of them. In the SUMO simulation environment, we collected around $5 * 10^5$ transition samples (s, a, s', r), that correspond to approximately 150 hours of driving. The data is collected on a 3-lane straight highway. Besides the RL agent, there are between 0 and 70 surrounding vehicles, positioned randomly on the highway, with varying desired speeds and driver behaviors. All surrounding vehicles are allowed to change lanes and accelerate/decelerate as their underlying controller suggests, we have control only over the RL agent. For the options-selecting agent, we used a random data collection policy. For the high-level agent, we collected data in a pseudo-random way, where sometimes the agent would keep the previously chosen action with higher probability. This is in order to make sure that we incorporate samples in the data that describe completely executed lane-changes.

V. EXPERIMENTS

A. *Benchmark agents*

As already mentioned, we compared the performance of the option-selecting RL agent to the one of 4 other benchmark agents: random agent, greedy agent, high-level RL agent and IDM-based SUMO agent. The random agent is the simplest baseline providing us with a benchmark performance of a non-intelligent agent. The greedy agent, out of all reachable gaps, selects the gap with the fastest trajectory leading towards it. The IDM-based agent is a SUMO-controlled agent. We set its parameters as relaxed as possible in order to make sure that no internal constraints prevent it from achieving the desired speed (such as no overtaking from the right, being too cooperative etc.). Finally, the high-level RL agent chooses from discrete actions: keep lane, left lane-change and right lane-change. Since for this agent there is no need to feed the actions into the network as a part of the RL state, (because they are discrete and fixed,) we used the classic DeepSet DQN architecture. However, we made sure that everything else is comparable to the way the option-selecting agent was trained. We used the same reward function and the same RL state (excluding the options-related features). During evaluation, all agents were rewarded based on the same criteria. The underlying trajectory planning and selecting mechanism is identical for all agents. They only differ in the action/option selection strategy.

B. *Evaluation setup*

We evaluated all agents on the same set of realistic scenarios, generated randomly, as described in IV-E. In order to objectively assess the driving of the agents, we generated scenarios with varying number of surrounding vehicles $n = \{10, 20, ..., 80\}$, simulating fluctuating traffic density, from light to heavy. For each number of surrounding vehicles we have 10 different evaluation scenarios (80 in total). Additionally, we evaluate the performance of the agents on critical, more challenging highway scenarios, shown in

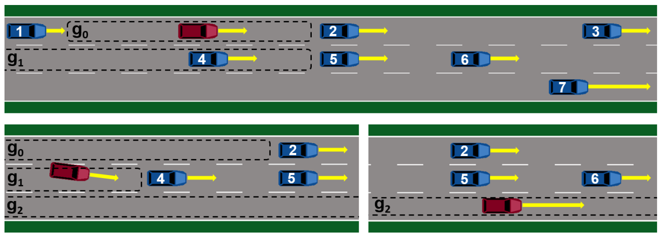

Fig. 4: The RL agent (red), surrounded by slow vehicles learned that it is beneficial to move over to the middle lane in order to reach the right-most lane where it will make progress faster.

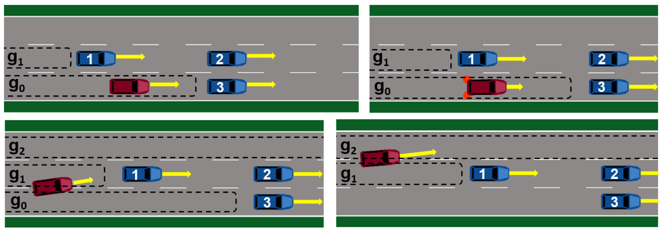

Fig. 5: More challenging dense traffic highway situation that can occur for e.g. by entering a highway. Only the options-agent has learned that even though when choosing option g_1, prolonged braking maneuver must be applied, it is still the best long-term strategic choice.

Figure 4 and Figure 5, where planning ahead is crucial. We compared the average achieved speeds of all agents on the 80 scenarios. The average is calculated over the 10 scenarios with same number of cars in it. In order to acquire a better overview of the performance, for the random agent and the RL agents, we calculated an average over 10 different runs. For the RL agents, we calculated the average over 10 models trained with the same randomly initialized hyper-parameters (Table I). This allows for a better inspection of the robustness of the approach. We performed over 500 random search runs (configuration space shown in Table I) using the Hyperband framework for distributed hyper-parameters optimization [12].

TABLE I: Training hyper-parameters and configuration spaces

Parameter	Final value	Search configuration space
training steps	$75K$	
batch size	64	
discount factor	0.99	
update type	soft	
learning rate	10^{-4}	$[10^{-1}, 10^{-2}, ..., 10^{-6}]$
τ	10^{-4}	$[10^{-1}, 10^{-2}, ..., 10^{-6}]$
ϕ hidden dim.	20	$[10, 20, ..., 100]$
ϕ output dim.	80	$[50, 60, ..., 130]$
ρ hidden dim.	80	$[50, 60, ..., 130]$
ρ output dim.	20	$[10, 20, ..., 100]$
FC$_1$/FC$_2$ dim.	100	$[50, 80, 100, 200, 300]$
pooling fn.	$\sum$	

VI. RESULTS

Figure 6 shows the final average speed achieved by each agent in the randomly generated scenarios. As expected the

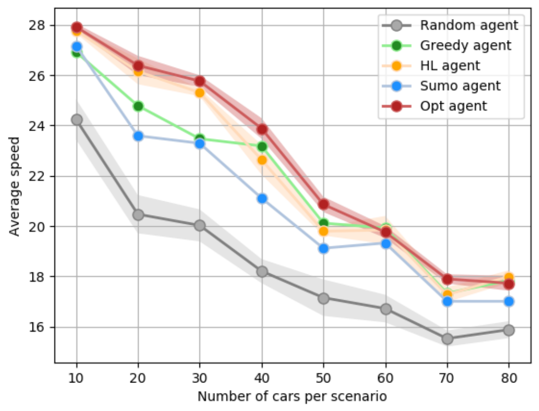

Fig. 6: Performance of all agents in the randomly generated scenarios. The shaded areas represent the standard deviation over the 10 runs for the random, the high-level and the options agents.

random agent (grey) shows the worse performance. The greedy agent performs better than the random but worse than the high-level agent and the options-agent. The SUMO-agent, due to its simplistic strategy falls slightly behind the greedy agent. The high-level agent and the options-agent have similar performance in the generic scenarios. This is because in simple, usual scenarios where the others behave predictably, the benefits of long-term planning are less visible. Additionally, the high-level agent has learned to exploit the longitudinal trajectories of the trajectory planner which is sufficient for navigating the common, randomly generated scenarios. However, in the section below we show that the options agent is superior as a whole, since when it comes to delicate situations, it has learned to incorporate long-term planning, unlike any of the comparison agents.

A. Performance in the critical scenarios

For further investigation of the performance, we looked into specific, more challenging scenarios, where planning ahead is crucial. In the scenario in Figure 4 (from left to right), the RL agent (red), is in gap g_0, surrounded by slow vehicles. In the starting position, there are 2 available gaps: stay in g_0 or start reaching ${g_1}$[2]. The RL agent and the leading vehicles of the reachable gaps are driving with the same speed, hence, there is no obvious short-term incentive to change a lane. However, the options-selecting agent has learned that it is long-term beneficial to move over to the middle lane. Since once it is there, the gap g_2 to the right opens up, where it will be able to speed up and overtake the slow vehicles. For the high-level agent, since it is limited to $3\,\mathrm{s}$ lane-changes, the only available gap is the starting one. This is because in the starting position its speed is the same as the speed of the leading vehicles and a $3\,\mathrm{s}$ lane change is not feasible. In any given situation when for example the surrounding vehicles are driving with similar speed as the RL agent, and the RL agent is limited to discrete actions, a specific maneuver that may seems simple and logical, may be infeasible. The IDM-based SUMO agent even though completely informed about the surroundings, without a mechanism for long-term considerations, failed to react appropriately. Since both reachable gaps g_0 and g_1 have the same speed, and moving to the next lane would result in an additional penalty, the greedy agent also fails to exit the initial gap. In Figure 5), the agent and the 3 surrounding vehicles drive with the same speed, lower than the RL agent's desired speed ($30\,\mathrm{m/s}$). However, the surrounding vehicles have achieved their desired velocities and have no intention to accelerate. From that situation the agent has to break in order to reach the middle lane gap g_1. Even though braking i.e. selecting a slow gap behind, translates directly into lower immediate reward, the options agent learned that it is essential in the long run, since staying in the initial gap is only locally optimal. Figure 7 summarizes the performance of all agents averaged over the multiple runs, in terms of average speed and duration on the scenarios from Figure 4 (blue) and Figure 5 (green) respectively.

[2]We do not consider double lane-changes directly, since there are more surrounding vehicles to be considered and it is harder to guarantee safety.

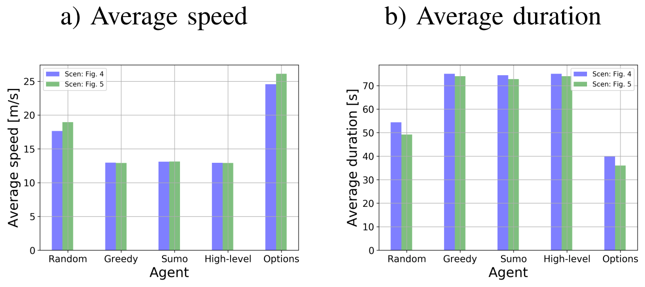

Fig. 7: Performance of all agents in the challenging scenarios from Figure 4 and Figure 5 in terms of speed and duration.

VII. CONCLUSION

In this work, we introduced the Amortized Q-learning with Model-based Action Proposals approach for decision-making in common and critical highway situations. It is a Reinforcement Learning-based method, with an embedded optimal control trajectory planner, that is not limited to fixed action sets and supports variable number of inputs. This is especially beneficial in confined space dynamic environments, where precise problem definition is crucial. Moreover, it brings the best of both worlds together: a safe optimization-based trajectory planning and execution and long-term optimal decision-making. We evaluated our options-selecting agent against 4 benchmark agents: a random agent, a greedy agent, IDM-based SUMO agent and a high-level discrete actions agent. We were able to show that the agents trained with our approach, not only achieved better overall performance but also were the only ones that managed to successfully navigate their way in the challenging scenarios. In the future, it would be interesting to incorporate an additional interaction term in the reward signal such as courtesy. This would make sure that the agent is cooperative and takes into account how its driving affects others.

References

[1] Ali Alizadeh et al. *Automated Lane Change Decision Making using Deep Reinforcement Learning in Dynamic and Uncertain Highway Environment*. 2019. arXiv: `1909.11538 [cs.RO]`.

[2] A. Bacha et al. "Odin: Team VictorTango's entry in the DARPA Urban Challenge". In: *J. Field Robotics* 25 (2008), pp. 467–492.

[3] F. Borrelli et al. "MPC-Based Approach to Active Steering for Autonomous Vehicle Systems". In: *International Journal of Vehicle Autonomous Systems* 3 (2005), pp. 265–291.

[4] P. Falcone et al. "Predictive Active Steering Control for Autonomous Vehicle Systems". In: *IEEE Transactions on Control Systems Technology* 15 (2007), pp. 566–580.

[5] Hado van Hasselt, Arthur Guez, and David Silver. *Deep Reinforcement Learning with Double Q-learning*. 2015. arXiv: `1509.06461 [cs.LG]`.

[6] C. Hoel, K. Wolff, and L. Laine. "Automated Speed and Lane Change Decision Making using Deep Reinforcement Learning". In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018, pp. 2148–2155. DOI: `10.1109/ITSC.2018.8569568`.

[7] Maria Hügle et al. "Dynamic Input for Deep Reinforcement Learning in Autonomous Driving". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2019), pp. 7566–7573.

[8] Maria Hügle et al. "Dynamic Interaction-Aware Scene Understanding for Reinforcement Learning in Autonomous Driving". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)* (2020), pp. 4329–4335.

[9] G. Kalweit et al. "Interpretable Multi Time-scale Constraints in Model-free Deep Reinforcement Learning for Autonomous Driving". In: *ArXiv* abs/2003.09398 (2020).

[10] Gabriel Kalweit et al. *Deep Inverse Q-learning with Constraints*. 2020. arXiv: `2008.01712 [cs.LG]`.

[11] Daniel Krajzewicz et al. "Recent Development and Applications of SUMO - Simulation of Urban MObility". In: *International Journal On Advances in Systems and Measurements* 3&4 (Dec. 2012).

[12] Lisha Li et al. "Efficient Hyperparameter Optimization and Infinitely Many Armed Bandits". In: *CoRR* abs/1603.06560 (2016). arXiv: `1603.06560`. URL: `http://arxiv.org/abs/1603.06560`.

[13] Branka Mirchevska et al. "High-level Decision Making for Safe and Reasonable Autonomous Lane Changing using Reinforcement Learning". In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)* (2018), pp. 2156–2162.

[14] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: `1312.5602 [cs.LG]`.

[15] M. Montemerlo et al. "Junior: The Stanford Entry in the Urban Challenge". In: *The DARPA Urban Challenge*. 2009.

[16] W. Schwarting, Javier Alonso-Mora, and D. Rus. "Planning and Decision-Making for Autonomous Vehicles". In: 2018.

[17] W. Schwarting et al. "Parallel autonomy in automated vehicles: Safe motion generation with minimal intervention". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)* (2017), pp. 1928–1935.

[18] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. *Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving*. 2016. arXiv: `1610.03295 [cs.AI]`.

[19] D. Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529 (2016), pp. 484–489.

[20] D. Silver et al. "Mastering the game of Go without human knowledge". In: *Nature* 550 (2017), pp. 354–359.

[21] R. Sutton, D. Precup, and Satinder Singh. "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning". In: *Artif. Intell.* 112 (1999), pp. 181–211.

[22] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: `http://incompleteideas.net/book/the-book-2nd.html`.

[23] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. "Congested traffic states in empirical observations and microscopic simulations". In: *Physical Review E* 62.2 (Aug. 2000), pp. 1805–1824. ISSN: 1095-3787. DOI: `10.1103/physreve.62.1805`. URL: `http://dx.doi.org/10.1103/PhysRevE.62.1805`.

[24] C. Urmson et al. "Autonomous driving in urban environments: Boss and the Urban Challenge". In: *J. Field Robotics* 25 (2008), pp. 425–466.

[25] Pin Wang, Ching-Yao Chan, and Arnaud de La Fortelle. *A Reinforcement Learning Based Approach for Automated Lane Change Maneuvers*. 2018. arXiv: `1804.07871 [cs.RO]`.

[26] Pin Wang, Hanhan Li, and Ching-Yao Chan. *Quadratic Q-network for Learning Continuous Control for Autonomous Vehicles*. 2019. arXiv: `1912.00074 [cs.LG]`.

[27] Sen Wang, Daoyuan Jia, and Xinshuo Weng. *Deep Reinforcement Learning for Autonomous Driving*. 2019. arXiv: `1811.11329 [cs.CV]`.

[28] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: *Machine Learning*. 1992, pp. 279–292.

[29] M. Werling et al. "Optimal trajectory generation for dynamic street scenarios in a Frenét Frame". In:

2010 IEEE International Conference on Robotics and Automation (2010), pp. 987–993.

[30] Tom Van de Wiele et al. "Q-Learning in enormous action spaces via amortized approximate maximization". In: *CoRR* abs/2001.08116 (2020). arXiv: `2001.08116`. URL: `https://arxiv.org/abs/2001.08116`.

[31] Changxi You et al. "Advanced Planning for Autonomous Vehicles Using Reinforcement Learning and Deep Inverse Reinforcement Learning". In: *Robotics and Autonomous Systems* 114 (Jan. 2019). DOI: `10.1016/j.robot.2019.01.003`.

[32] Manzil Zaheer et al. "Deep Sets". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 3391–3401. URL: `http://papers.nips.cc/paper/6931-deep-sets.pdf`.