Guiding Belief Space Planning with Learned Models for Interactive Merging

Johannes Fischer[1], Etienne Bührle[1], Danial Kamran[1] and Christoph Stiller[1]

***Abstract*—Safely navigating complex, interactive situations is one of the major challenges in planning for automated vehicles. The key difficulty is that drivers have unobservable properties determining their driving style. To make optimal decisions, the belief over latent driver states has to be considered, which is computationally challenging. For this reason, online planning algorithms suffer from the curse of dimensionality in highly interactive scenarios. On the other hand, learned policies often have difficulties generalizing to new environments with different driving styles. We propose to use policies trained in belief space as heuristics to guide online belief space planning algorithms, thereby alleviating the curse of dimensionality. We evaluate the proposed approach in a cooperative merging scenario.**

***Index Terms*— MCTS, DQN, reinforcement learning, belief state planning, interactive, cooperative, merging.**

I. Introduction

One of the greatest challenges in autonomous driving is how to solve highly interactive situations that require drivers to negotiate with each other. While humans intuitively rely on social cues in such situations, it is not easy for an automated system to safely navigate such encounters without being overly conservative.

One such challenging scenario is merging into an urban traffic flow, as depicted in Fig. 1. In dense traffic situations, other vehicles might be courteous and allow the merge. Hence, the merging vehicle has to clearly communicate its desire to merge while at the same time reasoning about the other vehicles' cooperativeness to estimate if a merge is possible.

Previous work has tackled the interactive nature of such situations by probabilistically estimating the latent parameters of other vehicles and then using online planning algorithms to act optimally with respect to the estimated parameters [1]–[3]. These approaches include sampling-based motion-planning [1], [2], as well as belief space planning with tree search methods [3]. The value of reasoning about other vehicles in belief space has been proven by Sunberg *et al.* [4]. While the probabilistic nature of those approaches allows dealing with latent parameters, they suffer from the curse of dimensionality and can not easily scale to complex interactive scenarios.

Others approach interactivity by game-theoretic planning, reasoning not only for the ego vehicle but also for the other vehicles [5], [6]. These methods can be combined with an online inference of parameters or objectives of other vehicles [7], [8]. However, game-theoretic planning is computationally highly demanding, which makes it difficult to apply in real-time systems with multiple interacting vehicles.

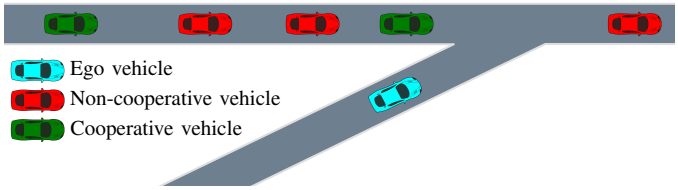

Figure 1: The ego vehicle (cyan) has to merge onto the main road where cooperative vehicles (green) might yield while non-cooperative vehicles (red) ignore it.

In contrast to those online planning algorithms, there is also a rich body of work that learns policies for interactive scenarios using reinforcement learning (RL). Continuous control policies for merging into dense traffic have been learned, where the other vehicles stochastically yield based on a cooperation parameter [9]. Leurent *et al.* use an attention mechanism to capture the interactivity between vehicles [10]. Bouton *et al.* demonstrate how reinforcement learning can be used directly in belief space to fully leverage information on the latent states [11]. Game-theoretic ideas have also been employed in iterative reinforcement learning schemes to approximate the strategic reasoning process [12].

Reinforcement learning approaches can scale up more easily to complex scenes with multiple interacting vehicles since no online planning is involved. However, additional measures have to be taken to ensure the safety of the learned policy, such as allowing only safe actions [13]–[15]. In contrast, planning algorithms have the advantage of actively reasoning about the situation encountered at deploy time, which helps to achieve safe driving behavior.

Silver *et al.* have combined tree search-based planning algorithms with learned heuristics to solve board games [16], [17]. This idea has been explored in the domain of automated driving, where a particle filter is used to estimate the most likely state, which is then used for planning [18].

Our work falls into the last category of methods. We use learned models to guide the online planning algorithm to the most promising areas of the search tree to overcome the curse of dimensionality. In contrast to Hoel *et al.* (2020), we propose to directly plan in belief space instead of only for the most likely state to fully consider the current uncertainty and

[1] Johannes Fischer, Etienne Bührle, Danial Kamran and Christoph Stiller are with the Institute of Measurement and Control Systems, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany. Corresponding author: johannes.fischer@kit.edu

leverage the effect of ego actions on future information. To this end, we use belief space reinforcement learning to train a policy and value function over the belief space. They are used during belief space tree search to estimate the value of new tree nodes and as a prior on the tree policy, respectively. Our method enables tree search methods to scale up and at the same time makes use of powerful learned policies while still planning online.

The contribution of our work is threefold:

- We propose a principled way of combining learning and planning in the belief space that can make use of different RL algorithms.
- We evaluate our method on a cooperative driving scenario and compare its performance with state-of-the-art baselines.
- We show that our algorithm is robust to transfer to new environments by evaluating it on two scenarios of increased difficulty that were not seen during training.

II. Background

We begin our analysis by presenting the theoretical frameworks employed to model the decision-making problem and giving an overview of the algorithms used in this work.

A. Partially Observable Markov Decision Processes

High-level decision-making problems can often be modelled as Partially Observable Markov Decision Processes (POMDPs). Such processes are described by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{Z}, \mathcal{R}, \gamma)$ where $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{O}$ are the state, action, and observation spaces, respectively, $\mathcal{T}$ is the transition model, $\mathcal{Z}$ the observation model, $\mathcal{R}$ the reward function and $\gamma \leq 1$ a discount factor [19]. In this discrete-time model, the environment (which includes the agent) is in a state $s \in \mathcal{S}$ at each point in time. After selecting an action $a \in \mathcal{A}$ in state s, the environment stochastically transitions to a successor state s' with probability $\mathcal{T}(s'|s,a)$ given by the transition model and the agent receives a reward $r = \mathcal{R}(s,a)$. The agent's goal is to choose actions such that the cumulative discounted future reward $\sum_{t=0}^{\infty} \gamma^t r_t$ (the so-called return) is maximized.

The main difficulty is that the agent is not able to observe the current state. Instead, it receives a noisy observation $o \sim \mathcal{Z}(\cdot|s,a,s')$ after transitioning from state s to s' with action a. With knowledge of the transition and observation models, the agent can keep track of the current state distribution, known as the belief b, using Bayes' theorem.

B. Belief MDP

If the state is fully observable, the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ forms a Markov Decision Process (MDP). Any POMDP can be equivalently reformulated as an MDP over the belief space $\mathcal{B}$ [19]. The belief state of this *belief MDP* transitions to successor beliefs with probabilities according to the observation model.

C. Monte Carlo Tree Search

A standard method for solving POMDPs is Monte Carlo Tree Search (MCTS) [20]. MCTS selects actions by constructing a search tree from many simulations of the current state for different action sequences where each simulation adds a new node to the tree. The layers of the tree alternate between *action nodes*, representing different action choices, and *state nodes*, representing stochastic state transitions. In each action node, the action value function $Q(s,a)$, denoting the expected return when starting in state s with action a, is estimated by averaging the return values of all simulations passing through that node with action a. The tree policy selects between action nodes within the tree by maximizing the upper confidence tree (UCT) objective

$$U(s,a) = Q(s,a) + c_{\text{uct}} \sqrt{\frac{\ln \sum_{a'} N(s,a')}{N(s,a)}}, \qquad (1)$$

where the parameter c_{uct} can be used to balance between exploitation of high-return areas and the exploration of actions with a low number of samples $N(s,a)$ [21]. When reaching a leaf node, the value of the leaf node is estimated by the return of a random rollout from this node. Then the leaf node is expanded and all visited nodes are updated with their return values.

The tree search ends after a predefined number of simulations or after exhausting a computational budget. Then the action with the highest Q-value estimate is selected.

In continuous environments with stochastic transitions, a new state would be sampled in every simulation. Hence, the tree would remain shallow. To remedy this, progressive widening can be used to artificially limit the number of child nodes to $kN(s,a)^{\alpha}$, with hyperparameters k and α [22]. If the maximal number of child nodes is already reached, one of the existing nodes is sampled at random.

D. Deep Reinforcement Learning

Another family of widely used methods for decision-making problems is reinforcement learning. Such methods do not require knowledge of the transition or observation models. But instead, they learn an optimal policy π from many interactions with the environment [23]. In continuous environments, neural networks are frequently used to approximate the value function, e.g. in the Deep Q-network (DQN) algorithm [24].

III. Approach

Combining planning and learning has been shown to be very effective in the game of Go and other board games [16], [17], but also in the domain of automated driving [18]. However, these works either solve fully observable environments or plan only for the most likely state.

Just acting based on the last observation or the most likely state can be suboptimal because it does not take the information of the full belief into account. To optimally solve a POMDP, belief space planning is necessary. Therefore, we propose to combine MCTS and reinforcement learning to solve

partially observable problems directly in the belief space. One difficulty this imposes is that the belief space is of much higher dimensionality than the state space: A binary state variable alone introduces a continuous dimension in the corresponding belief space.

A. Algorithm Overview

Our algorithm follows the general MCTS procedure described in Section II-C, but solves the belief MDP. This requires a belief updater $\mathcal{U}$ that produces the posterior belief b' given a prior belief, the chosen action and the observation. Whenever a new simulation starts in the root node, a state is sampled from the root belief. Throughout this simulation, the sampled state is simulated forward with the chosen actions and is used to generate observations for updating the belief with the belief updater $\mathcal{U}$. Hence, the simulations trace trajectories in the belief space.

B. Belief Space Reinforcement Learning

The belief represents the agent's knowledge of the state. For optimal decision-making, a reinforcement learning agent has to make its decisions based on the belief. Often this is approximated by providing a history of observations to the agent [15], or the belief is tracked implicitly within a recurrent neural network architecture [25]. Since we propose to use MCTS planning in belief space, we must also train the learned heuristics with belief space reinforcement learning. In our work, we follow Bouton *et al.*, who learned a policy directly in belief space by extending reinforcement learning with a belief updater [11]. The input to the neural networks is a vectorized representation of the belief.

C. Guiding the Tree Policy with Learned Models

In the following, we present different methods to employ a model trained with reinforcement learning as guidance for the MCTS tree policy, similar to the methods presented for the AlphaZero algorithm [16]. However, we adapt these methods to the belief space formulation of a POMDP. Depending on whether the reinforcement learning agent learned a policy, or a value or Q-value function, or both, only some of the strategies are applicable.

The MCTS tree policy selects the action that maximizes the upper confidence tree formula in Eq. (1). This formula consists of an exploitation term preferring high estimated Q values and an exploration term that considers how often each action has been tried. We consider both terms for using a learned model as additional guidance.

1) Using a learned policy for rollouts: A learned policy can be used to estimate the value in leaf nodes. Instead of performing a random rollout to estimate the value of leaf nodes, the rollout can be performed following the optimal policy. Since random rollouts are highly unlikely to produce good value estimates, this greatly improves the value estimate in the leaf node, which is used to update all nodes in this branch of the tree. Hence, this makes the exploitation term of UCT much more accurate.

2) Using Q-value estimates instead of rollouts: Similarly, a learned value function can be used to estimate the value at leaf nodes. Whenever a leaf node is reached, the trained value network is queried for the simulated belief instead of estimating its value with a random rollout. Since the learned value function subsumes different stochastic evolutions of the environment, this results in a lower variance estimate than using a rollout. At the same time, this approach saves the computational cost of performing the rollout, allowing a larger number of simulations to be executed from the root node within a given computational budget. Note that if the learned model provides only Q-values, the value of a belief b is simply given by $V(b) = \max_a Q(b, a)$.

3) Using Q-value estimates to initialize action node values: Each action node maintains a running estimate of the Q value corresponding to this state-action pair. This estimate is continually updated with each node visit and is used in the UCT formula. When a new node is added to the tree, by default the Q values are initialized with zero. Instead, a learned Q-value function can be used to initialize the action node's value.

4) Using a RL policy as exploration prior: The AlphaZero algorithm proposes to use a trained policy π_θ with weights θ as an action prior in a variant of UCT. Formulated for belief space planning, the formula is given by

$$U_\theta(b, a) = Q(b, a) + c_{\text{uct}} \pi_\theta(b, a) \frac{\sqrt{\sum_{a'} N(b, a')}}{1 + N(b, a)}. \quad (2)$$

This strategy leads to more exploration in the tree areas that are believed to be highly rewarding by the policy.

This strategy can be used even if the agent did not explicitly learn a policy π_θ (e.g. a DQN agent). In this case, we propose to use a Boltzmann policy

$$\pi_\theta(b, a) = \frac{\exp(Q_\theta(b, a))}{\sum_{a'} \exp(Q_\theta(b, a'))} \quad (3)$$

in Eq. (2).

IV. Implementation

In the following, we describe how the merging scenario and the interactive behavior of the other vehicles is modelled as a Partially Observable Markov Decision Process (POMDP) closely following the work of Bouton *et al.* [11].

A. POMDP Formulation

In the considered scenario, the ego vehicle starts on a merging road and has to merge onto a main road with varying traffic density.

1) States: We model the merging scenario in a continuous state space. Since we are primarily interested in the interactions of the merging behavior and each vehicle follows a defined path, only the longitudinal motion of the vehicles is considered. Hence, the physical state of each vehicle consists of its longitudinal position x, measured as the distance to the merging point, velocity v, and acceleration a. Additionally, each vehicle's state contains an unobserved parameter determining its interactive driving behavior, namely its cooperation

level c, which is described in detail in Section IV-B. Hence, the full state $s = (s^{\mathrm{e}}, s^1, \ldots, s^N)$ consists of the ego vehicle state $s^{\mathrm{e}} = (x^{\mathrm{e}}, v^{\mathrm{e}}, a^{\mathrm{e}})$ and the physical and latent states of a variable number of N other vehicles $s^i = (x^i, v^i, a^i, c^i)$.

2) Actions: The scenario evolution is discretized into time steps of length Δt. To achieve smooth driving, we model the action space using jerk levels. Specifically, the agent can choose between the jerk levels $j_t \in \{-1\,\frac{\mathrm{m}}{\mathrm{s}^3}, 0\,\frac{\mathrm{m}}{\mathrm{s}^3}, 1\,\frac{\mathrm{m}}{\mathrm{s}^3}\}$ which update the acceleration according to $a_t^{\mathrm{e}} = a_{t-1}^{\mathrm{e}} + j_t \Delta t$. Together with an emergency brake action that immediately sets the acceleration to $a_t^{\mathrm{e}} = -4\,\frac{\mathrm{m}}{\mathrm{s}^2}$, the agent has the choice between four actions in each step.

3) Transition Model: All vehicles follow a simplified dynamics model according to

$$x_{t+1} = x_t + v_t \Delta t + \frac{1}{2} a_t \Delta t^2, \tag{4}$$

$$v_{t+1} = v_t + a_t \Delta t. \tag{5}$$

The ego vehicle acceleration is determined as described in the previous section whereas the other vehicles choose their acceleration according to the driver model described in Section IV-B. When vehicles reach the end of the road, they are spawned at the beginning of the main road again with a probability p_{spawn}.

4) Reward Model: The reward function encodes the desired objectives of safety, efficiency, and comfort. At each time step t, the agent receives a penalty of $-0.1a_t^2 - 0.1j_t^2$ to prevent excessive acceleration and jerk. Once the agent reaches the goal region $50\,\mathrm{m}$ behind the merge point, it receives a reward of $+100$. In case of a collision with another vehicle, it receives a penalty of -100. The agent is encouraged to reach the goal quickly since future rewards are discounted with a discount factor of $\gamma = 0.99$. An episode terminates if the agent reaches the goal, collides with another vehicle, or after a fixed time of $100\,\mathrm{s}$ has elapsed.

5) Observation Model: The agent is able to observe the positions x^i and velocities v^i of other vehicles, as well as its own physical state s^{e}. The acceleration and cooperation level of other vehicles is not observable. Since this work is focused on the interaction between vehicles and not on sensor uncertainties, there is no noise in the observation. To effectively deal with a varying number of vehicles in the scenario, we consider only the four most important vehicles for merging in the observation. These are the vehicles right before and after the merge point, and the front and rear vehicle of the projection of the ego car onto the main road, as illustrated in Fig. 2. The projection is defined by the longitudinal distance to the merge point.

B. Model for Cooperative Driving

The behavior of the vehicles on the main road is modelled by the Cooperative Intelligent Driver Model (C-IDM) [11], an extension of the Intelligent Driver Model (IDM) [26]. As with standard IDM, the model computes a longitudinal acceleration based on the front vehicle. Additionally, C-IDM also takes merging vehicles into account by introducing a cooperation level c in the interval $[0, 1]$.

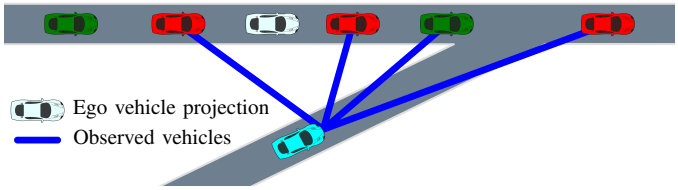

Figure 2: The ego vehicle (cyan) can observe the vehicles before and behind the merge point as well as the vehicles before and behind the ego projection (white) onto the main road.

C-IDM estimates the time-to-merge (TTM) for the merging vehicle ($\mathrm{TTM}_{\mathrm{merge}}$) as well as for the vehicle on the main road ($\mathrm{TTM}_{\mathrm{main}}$) using a constant velocity prediction. If $\mathrm{TTM}_{\mathrm{merge}} < c \cdot \mathrm{TTM}_{\mathrm{main}}$, the projection of the merging vehicle onto the main road is used as the IDM target for the main road vehicle. If $\mathrm{TTM}_{\mathrm{merge}} \geq c \cdot \mathrm{TTM}_{\mathrm{main}}$, or if no merging vehicle is within the field of view of $30\,\mathrm{m}$, the main road vehicle follows standard IDM behavior. Thus, cooperative cars ($c = 1$) will slow down for the merging vehicle if it reaches the merging point earlier. Non-cooperative vehicles ($c = 0$) will not slow down in any case.

Hence, given the cooperation level c, the C-IDM behavior is deterministic. While the model might not accurately represent human-like driving, it certainly exhibits behavior that happens in real traffic merging situations.

C. Initial State Distribution

When initializing the scenario, the ego vehicle is placed at a distance of $50\,\mathrm{m}$ from the merge point on the merging road with an initial velocity of $10\,\frac{\mathrm{m}}{\mathrm{s}}$. A random number of $N \in \{N_{\min}, \ldots, N_{\max}\}$ vehicles is spawned in random positions on the main road with normally distributed initial velocities with a mean of $5\,\frac{\mathrm{m}}{\mathrm{s}}$ and a standard deviation of $1\,\frac{\mathrm{m}}{\mathrm{s}}$. The desired velocity v_{des} for each vehicle is also randomly sampled depending on the desired traffic conditions with the parameters given in Table I. Each vehicle has a random cooperation level $c \in [0, 1]$. The remaining IDM parameters have constant values of desired time headway $T = 1.5\,\mathrm{s}$, minimum spacing $d = 2\,\mathrm{m}$, and maximum acceleration $|a|_{\max} = 2\,\frac{\mathrm{m}}{\mathrm{s}^2}$. To generate a more natural initial traffic scene, the vehicles on the main road are simulated forward for a random time interval between $5\,\mathrm{s}$ and $10\,\mathrm{s}$ using their respective driver model, while the ego vehicle state remains unchanged.

D. Tracking the Cooperation Belief

As described in Section III-B, the reinforcement learning agent is trained in the belief space. In our scenario, there is one latent state per vehicle – the cooperation level. The physical states of the vehicles are fully observable, while the desired velocity parameter of the IDM model is not observable. However, it is not estimated, since the cooperation level has a much higher impact on the resulting behavior for the ego

vehicle. The other IDM parameters are assumed to be known. Hence, the belief is only estimated over cooperation levels, which decreases the required computational effort for belief updates.

Following prior work, the cooperation level is assumed to be binary for filtering to make the belief updates more efficient [11]. Let $\hat{\theta}_t^i$ denote the estimated probability for $c^i = 1$ at time t and $\hat{s}_t^{\mathrm{ML}}$ the most likely state. In the update of vehicle i's cooperation $\hat{\theta}_t^i$, the most likely state $\hat{s}_t^{\mathrm{ML}}$ is constructed from the known physical states, randomly sampled desired velocities, and assuming $c^j = \hat{\theta}_t^j$ for all other vehicles $j \neq i$.

After receiving an observation o_t of the scene, the belief of the cooperation c^i for vehicle i is updated using Bayes' theorem as

$$\hat{\theta}_{t+1}^i = \frac{p(o_t|\hat{s}_t^{\mathrm{ML}}, c^i = 1)\hat{\theta}_t^i}{p(o_t|\hat{s}_t^{\mathrm{ML}}, c^i = 0)(1 - \hat{\theta}_t^i) + p(o_t|\hat{s}_t^{\mathrm{ML}}, c^i = 1)\hat{\theta}_t^i}$$

This means that first, the scene is predicted for both hypotheses $c^i = 0$ and $c^i = 1$ using C-IDM. The likelihood $p(o_t|\hat{s}_t^{\mathrm{ML}}, c^i)$ of observing o_t given the most likely state and the cooperation level is then determined by adding Gaussian noise with a standard deviation of $1\,\mathrm{m}$ for positions and $1\,\frac{\mathrm{m}}{\mathrm{s}}$ for velocities to the predicted state. This Gaussian noise makes the proposed approach more resilient to model mismatch.

To reduce the computational cost, the belief is only updated for observed vehicles. Furthermore, the belief about the direct leading vehicle is not updated, since its cooperation level cannot influence the ego merge. Hence, a maximum of three cooperation beliefs is updated for each observation.

E. Reinforcement Learning with Cooperation Beliefs

The dimension of the belief varies with the number of vehicles on the scene. To simplify the training of the reinforcement learning agent, we choose a belief representation with a fixed size as input to the neural networks. This representation consists of the physical ego state $(x^{\mathrm{e}}, v^{\mathrm{e}}, a^{\mathrm{e}})$ and the position, velocity and cooperation estimate (x^i, v^i, c^i) of the four observed vehicles, as described in Section IV-A5. In total, this results in a 15-dimensional vector representation of the belief, where the belief is estimated using the method introduced in the previous section. The network architecture and DQN parameters are presented in Table II.

V. Experiments

We evaluate our approach in the merging environment described in Section IV-A. The scenario was implemented using the POMDPs.jl framework [27]. We consider three scenarios that differ in traffic density and desired velocities of other vehicles: The first scenario represents *moderate* urban traffic, while the other two represent *dense* urban traffic and *fast* traffic, respectively. In the moderate scenario, vehicles have a low desired velocity and the traffic density is moderate. The dense scenario uses the same desired velocity range, but the traffic density is increased. In contrast, the vehicles have higher desired velocities in the fast scenario, while the traffic density remains moderate. The parameters defining the scenarios are provided in Table I. In the dense and fast scenario, vehicles only spawn again with $p_{\mathrm{spawn}} < 1$ at the beginning of the lane after reaching the end. This slowly decreases the traffic density over time, simulating larger gaps that also occur in dense traffic occasionally.

	Traffic condition		
Parameter	Moderate	Dense	Fast
$N_{\min}$	4	8	5
$N_{\max}$	8	12	10
p_{spawn}	1.0	0.3	0.8
$v_{\mathrm{des,min}}$	$4\,\mathrm{m/s}$	$4\,\mathrm{m/s}$	$8\,\mathrm{m/s}$
$v_{\mathrm{des,max}}$	$6\,\mathrm{m/s}$	$6\,\mathrm{m/s}$	$12\,\mathrm{m/s}$

Table I: Scenario-specific parameters.

Parameter	Value
Neural network architecture	2 dense layers, 64 and 32 nodes
Activation function	ReLU (only on hidden layers)
Loss function	Huber loss
Optimizer	Adam [29]
Learning rate	10^{-4}
Batch size	200
Training steps	$3 \cdot 10^6$
Replay buffer size	$1 \cdot 10^5$
Target update frequency	5,000

Table II: Parameters used for training the DQN agent.

In our experiments, we train a DQN agent with experience replay on the belief MDP as described in Sections III-B and IV-E. We used the ReinforcementLearning.jl framework to train the agent. [28]. The hyperparameters of the DQN agent are given in Table II.

This agent is then used to guide the online belief space search of the MCTS-based algorithms using the strategies presented in Section III-C and serves as a baseline for the proposed methods. The belief updater presented in Section IV-D is used to update the agent's belief during the experiments but also in the belief space MCTS tree simulations.

To assess how well the proposed approach transfers to new, unseen environments, we train the DQN agent only in the moderate scenario and then evaluate all scenarios using this agent. Hence, our evaluation consists of one in-distribution experiment and two out-of-distribution experiments. For each of the three scenarios, we compare the following decision-making algorithms:

1) Belief Space Reinforcement Learning: The baseline for all MCTS-based algorithms is the plain DQN agent trained on the belief MDP, referred to as *Belief RL* [11].

2) Belief Space MCTS with Random Rollout Policy: This policy represents the standard MCTS formulation where the value of leaf nodes is estimated by rolling out the simulation with a random policy. We refer to it as *Random MCTS*.

3) Belief Space MCTS with Neutral Value Estimate: The *Neutral MCTS* assumes that no more rewards are collected after reaching a leaf node. While this makes it more difficult to find the sequence of actions that leads to the goal, it is computationally cheap.

4) Belief Space MCTS with Informed Rollout Policy: For the *Informed Rollout MCTS (IR MCTS)*, we use the greedy DQN policy to estimate leaf node values with rollouts.

5) Belief Space MCTS with Q-value Estimates and Action Prior: This method makes the most use of the trained models: Leaf node values are estimated using the learned value function and Q-value estimates are initialized with the learned Q-values. Additionally, the Boltzmann policy in Eq. (3) is used as a prior on the exploration term as presented in Eq. (2). We refer to it as *Q-Zero*.

6) Belief Space MCTS with Q-value Estimates: This policy represents an ablation of the previous one. It also uses the learned Q-value function to estimate leaf node values and to initialize Q-value estimates in new action nodes. However, it uses the standard UCT formula without the learned action prior for the tree policy. It is referred to as *Q-MCTS*.

7) Belief Space MCTS with Value Estimates: The last policy represents a further ablation and is referred to as *V-MCTS*. It only makes use of the learned model for estimating the value of leaf nodes as $V(b) = \max_a Q(b, a)$.

All the MCTS-variants use progressive widening of the belief space with parameters $k = \alpha = 0.5$. The exploration constant for UCT was set to $c_{\mathrm{uct}} = 50$ and the maximum depth for tree search and rollouts to $d = 30$ in all experiments. All algorithms have a computational budget of $1\,\mathrm{s}$ of online planning time. Note that it is not possible to use POMDP solvers like POMCPOW [30] on this problem, since the observation is modelled deterministically.

For each of the algorithms, we conduct 1000 experiments with random initial states, as described in Section IV-C. Policies are judged based on collision rate, timeout rate, and the average number of steps to reach the goal in successful episodes. Furthermore, we also report the average discounted and undiscounted cumulative reward.

VI. Results

Table III presents the results for the moderate scenario, on which the belief RL agent was trained. While belief RL performs well, the MCTS approaches not employing learned models are struggling: Random MCTS has both, collision and timeout failures, while neutral MCTS acts very conservatively and has a lot of timeout failures. The approaches making use of learned models as heuristics generally perform better in terms of reward than the classical MCTS variants. The best performance is achieved by Q-Zero, closely followed by its ablation Q-MCTS. Their performance almost matches the belief RL agent. IR MCTS also has a very high success rate but on average requires more steps to reach the goal. V-MCTS acts very conservatively and hence has almost no collisions but a significant number of timeouts.

For the dense traffic transfer environment, the general picture is very similar, as illustrated in Table IV. The MCTS methods using learned Q-value functions as guidance outperform the MCTS approaches not using learning. The belief RL agent now suffers from some collisions, since it cannot easily handle the increased traffic density. Its driving behavior was optimized for moderate traffic, hence, it results in a high-risk policy in dense traffic situations. For this reason, it reaches the goal quickly if successful but at the cost of many collisions. Despite the suboptimality of this policy, the MCTS algorithms can effectively make use of it as guidance to promising action sequences while still avoiding highly risky situations. In particular, Q-MCTS and Q-Zero require only slightly more steps to safely reach the goal, with Q-Zero achieving a lower collision rate than Q-MCTS.

As Table V shows, the results for the second transfer environment with fast traffic flow are similar to dense traffic. While belief RL again results in a high-risk policy regarding collisions, the MCTS-based algorithms can still use it to greatly improve their performance to a minimum of collisions without being conservative. In this scenario, Q-Zero reaches the goal significantly faster than Q-MCTS at a comparable level of safety.

VII. Conclusions and Outlook

This work presents a methodology for augmenting online planning algorithms with learned policies to overcome the curse of dimensionality in belief space planning. We use trained neural networks as heuristics to estimate the value of leaf nodes during tree search as well as to guide the tree policy to the most promising parts of the search tree. The efficacy of the method is shown in a cooperative merging scenario, including two transfer environments with different characteristics.

In our work, we use Monte Carlo Tree Search (MCTS) as the online planning component. However, the ideas can be transferred to other online planning algorithms, such as optimization- or sampling-based motion planners. Furthermore, our approach can be used in other highly interactive driving situations such as roundabouts or traffic weaving.

From our experiments, we conclude that MCTS can greatly improve its online planning performance by using learned models. The experiments on the transfer environments show that even if the policy itself is suboptimal, it can be used effectively to guide the tree exploration in MCTS. Our ablation studies show that using the learned model only to estimate the value of leaf nodes can result in over-conservative behavior. If additionally the Q-values of action nodes are initialized using the learned Q-value function, this acts as an initial prior on the UCT action selection and already results in a very good policy. The full Q-Zero method uses a Boltzmann policy based on the learned Q-values as a persistent action prior, which further improves the performance.

Another benefit of Q-Zero is that querying a small neural network is a faster operation than simulating a rollout of the environment. For this reason, Q-Zero and its ablations are able to perform more simulations within the same computational budget, leading to a performance superior to IR MCTS.

A promising direction for future research is to let the actions represent high-level maneuvers with an underlying low-level controller that guarantees safety, as demonstrated in an RL context by Mirchevska *et al.* [31]. Additionally, data-driven

Algorithm	Total reward	Disc. reward	Collision rate [%]	Timeout rate [%]	Number of steps
Belief RL	**97.0**	**82.2**	**0.1**	**0.0**	**17.3**
Random MCTS	54.9	36.8	3.7	4.7	34.8
Neutral MCTS	16.1	12.7	0.5	79.2	24.0
IR MCTS	82.5	58.9	0.7	0.3	34.6
Q-Zero	95.6	80.4	0.2	**0.0**	18.7
Q-MCTS	94.5	79.3	0.6	**0.0**	18.7
V-MCTS	68.3	50.9	0.2	24.2	31.7

Table III: Average metrics in the moderate traffic scenario.

Algorithm	Total reward	Disc. reward	Collision rate [%]	Timeout rate [%]	Number of steps
Belief RL	89.7	73.5	2.9	0.3	**20.0**
Random MCTS	54.9	27.8	3.0	0.1	45.7
Neutral MCTS	16.7	12.8	**0.0**	79.8	25.4
IR MCTS	77.9	51.9	0.6	**0.0**	36.5
Q-Zero	**94.7**	**76.8**	0.2	**0.0**	22.3
Q-MCTS	92.8	74.1	1.0	0.1	23.6
V-MCTS	70.0	43.5	**0.0**	22.2	48.5

Table IV: Average metrics in the dense traffic scenario.

Algorithm	Total reward	Disc. reward	Collision rate [%]	Timeout rate [%]	Number of steps
Belief RL	90.7	79.1	2.5	**0.0**	**14.5**
Random MCTS	50.9	30.7	4.7	2.9	39.1
Neutral MCTS	15.1	13.2	0.3	79.9	16.9
IR MCTS	77.6	56.7	0.5	**0.0**	29.6
Q-Zero	**96.2**	**83.1**	0.1	**0.0**	16.1
Q-MCTS	95.4	79.9	**0.0**	**0.0**	19.4
V-MCTS	31.7	15.5	**0.0**	57.4	66.0

Table V: Average metrics in the fast traffic scenario.

approaches can be used to learn more realistic cooperation models from human driving data.

Acknowledgements

This work has been supported by the BMWGroup, Germany. The authors also thank the German Research Foundation (DFG) for being funded within the priority program "SPP 1835 – Cooperative Interacting Automobiles (CoInCar)".

References

[1] E. Ward, N. Evestedt, D. Axehill, and J. Folkesson, "Probabilistic Model for Interaction Aware Planning in Merge Scenarios," *IEEE Transactions on Intelligent Vehicles*, vol. 2, no. 2, pp. 133–146, Jun. 2017, ISSN: 2379-8904.

[2] E. Schmerling, K. Leung, W. Vollprecht, and M. Pavone, "Multimodal Probabilistic Model-Based Planning for Human-Robot Interaction," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 3399–3406.

[3] C. Hubmann, J. Schulz, G. Xu, D. Althoff, and C. Stiller, "A Belief State Planner for Interactive Merge Maneuvers in Congested Traffic," presented at the IEEE International Conference on Intelligent Transportation Systems (ITSC), 2018.

[4] Z. N. Sunberg, C. J. Ho, and M. J. Kochenderfer, "The Value of Inferring the Internal State of Traffic Participants for Autonomous Freeway Driving," in *American Control Conference (ACC)*, Seattle, 2017.

[5] S. Le Cleac'h, M. Schwager, and Z. Manchester, "ALGAMES: A Fast Solver for Constrained Dynamic Games," in *Robotics: Science and Systems XVI*, Robotics: Science and Systems Foundation, Jul. 12, 2020, ISBN: 978-0-9923747-6-1.

[6] D. Fridovich-Keil, E. Ratner, L. Peters, A. D. Dragan, and C. J. Tomlin, "Efficient iterative linear-quadratic approximations for nonlinear multi-player general-sum differential games," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 1475–1481.

[7] S. Le Cleac'h, M. Schwager, and Z. Manchester, "LUCIDGames: Online Unscented Inverse Dynamic Games for Adaptive Trajectory Prediction and Plan-

ning," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5485–5492, Jul. 2021, ISSN: 2377-3766.

[8] L. Peters, D. Fridovich-Keil, V. Rubies-Royo, C. J. Tomlin, and C. Stachniss, "Inferring objectives in continuous dynamic games from noise-corrupted partial state observations," in *Proc. of Robotics: Science and Systems (RSS)*, 2021.

[9] D. M. Saxena, S. Bae, A. Nakhaei, K. Fujimura, and M. Likhachev, "Driving in Dense Traffic with Model-Free Reinforcement Learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 5385–5392.

[10] E. Leurent and J. Mercat, "Social Attention for Autonomous Decision-Making in Dense Traffic," Nov. 27, 2019.

[11] M. Bouton, A. Nakhaei, K. Fujimura, and M. J. Kochenderfer, "Cooperation-Aware Reinforcement Learning for Merging in Dense Traffic," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, Oct. 2019, pp. 3441–3447.

[12] M. Bouton, A. Nakhaei, D. Isele, K. Fujimura, and M. J. Kochenderfer, "Reinforcement Learning with Iterative Reasoning for Merging in Dense Traffic," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, Sep. 2020.

[13] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," presented at the AAAI Conference on Artificial Intelligence, 2018.

[14] B. Mirchevska, C. Pek, M. Werling, M. Althoff, and J. Boedecker, "High-level Decision Making for Safe and Reasonable Autonomous Lane Changing using Reinforcement Learning," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Maui, HI: IEEE, Nov. 2018, pp. 2156–2162, ISBN: 978-1-72810-321-1 978-1-72810-323-5.

[15] D. Kamran, T. Engelgeh, M. Busch, J. Fischer, and C. Stiller, "Minimizing safety interference for safe and comfortable automated driving with distributional reinforcement learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Prague, Czech Republic, Sep. 2021.

[16] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 7676 Oct. 2017, ISSN: 1476-4687.

[17] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, *et al.*, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," Dec. 5, 2017.

[18] C. Hoel, K. Driggs-Campbell, K. Wolff, L. Laine, and M. J. Kochenderfer, "Combining Planning and Deep Reinforcement Learning in Tactical Decision Making for Autonomous Driving," *IEEE Transactions on Intelligent Vehicles*, vol. 5, no. 2, pp. 294–305, Jun. 2020, ISSN: 2379-8904.

[19] M. Kochenderfer, *Decision Making Under Uncertainty - Theory and Application*, 1st. MIT Press, 2015, ISBN: 0-262-02925-1 978-0-262-02925-4.

[20] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, *et al.*, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012, ISSN: 1943-068X, 1943-0698.

[21] L. Kocsis and C. Szepesvári, "Bandit Based Monte-Carlo Planning," in *Machine Learning: ECML 2006*, vol. 4212, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 282–293, ISBN: 978-3-540-45375-8 978-3-540-46056-5.

[22] A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard, "Continuous Upper Confidence Trees," in *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, C. A. C. Coello, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 433–445, ISBN: 978-3-642-25566-3.

[23] R. S. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, Second edition, ser. Adaptive Computation and Machine Learning. Cambridge, MA London: The MIT Press, 2018, 526 pp., ISBN: 978-0-262-03924-6.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, ISSN: 0028-0836, 1476-4687.

[25] T. Tram, A. Jansson, R. Grönberg, M. Ali, and J. Sjöberg, "Learning Negotiating Behavior Between Cars in Intersections using Deep Q-Learning," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Nov. 2018, pp. 3169–3174.

[26] M. Treiber, A. Hennecke, and D. Helbing, "Congested Traffic States in Empirical Observations and Microscopic Simulations," *Physical Review E*, vol. 62, no. 2, pp. 1805–1824, Aug. 1, 2000, ISSN: 1063-651X, 1095-3787.

[27] M. Egorov, Z. N. Sunberg, E. Balaban, T. A. Wheeler, J. K. Gupta, and M. J. Kochenderfer, "POMDPs.jl: A Framework for Sequential Decision Making under Uncertainty," *Journal of Machine Learning Research*, vol. 18, no. 26, 2017.

[28] J. Tian and o. contributors, *ReinforcementLearning.jl: A reinforcement learning package for the Julia programming language*, 2020.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[30] Z. Sunberg and M. Kochenderfer, "Online Algorithms for POMDPs with Continuous State, Action, and Observation Spaces," in *Proceedings of the Twenty-Eighth*

International Conference on Automated Planning and Scheduling, AAAI, 2018, pp. 259–263.

[31] B. Mirchevska, M. Hügle, G. Kalweit, M. Werling, and J. Boedecker, "Amortized Q-learning with Model-based Action Proposals for Autonomous Driving on Highways," in *IEEE International Conference on Robotics and Automation*, Xi'an, China: IEEE, 2021, pp. 1028–1035.