Utrecht University

Jokke Mats Jansen

Master's Thesis

Artificial Intelligence
Faculty of Science

Supervisors

First: Dr. Shihan Wang
Second: Dr. Leendert van Maanen

2021

Abstract

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Chapter 2

Theoretical foundation

The problem examined in this thesis is assisted lane keeping with shared control by a potentially distracted human driver and an agent. The agent acts as an Advanced driver assistance system (ADAS) to assis the driver in keeping the car centered in its lane. The driver's attentiveness and the exact position of the car are unknown to the agent. In this chapter, the basic theoretical concepts which serve as a foundation to formally model the problem and to solve it are presented. The task involves sequential decision making, where every prior decision influences the following ones. Section 2.1 shows how sequential decision making tasks can be formulated using Markov decision processes (MDP). However, since the agent only observes partial information, uncertainty about the driver's attentiveness and the car's road position is involved. Section 2.2 introduces the Partially observable Markov decision process (POMDP), which is an extension of an MDP, accounting for partial observability of information. It is well suited to model the uncertainty involved in the problem. Solving POMDP is a difficult task. Section 2.3 discusses the key challenges involved in solving POMDP. Many solution approaches have been proposed. In section 2.4.1 an overview over proposed solvers is provided.

2.1 Sequential decision making

Lane keeping of a car is a sequential decision making task. Every steering action that is performed directly influences the choice of the best succeeding steering actions. MDP are well suited and widely used to model sequential decision making tasks. An MDP is a discrete time framework for a decision maker, the agent, to interact with an environment. At every time step, the environment is in a certain state, fully observable by the agent. The agent interacts with the environment by performing an action that determines the next state of the environment. The underlying assumption, the Markov property, is that the next state of the environment only depends on its current state and the agent's action. The transition to a succeeding state after an action has been performed does not need to be deterministic but can be probabilistic, accounting for randomness in the environment. After performing an action, the agent receives a numerical reward (also called return). The agent's goal is to maximize the cumulative reward it receives over time. An action that leads to a high immediate return is not optimal if another action leads to a higher cumulative reward in the long run. Thus, the agent needs to find an optimal policy that decides the best action to take in every state. In case the state transition probabilities are known to the agent, the optimal policy can be found using model-based techniques such as value or policy iteration. If the transition model is unknown, model-free

reinforcement learning can be applied to learn an optimal policy.

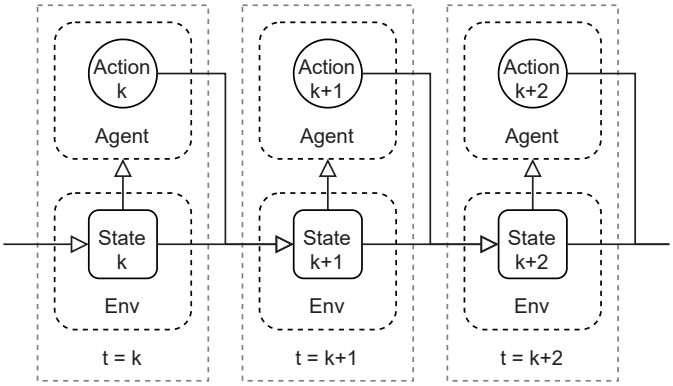

Figure 2.1: Markov decision process (MDP)

Assisting a human driver in the lane keeping task is essentially a sequential decision making task as well. However, the agent that is assisting the human driver does not know about the driver's internal psychological state, and therefore her attention. A distracted driver may steer poorly and needs assistance. But how can the agent tell whether the driver is distracted? Reading the driver's mind is not feasible and even if it were, it would be too invasive for this task. Instead, the agent needs to estimate the driver's internal state in order to act adequately. A POMDP is a generalization of an MDP that allows to plan under uncertainty. Even without observing the full state of the agent's environment, of which the driver is part of, a POMDP allows the agent to estimate the environment's true state using the partial information it observes. A POMDP serves as the foundation of this thesis. The lane keeping assistance problem this thesis aims to solve can be defined as a POMDP. First, a formal definition is needed.

2.2 Partially observable Markov decision process (POMDP)

The POMDP generalizes the MDP for planning under uncertainty. The environment's true state is unknown to the agent. It has to rely on observations with partial information about the environment's true state to choose its actions. Kaelbling et al., 1998 define a POMDP as a tuple (S, A, T, R, O, Z), where:

- S is the set of all possible states $s \in S$ of the environment. A state describes the environment at a time point. It must not be an all-encompassing description but must include all relevant information to make decisions. The state is hidden from the agent. This is the main difference to an MDP.
- A is the set of all possible actions $a \in A$ the agent can perform in the environment.

- $T : S \times A \times S \to [0, 1]$ defines the conditional state transition probabilities. $T(s, a, s') = Pr(s'|s, a)$ constitutes the probability of transitioning to state s' after performing action a in state s.
- $R : S \times A \to \mathbb{R}$ is the reward function providing the agent with a reward of $R(s, a)$ after performing action a in state s.
- O is the set of all possible observations $o \in O$. Observations are the agent's source of information about the environment, enabling the agent to estimate the environment's state.
- $Z : S \times A \times O \to [0, 1]$ defines the conditional observation probabilities. $Z(s', a, o) = Pr(o|s', a)$ represents the probability of receiving observation o at state s' after performing action a in the previous state.

At any time, the environment is in some state s. Unlike in the case of an MDP, the agent cannot directly observe the environment's state. Instead, the agent receives an observation o that provides partial information about the current state. The agent uses the observations it perceives over time to estimate the true state of the environment in order to choose adequate actions. At any time step t, it has to take into account the complete history h_t of actions and observations until t:

$$h_t = \{a_0, o_1, ..., o_{t-1}, a_{t-1}, o_t\} \tag{2.1}$$

Keeping a collection of all past observations and actions is very memory expensive. A less memory demanding alternative is to only keep a probability distribution over the states at every step, called a belief b. $b(s, h)$ denotes the probability of being in state s given history h.

$$b_t(s, h) = Pr(s_t = s|h_t = h) \tag{2.2}$$

The belief is a sufficient statistic for the agent to form a decision about its next action (Smallwood & Sondik, 1973). Thus, only the belief needs to be kept and can be recursively updated whenever an action is performed and a new observation arises. The agent starts with an initial belief b_0 about the initial state of the environment. At every subsequent time step, the new belief b' can be recursively calculated based on the previous belief b, the last action a and the current observation o. The previous belief can then be discarded as the history it represents is no longer up-to-date. For an exact update of the belief one can apply the Bayes theorem:

$$
\begin{aligned}
b'(s') &= Pr(s'|o,a,b) \\
&= \frac{Pr(o|s',a,b)Pr(s'|a,b)}{Pr(o|a,b)} \\
&= \frac{Pr(o|s',a)\sum_{s\in S} Pr(s'|a,b,s)Pr(s|a,b)}{Pr(o|a,b)} \\
&= \frac{Z(s',a,o)\sum_{s\in S} T(s,a,b)b(s)}{Pr(o|a,b)}
\end{aligned}
\tag{2.3}
$$

The agent chooses its actions based on its belief according to its policy π. The agent's policy defines the action to choose at any given belief state. It describes the strategy for every possible situation the agent can encounter. Solving a POMDP consists in finding an optimal policy π^* that maximizes the the cumulative reward obtained over some time horizon N starting from initial belief b_0 using a discount factor $0 \leq \lambda \leq 1$:

$$
\pi^* = argmax_\pi \; E\left[\sum_{t=0}^{N}\sum_{s\in S} b_t(s)\sum_{a\in A}\lambda^t R(s,a)\pi(b_t,a)|b_0\right] \tag{2.4}
$$

The return that is gained by following a policy π from a certain belief b can be obtained with the value function $V^\pi(b)$:

$$
V^\pi(b) = \sum_{a\in A}\pi(b,a)\left[\sum_{s\in S} b(s)R(s,a) + \lambda\sum_{o\in O} Pr(o|b,a)V^\pi(b')\right] \tag{2.5}
$$

The optimal policy π^* maximizes $V^\pi(b_0)$. For any POMDP there exists at least one optimal policy.

2.3 Key challenges

2.3.1 Curse of dimensionality and curse of history

Computing an optimal policy for a POMDP is challenging for two distinct but interdependent reasons (Pineau et al., 2006). On the one hand, there is the so-called curse of history: Finding an optimal policy is like searching through the space of possible action-observation histories. The number of distinct histories grows exponentially with the size of the time horizon. Therefore, planning further into the future increases the computation complexity exponentially. While finding an optimal policy can be relatively easy for short histories, it becomes computationally infeasible for larger time horizons. On the other hand, there is the curse of dimensionality: The belief space is ($|S|$-dimensional. Therefore, the size of the belief space, representing the number of states in a POMDP, grows exponentially with $|S|$.

The task of finding an optimal policy for a finite horizon POMDP is PSPACE-complete (Papadimitriou & Tsitsiklis, 1987). Solving POMDP to optimality is computationally infeasible with a large state space or time horizon. For this reason, approximate algorithms are often applied.

2.3.2 Unknown transition and observation probabilities

For many problems, it is difficult or impossible to know the probability distributions T or Z explicitly. This is also the case for the shared control lane keeping scenario assessed in this thesis. Neither the transition probabilities, nor the observation probabilities are known a priori. The belief update method using Bayes' theorem presented in Equation 2.3 is not computable without knowing the probabily distributions explicitly. However, exact updates are too complex for problems with a large state space in any case (Silver & Veness, 2010). Some solution approaches circumvent the problem of unknown transition and observation probability distributions by only requiring a generative model that can sample state and observation transitions. A generative model can stochastically generate a successor state, reward, and observation, given the current state and action. Thereby, it implicitly defines the transition and observation probabilities, even if they are not explicitly known. The generative model used in this thesis is described in detail in Section 3.2.5.

2.4 Algorithms to aproximately solve large POMDP

2.4.1 Overview of aproximate POMDP solvers

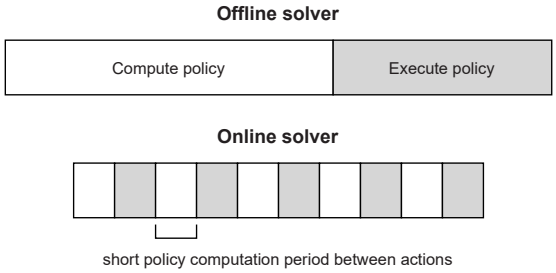

Figure 2.2: Comparison of offline and online solving procedure

There are two general approaches to solve POMDP: offline and online. Online solvers compute the optimal policy prior to execution for all possible future scenarios. Their advantage is that once the policy is found, policy execution is fast as there isonly a very minimal, negligible time overhead. However, offline planning is hard to scale to complex problems as the number of possible future scenarios grows exponentially with the size of the time horizon (curse of history). Furthermore, while the performance for small to medium sized POMDP can be

quite good, computing the policy may take a very long time, and even small changes in the dynamics of the environment require a full recomputation (Ross et al., 2008). Online solvers interleave planning and plan execution. At every time step, only the current belief is considered to compute the next optimal action by searching ahead until a certain depth is reached. On the one hand, the scalability is greatly increased. On the other hand, sufficiently more online computation than with offline planning is required. The amount of available online planning time at each time step limits the performance.

As discussed in Section 2.3.1, solving POMDP to optimality exactly is only feasible for small discrete POMDP. Larger POMDP are usually solved approximatelty. A wide range of offline and online approximate solvers is available. For discrete POMDP, most effective successful offline approximate solvers apply a form of Point-based Value Iteration (PBVI), where only a representative subset of the belief space is considered to approximate the value function (Pineau et al., 2003). State-of the art methods include Perseus, and Heuristic Search Value Iteration (HSVI). Perseus chooses belief states to consider by randomly sampling trajectories from an initial belief (Spaan & Vlassis, 2005). It is sufficiently more compute efficient than PBVI but can suffer from a slow convergence behavior (Shani et al., 2013). HSVI improves on Perseus potential slow convergence in large domains by constructing a belief search tree, maintaining the order in which beliefs are visited, in order to perform value updates in reverse order incrementally (Smith & Simmons, 2004).

Even the most advanced offline solvers reach their limit when dealing with large POMDP. Moreover, the state space of the POMDP for the shared control lane keeping task considered in this thesis has many continuous state variables (see Section 3.1.1.1). Most offline solvers are not suited to be used with continuous states. There have been offline methods proposed to solve continuous POMDP, for example in Bai et al., 2014, and Brechtel et al., 2013. However, they have only been used for problems with relatively few state variables and are not considered further in this thesis.

When it comes to online solvers, the paradigm of only trying to find a good local policy for the current belief state makes them sufficiently more efficient. The general approach is to construct a search tree rooted at the current belief, evaluating all possible further actions and observations. The methods differ in how the tree is explored to efficiently generate a good approximately optimal action to perform at the current belief. After this action has been performed in the real environment, the agent receives a reward and observation. On this basis, it updates its belief and the process repeats from the new belief.

Recently, very promising results have been obtained for large POMDP using Monte Carlo sampling. Exploring the entire search tree is not feasible for deep time horizons because of the curse of history and the curse of dimensionality (see Section 2.3.1). Monte Carlo tree search (MCTS) methods address this issue by using a generative model to sample state transitions and observations (see Section 2.3.2). By doing so, only a subset of histories is considered. The curse of dimensionality can be overcome in a similar fashion. Instead of evaluating all belief states, the start states for the search tree are sampled from the belief space.

The number of belief states to consider can thereby be drastically reduced.

2.4.2 Online POMDP solving using Monte Carlo tree search

The approach of using Monte Carlo sampling for both the choice of evaluated histories and estimating the belief was first applied in the Partially observable Monte-Carlo Planning (POMCP) algorithm by Silver and Veness, 2010. POMCP constructs a search tree of sampled histories. Each node stores its estimated value and sampled states that it corresponds to. The estimate is given by the return gained from forward simulations using the generative model during which the node is visited. The exploration is controlled using the Upper Confidence Bounds 1 (UCB1) (Auer et al., 2002) algorithm for action selection. The key idea is to aproximate the belief space using the same set of states that have been encountered during the forward search. Instead of representing the belief as a probability distribution over the states, it is given by the collection of states at the nodes. The underlying assumption is that if a state is more likely, it will be visited more often during the sampled trajectories. The relative number of times a state occurs in the collection defines its probability. Using this belief representation alleviates POMCP from expensive belief update calculations. POMCP has successfully been applied to approximately solve large POMDP. It is the solver used in this thesis. A detailed definition follows in Section 3.2.1.

The Determinized sparse partially observable tree (DESPOT) algorithm by Ye et al., 2017 is a similar approach that can be seen as an evolution of POMCP. DESPOT is efficient as only a fixed number of sampled scenarios are considered. A scenario is a determinized trajectory in the belief tree that is defined in advance. At every depth of the belief tree, all actions but only a subset of resulting observations are considered. Thereby, the observation space is simplified. DESPOT's main advantage over POMCP is the ability to overcome POMCP's relatively poor worst-case behavior (Coquelin & Munos, 2007) caused by the UCB1 algorithm's tendency to overfit. DESPOT circumvents this by using regularization in the value function. Moreover, DESPOT is an anytime algorithm, building its tree incrementally. In addition to its value, at every node upper and lower bounds for its performance are maintained. First, a suboptimal policy is searched using heuristic search (Smith & Simmons, 2004) and then it is incrementally improved upon. Branch-and-bound pruning is performed by pruning action nodes from the tree if their expected value is lower than the lower bound of another action. There are further extensions of DESPOT: HyPDESPOT is a parallelized version of DESPOT with significant performance enhancements (Cai et al., 2018). DESPOT-α further improves DESPOT's capability to handle very large observation and state spaces (Garg et al., 2019). And DESPOT-IS applies importance sampling to account for very rare events that are hard to sample (Luo et al., 2018). Unfortunately, DESPOT and its derivatives require the observation probability Z to be explicitly known. This is not feasible for the shared control lane keeping task considered in this thesis.

2.4.3 Solving continuous POMDP

A continuous POMDP has continuous state, observation, and action spaces. This is the case for the scenario of lane keeping with a human in the loop that is examined in this thesis (see Section 3.1). Hence, a solver that can handle continuous POMDP is required.

The aforementioned algorithms POMCP and DESPOT can natively be applied for continuous state spaces (Goldhoorn et al., 2014). As both use a collection of sampled states as their belief representation, they do not have to be adjusted to be able to represent beliefs about continuous states. These can also just be added to the collection if they occur during sample trajectories. It is unlikely that two identical continuous states are inserted. Therefore, the individual count of a state in the belief is rendered meaningless. Nevertheless, if the agent samples multiple similar situations, the corresponding states are also similar. For the continuous case, the assumed probability of being in a certain state is be given by the number of belief states that are close to it.

However, for continuous action and observation spaces the MCTS search tree used in POMCP and DESPOT degenerates. Only a single layer of nodes can be realized as every search leads to a new branch (Sunberg & Kochenderfer, 2018). It is possible to discretize the action and observation spaces to bypass this limitation (Goldhoorn et al., 2014). Thereby, the continuous spaces are transformed into a discrete representation and the traditional methods are applicable. Action and observation discretization is performed in this thesis to be able to use POMCP. The details are discussed in Section 3.2.2.

Solvers that work with fully continuous POMDP without discretization have been developed. POMCPOW is an extension of POMCP using progressive widening (Sunberg & Kochenderfer, 2018). It uses weighted belief updates and limits the amount of observations considered during planning. Observations are only gradually added to the lookahead tree as planning progresses. Lazy Belief Extraction for Continuous Observation POMDPs (LABECOP) is another recent approach (Hoerger & Kurniawati, 2020). It is based on MCTS as well but avoids limiting the number of considered observations. As promising as these methods are, they again require the observation probabilities Z to be known explicitly and are therefore not applicable to the problem addressed in this thesis.

Chapter 3

Methodology

This chapter formally defines the POMDP used to model the shared-control lane keeping task that is considered in this thesis. Furthermore, the chosen solution approach is presented. Section 3.1 begins with an overview over the three components comprising the problem: The driving simulator serving as the environment, the driver model representing the human driver, and the agent assisting the driver. Section 3.1.1 describes how the The Open Racing Car Simulator (TORCS) is used to simulate the dynamics of a car driving on a highway. The simple driver model that we use to simulate human driving behavior is specified in Section 3.1.2. The agent employs the Partially observable Monte-Carlo Planning (POMCP) algorithm to solve the POMDP online. A detailed explanation of how the algorithm is applied is provided in Section 3.2.1.

3.1 Lane keeping with a human in the loop as a POMDP

The problem addressed in this thesis is an assisted driving lane keeping task, where a human driver shares control with an agent over the steering of a car driving on a highway. The goal is to keep the car centered in its lane. Both the agent and the driver have only lateral control; they can steer the car but the car's speed is fixed. The driver can be attentive or distracted and alternates between the two states. The general assumption is that an attentive driver shows (nearly) optimal steering behavior, while a distracted driver steers suboptimally and needs assistance. The agent, however, cannot observe whether the driver is attentive or not. Moreover, it only receives partial sensory information about the position of the car on the road. To fulfill the goal of consistently keeping the car centered in the lane, the agent has to effectively estimate the car's true position and the driver's state of attentiveness according to the information it receives over time. Based on its estimate, the agent determines what actions the driver is likely going to take and where it believes the car to be positioned on the road. It can then plan ahead and select adequate steering actions.

The problem can be formulated as a POMDP as follows (see Section 2.2 for a general definition of a POMDP):

- The overall state space S is composed of all possible states for the car and the driver. The state of the car is given by its current position on the road and the forces which it is currently exposed to (see Section 3.1.1.1). In the case of the driver, her current attentiveness, and the remaining duration for which she stays in this state of attentiveness are cruicial (see Section 3.1.2).

- The action space A consists of all steering actions *the agent* can perform (see 3.1.1.2). In the experiments, two different sets of actions are referred to: First, a full action set, which enables the agent to overrule and effectively reverse the driver's actions completely. Second, a reduced action set containing only moderate steering actions.
- The reward function R is based on the car's distance to the center of the lane and its relative angle to the road path (see Section 3.1.3). The attentiveness of the driver is not considered. The agent is expected to estimate it based on the driver's behavior alone. However, the reward is correlated with the quality of the agent's estimate as its actions can only lead to optimal steering behavior with a correct estimate.
- The observation space O includes all possible observations. The agent observes sensory information about the car's current distance to the road and relative yaw angle (see 3.1.1.3). Moreover, the driver's last action is observed. At any time step, the agent only receives the action of the driver for the last time step. Thereby, part of the planning process can be performed during policy execution (see Section 3.1.2).
- The conditional state transition probabilities T and the conditional observation probabilities Z are not explicitly given but implicitly defined by TORCS and the driver model. Each offer an interface to be used as a generative model by the agent (see Section 3.2.5).

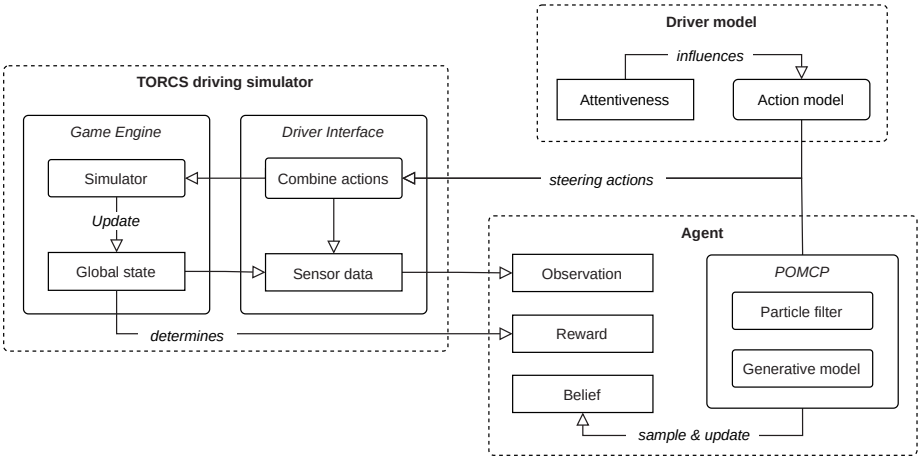

Figure 3.1: Overview of the modules used to represent and solve the shared control lane keeping POMDP

Rather than experimenting with real humans and a real car, simulation models are is used for both the car and the human driver in experiments. Figure 3.1 gives an overview of the three distinct modules employed to represent the

problem as a POMDP and solve it: First, the racing car simulator TORCS (Espié et al., 2005) simulates the dynamics of a car driving on a highway. Second, the driver model substitutes the human driver. Third, the agent applies POMCP in order to solve the POMDP online. The modules are described in detail in the following sections.

3.1.1 TORCS as a highway driving simulator

The Open Racing Car Simulator (TORCS) is an open-source car driving simulator (Espié et al., 2005). As the name suggests, TORCS was initially developed to simulate racing car tournaments. However, as racing cars are fundamentally also just cars and everything, including the tracks, is highly customizable, highway driving can be simulated just as well. Part of TORCS is a comprehensive and realistic discrete-time simulation engine to simulate car dynamics, as well as an API for computer-controlled drivers, so-called robots.

TORCS is used for two purposes in this thesis. First, its simulation engine serves as the environment for the agent and driver. The steering actions of the agent and driver are combined in a TORCS *robot* satisfying the driver interface. TORCS maintains the car's true state and updates it based on the combined steering action. Second, the simulation engine is used as a generative model to sample state and observation transisitions for the forward search performed during the agent's online MCTS policy computation (planning).

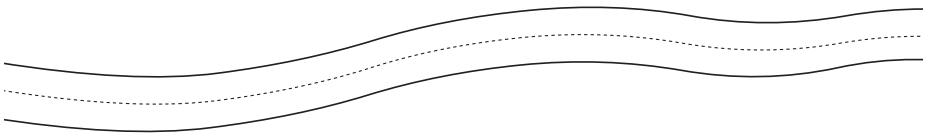

Figure 3.2: Course of the road of a section of the TORCS highway track used for experiments.

Typical racing tracks for TORCS have sharp road bends and a generally wide road which can have different widths in its course. This does not reflect a highway driving scenario well. Therefore, a custom highway track is used instead (see Figure 3.2). The custom track only has moderate road bends and a constant lane width of 3.75m, as it is common in Europe (Schoon, 1994). The road is completely flat and there are no other cars on it. Moreover, a fixed speed of 80 km/h is set during experiments.

3.1.1.1 Car state and its updates

TORCS data model for the car state is too extensive to list here in full[1]. Most important are the car's position on the track, its current velocity and its accelleration. Among others, there are additional attributes for the state of

[1] See tCar struct in the TORCS API documentation

transmission and engine, the friction and spin of the wheels, and aerodynamic influences such as the current air speed. The values in the state are continuous. Since the problem is a lane keeping task, any state representing a lane departure is consiered a *terminal state*, resulting in the end of an experiment. The car is considered to have departed from its lane if the center of car lies more than 20cm beyond side lane markings.

The state is updated by the simulation engine every 0.002 seconds of simulated time (this is the driving time that is simulated, not real time). By default, robots provide new control actions to the car every 0.02 seconds. However, for the experiments in this thesis, a rate of 0.1 seconds is chosen. This makes the ride less smooth but reduces the frequency in which the agent has to plan ahead, reducing the amount of planning time. The steering action is repeated until a new one is provided.

3.1.1.2 Actions

Accelleration, braking, and gear changes are performed by a simple controller intended to keep the speed constant at at all times. The human driver and the agent share control of the steering wheel. The steering input of the driver a_{driver} and agent a_{agent} are added to $a_{car} \in [-1, +1]$ using Equation 3.1. A steering action of -1 means steering fully to the right (159 degrees), and an action of $+1$ has the effect of fully steering to the left (21 degrees).

$$a_{car} = \min(-1, \max(1, (a_{driver} + a_{agent}))) \tag{3.1}$$

Table 3.1 shows the discrete actions for the driver and the agent. The discrete values have been chosen empirically. More actions allow for a more precise control. However, the number of actions is the branching factor for the search tree the agent constructs to solve the POMDP, and thus has a big impact on the complexity of the search problem. Therefore, a compromise between precision and performance is made. Because minor actions are more likely and generally preferable in a highway driving scenario, the resolution is higher for small steering actions.

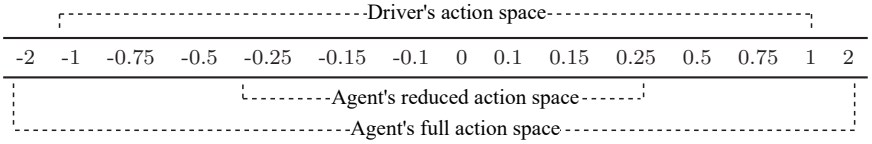

Table 3.1: Discrete steering actions for the driver and the agent.

If the driver is distracted while the car is in a road bend, in the most extreme situation, she could potentially steer into the opposite direction of where she needs to steer in oder to keep the car centered in its lane. In this case, to correct the driver's incorrect action, the agent needs to be able to effectively reverse the driver's action. Therefore, the action space of the agent is extended by

plus two and minus two. The agent is evaluated in experiments with the full action set but also with a small subset of the full action set including only minor actions (reduced action space in Table 3.1). The motivation behind this is the assumption that strong steering actions are seldomly needed while driving on a highway and leaving them away might reduce the planning complexity.

3.1.1.3 Observations

In a POMDP, the agent has only partial information about the true state of the environment. The information the agent has stems from observations it makes by interacting with the environment. For the shared control lane keeping task, the agent is given an observation after every steering action (after every 0.1 seconds of simulated time). It observes the current horizontal position of the car (lane centeredness), the car's relative yaw angle with respect to the track axis, and the driver's action from the last time step. The driver's current steering action is not observable. The agent has to estimate the most likely next action of the driver by considering the history of past observations.

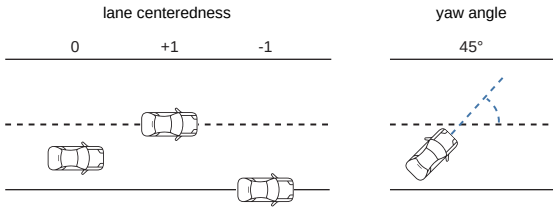

Figure 3.3: Illustration of lane centeredness and yaw angle values

The observations are discrete. The lane centeredness is originally in a continuous interval of $(-\infty, \infty)$, with values in between -1 (right lane border) and +1 (left lane border) denoting that the car is within its lane, and everyting beyond standing for an off-track position (see Figure 3.3). 99 discrete values represent evenly spaced segments between -1 and +1, and two additional values stand for left off lane and right off lane positions. The angle is transformed from an interval of $[-\pi, +\pi]$ radians to 101 discrete values representing evenly spaced segments. The driver's action is discretized as stated in the last section.

3.1.2 Driver model

Instead of performing experiments with an actual human driver, the driver is simulated using a stochastic driver model. The driver model determines when a driver is attentive or becomes distracted, for how long the attentive or distracted period persists, and what actions the driver takes. The driver model is rather simple. If the agent can plan successfully with a simple driver model, this serves as an initial confirmation that the solution approach is promising. The focus of this thesis is not a realistic driver model.

3.1.2.1 Configurations

Three different driver model configurations with increasingly complex dynamics are used in the experiments:

1. **Simple driver model:** The simplest model steers optimally when the driver is attentive, and if it is in a distracted state, the model repeats the last attentive steering action until the driver regains her attentiveness. A distracted driver's ability to notice changes in the course of the road is limited because of a reduced situational awareness and therefore she does not adjust to road changes like an attentive driver would (Kass et al., 2007; Young et al., 2011).

2. **Steering overcorrection:** Another, more complex scenario, is based on the assumption that a driver who has just regained attentiveness after having been distracted performs an overly strong steering correction during her first action, and thereby overshoots. The first action of the attentive driver is increased by a random amount between 10 and 25 percent. The overcorrection is added to the continuous driver action before discretization.

3. **Steering overcorrection and noise:** The last, most complex scenario introduces action noise. It acts just like the second scenario, just with an additional random noise between five and 20 percent. The action can thereby become five to 20 percent stronger or weaker. The noise is intended to make the driver less predictable. It is also added before the action discretization is applied.

The actions of the driver are discretized by mapping their continuous values to the closest vaue in the discrete action space as it can be seen in Table 3.2.

Action	-1	-0.75	-0.5	-0.25	-0.15	-0.1	0	0.1	0.15	0.25	0.5	0.75	1
From	-1	-0.875	-0.625	-0.375	-0.2	-0.125	-0.05	0.05	0.125	0.2	0.375	0.625	0.875
To	-0.875	-0.625	-0.375	-0.2	-0.125	-0.05	0.05	0.125	0.2	0.375	0.625	0.875	1

Table 3.2: Driver action discretization. For every action, the smaller number from the corresponding interval is excluding, while the larger one is including.

3.1.2.2 Driver state and its updates

The driver state consists of two variables: The current state (attentive or distracted), and the duration it remains in its current state. The driver model is initially set to an attentive state. The duration it remains in this state is randomly chosen but lies between one second (10 actions) and 5 seconds (50 actions) of simulated time. Every 0.1 seconds, an action is chosen depending on the driver's current state. Afterwards, the remaining duration is decremented by 0.1. When the time runs out for the curren state, the state reverses; an attentigve driver becomes distracted and a distracted driver regains attentiveness. The

duration for the resulting state is randomly chosen again. The process repeats until the experiment is over.

3.1.3 Reward

The reward function defines the goal of the agent. For the task of lane centering, two sub-goals need to be considered: First, the car is supposed to stay as close to the lane center as possible. Second, the car's yaw angle (the direction into which the car is headed) should be as close to the track axis angle as possible.

Equation 3.2 shows the reward function for the agent that incorporates both targets. The relative yaw angle is denoted as θ, and ϕ represents the lane cen0teredness (see Figure 3.3). A lane centerednss of zero means the car is centered. If the car is on the left most side of the lane, the value equals one, for the right most side it equals minus one. The agent receives the maximum reward if the car is in the middle of the road, while its relative yaw angle is equal to zero. Similar reward formulations have been used successfully before (Kővári et al., 2020; Graves et al., 2020). The attentiveness of the driver is not directly included in the reward function. However, it is implicitly considered. If the agent performs an action that leads to a suboptimal combined steering action, it is penalized by receiving a lower reward. Thereby, if the agent acts when the attentive driver behaves optimally, it is punished indirecly.

$$R = \begin{cases} \cos\theta + |\phi|, & \text{if } \phi \in [-1, +1] \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

3.2 Solution approach using the POMCP algorithm

3.2.1 General POMCP definition

The key idea of Partially observable Monte-Carlo Planning (POMCP) is to use Monte Carlo sampling both to sample start states from the belief and to sample histories using a generative model (Silver & Veness, 2010). POMDP is an anytime online solver. For an online solver, policy computation (planning) and execution (acting) are intertwined. An anytime algorithm builds the solution incrementally. It can be stopped at any time and there will be a solution, albeit it might not be very good if the algorithm is stopped early.

The number of histories to consider in a POMDP grows exponentially with respect to the depth of the planning horizon. This is called the curse of history. POMCP overcomes this limitation by using a generative model to sample state transitions. By doing so, only a subset of histories is considered; the size of the belief tree is reduced and the curse is *broken* (see Figure 3.4). Furthermore, there is the curse of dimensionality. The belief space has the same dimensionality as the number of states. Thus, the number of beliefs to consider grows exponentially with the number of states. POMCP uses the states encountered during the construction of the tree to represent the belief. Start belief states are sampled

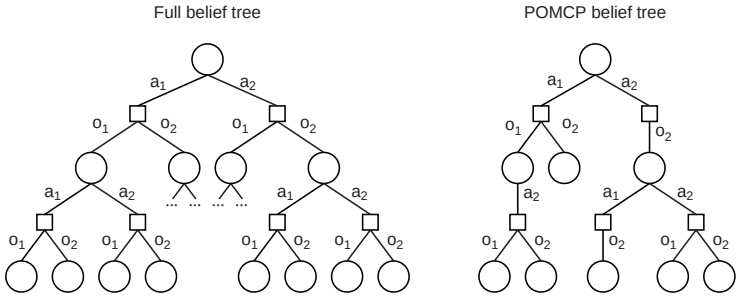

Figure 3.4: Contrasting a full belief tree (left) with a POMCP belief tree (right) for a POMDP with two actions and two observations and a time horizon of two. A belief is stored at every circle node. The full belief tree has 21 belief nodes, while the POMCP belief tree has just nine. The number of nodes of the full belief tree grows exponentially with the time horizon (curse of history), whereas the POMCP belief tree only contains a number trajectories which have been sampled from a generative model. By performing the sampling, the size of the tree is reduced and the curse is *broken*.

from these states, effectively limiting the number of considered belief states and thereby also breaking the second curse.

POMCP constructs a search tree representing histories h of actions and observations. At each node, $N(h)$ stores the number of times the node and thereby the coresponding history h has been visited during the sampled trajectories simulated with the generative model. $V(ha)$ gives the action node's expected value that is approximated by the average return of simulations starting at history h and performing action a. At every observation node, the belief over the states is maintained by employing a particle filter. Each observation node stores a collection of all states that led to the represented observation during planning. Whenever an observation occurs, the corresponding state is stored in this collection. The states in the collection are called particles and together the particles represent the agent's belief B(h) at the corresponding observation node. The more likely a belief state is, the more often it occurs as a particle in the belief. By using the particle filter method, expensive belief update calculations are not necessary. The collection of states alone aproximates the posterior probability distribution for the belief.

Figure 3.5 illustrates the process of POMCP. The algorithm starts with an initial belief about the environment. If the belief for the current history h_{real} does not contain particles in the beginning of a planning episode, the agent has lost track of the environment's state completely. One could construct a new belief by sampling the state space in this case. However, for the car driving scenario, this approach is too inaccurate. Accessing the real state of the environment to build the belief would be cheating. Instead, we consider the planner to have failed and select actions randomly from this point on.

To select which action to perform in the real environment, a fixed number of

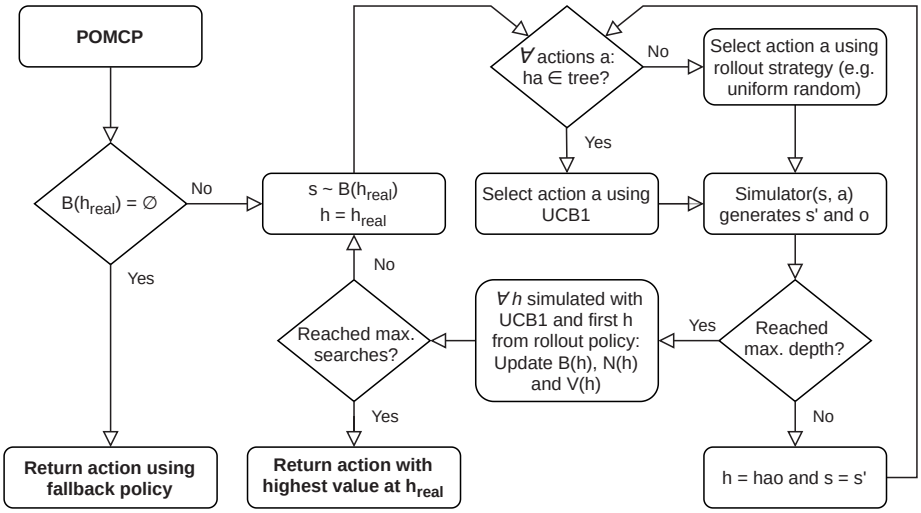

Figure 3.5: Flow chart illustrating the Partially observable Monte-Carlo Planning (POMCP) algorithm

forward searches is performed from the current history (often a certain maximum planning time is used alternatively). During these searches using the generative model, the belief and the expected action values are updated. After all searches are complete, the action a_{best} with the highest value at the current history h_{real} is returned. After this action is executed in the real environment, with an observation o_{last} the tree can be pruned. Only the nodes from history $h_{real}a_{best}o_{last}$ onward stay relevant as all other histories are rendered impossible. Then, the process repeats from the new history.

The number of searches that is performed during the planning has a substantial impact on the quality of the derived policy. If the belief state is correct, POMCP is proven to converge towards the optimal policy if the number of searches, and therefore the number of visits at each node, approaches infinity (Silver & Veness, 2010). Unfortunately, the computational complexity is exponential with respect to the number of performed forward searches. However, for a good policy, no infinite amount of searches is necessary. The performance is expected to increase with the number of searches until convergence occurs.

With the number of searches, the computational complexity

The start state for each search is sampled from the belief at the curret history. The search tree is searched in two stages: Simulation and rollout. As long as the search tree contains child nodes for all actions at the currently considered history, the simulation stage is active. During the simulation stage, no new nodes are added to the tree. The tree is searched using the Upper Confidence Bounds 1 (UCB1) algorithm to select actions (Auer et al., 2002). UCB1 chooses actions by the principle of optimism in the face of uncertainty. Even with just little knowledge, the algorithm selects the best action greedily. If this optimistic guess

turns out to be correct, the algorithm can further continue to exploit this action and regret is+ kept to a minimum. If the action leads to a bad return, its value is assumed to deteriorate quickly, allowing the algorithm to select an alternative action. Exploration is controled by enhancing the value of rarely-tried actions with a fixed exploration constant.

State transitions are simulated using the generative model. Given the current history and chosen action, the generative model returns a successive state s', observation o, and reward r. The successive state s' is then added to the belief at the observation node corresponding to o and the count for the current history is incremented. The search continues from s' in the same manner.

If any history is visited for the first time during the simulation stage, the algorithm continues in the rollout stage. First, all action nodes are initialized with initial counts an values. These are usually zero, unless preferred actions are used (See Section 3.2.4). Then, the history is *rolled out* further using uniformly random action selection and the generative model. The process is continually repeated using the succeeding states from the generative model until a maximum depth is reached in the tree. During the rollout, no further nodes than the ones just initialized are added to the tree. The tree's growth is thereby limited to one level of depth per search. The main purpose of the rollout is to form a first estimation of the newly encountered history. After every search, the values at all nodes encoutered during the search are updated by backpropagating the rewards through the tree.

3.2.2 Action and observation space discretization

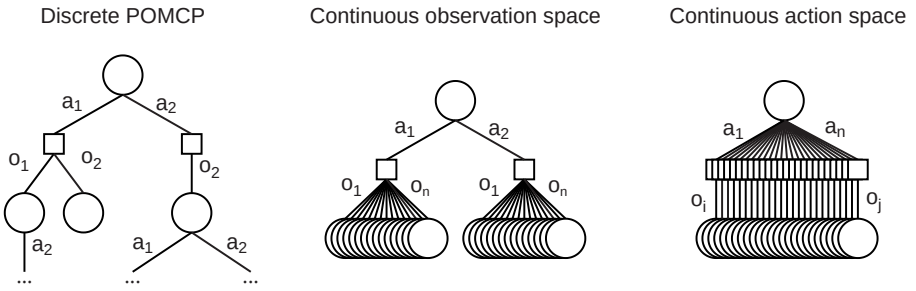

Figure 3.6: Comparison of POMCP belief trees with discrete observations (left) and continuous observations (right) with two actions.

POMCP is not suited to solve continuous POMDP. However, using POMCP with a continuous states is possible as the particle filter approach can still provide a good approximation of the belief as long as the number of samples is large enough. To account for continuous action and observation spaces, discretization is necessary. Figure 3.6 shows how POMCP behaves when tasked with solving POMDP with continuous observation or acrtion spaces. If the observations are continuous, the search tree cannot extend beyond the first observation layer as

most likely, every observation is unique and thus, no history will ever be visited twice. In the case of continuous actions, the chance of exectuing the same exact action twice is very low. Therefore, in this case likewise no history is reached twice. Planning becomes impossible. However, POMCP can be successfully applied with continuous POMDP by discretizing the action and observation spaces (Goldhoorn et al., 2014).

The action space is discretized as outlined before in Section 3.1.1.2 and the discretization of the observation space is defined in Section 3.1.1.3. A balanced discretization resolution is chosen empirically. A too fine grained discretization leads to very wide belief trees and can thereby hinder convergence, while a coarse discretization increases the convergence probability but comes with a lower precision in planning.

3.2.3 Particle deprivation and particle injection

Particle filter approaches, POMCP included, can fail due to a phenomenon called particle deprivation. Because of the random nature of the process, the belief can sometimes converge towards a state that is far from the environment's true state. Particles that differ from the converged state have a low probability to be selected while sampling (low relative count). Hence, with each iteration, they become scarcer until they are completely erased from the belief. At this point, the agent is sure to be in an erroreous state and cannot recover anymore. Particle injection (also called particle reinvigoration) is a method to counteract this problem by introducing a number of random particles to the belief at each iteration (Kochenderfer et al., 2021). While this reduces the accuracy of the belief, it prevents its complete convergence towards a wrong state.

Particle injection is used to increase the variance of the belief about the driver model state. Only observable information is used. Concretely, particle injection is implemented by adding driver model states with a random number of remaining actions and the same action as the one that was last observed. The number of remaining actions can be lower than the minimum defined in Section 3.1.2 because this limit is only intended for initial sampling and the true remaining number of driver actions in a particular state might be lower after having performed actions already. Like in the original POMCP paper, the amount of transformed particles that are added before each planning step is 1/16 of the number of searches. The particles can be added during policy execution, and therefore, do not influence planning time.

3.2.4 Preferred actions

Preferred actions are a way to pass domain knowledge to the agent. In the case of the lane keeping scenario on a highway, one thing to consider is that strong steering actions are seldomly needed. Strong actions should only be needed as countermeasure when a distracted driver turns strongly into the wrong direction. However, they are needed in these situations, however scarce they are. The idea is to make minor actions more likely to be chosen during rollouts and give them

an initial value. Thereby, preferred actions are selected and tried out first by the UCB1 algorithm. If they lead to good results, the agent can exploit on them quicker, if not, the agent will also evaluate less preferred actions. The initial value for all actions is set to $v_{init}(a) = 0.9 + 0.1 * Pr(a)$. The action selection probabilities during the rollout phase and the action's initial values are shown in Table 3.3. The initial count is set to zero for all actions.

Action	±2	±1	±0.75	±0.5	±0.25	±0.15	±0.1	0
Probability (%)	2.5	5	5	5	7.5	10	10	10
Initial value	0.9025	0.905	0.905	0.905	0.9075	0.91	0.91	0.91

Table 3.3: Probability of choosing an action during rollouts if preferred actions are used. The positive and negative values *each* have the same probability.

3.2.5 Generative model

The state transition probabilities T and the Observation probabilities Z are not explicitly known. Instead, the agent uses a generative model to sample the tranisitions. For the task at hand, the agent combines TORCS and the driver model to form a generative model. The agent does not know about the real state of the driver model, nor of TORCS. During planning, the agent can use TORC's simulation engine and an interface to the driving model to simulate tranisitions by performing actions starting from arbitrary belief states. The generative model then returns a next state for both driver and TORCS, a reward (using the reward function from Section 3.1.3), and an Observation. If the belief state is close to the real state, the next state, reward, and observation from the generative model will be close to what they would be if the agent had performed an action in the real environment.

Chapter 4

Experimental setup

In this chapter, the experimens that are performed are introduced. First, an overview of the different evaluated scenarios is given in Section 4.1. Subsequently, in Section 4.2 some important experiment design decisions are explained and justified. Furthermore, the process of tuning the hyperparameters is discussed in Section 4.3. Lastly, the performance metrics for the evaluation are presented in Section 4.4.

All experiments have been performed single threaded, but in parallel, on a Google Cloud C2 Compute-optimized[1] virtual machine with 60 cores and 240 GB RAM, running Ubuntu 20.04 LTS.

4.1 Evaluated scenarios

Three different agent configurations are evaluated, with differing action selection procedures. Moreover, there are three driver models with increasing complexity. In the experiments, each agent is tested with each of the driver models. The aim is to find out how well the different agents can handle the increasing complexity of the driver models.

The three agents that are considered are not fundamentally different. They all use the POMCP algorithm. The difference lies in the way they select actions. The first agent utilizes the full action space. During rollouts, it chooses actions uniformly random. The second agent only considers a subset of the action space containing only minor steering actions (see Table 3.1). For both the first and the second agent, the actions are given an initial value of zero. The third agent uses preferred actions (see Section 3.2.4). It considers the full action space but assigns different probabilities to the actions for their selection during rollouts, preferring minor steering actions over strong ones. Furthermore, it assigns initial values to the action nodes (see Table 3.3).

The driver model configurations are introduced in Section 3.1.2. The basic driver model acts optimally when the driver is attentive and, when distracted, the last action which was performed while the driver was still attentive is repeated. The second model adds complexity by introducing a random amount of steering overcorrection when a formerly distracted driver regains her attentiveness. The intuition is that a driver who suddenly realizes that she has deviated from the lane center is startled and thus steers too strongly to correct the car's position. The third driver model is based on the second one and adds a random amount of action noise on top. This noise is meant to make the driver's behavior more realistic and less predictable.

[1] See Google Cloud machine families

In total, this maks nine scenarios (three agents × three driver models). For each of these scenarios, 13 experiments are executed with varying number of forward searches during planning with the POMCP algorithm (see Section 3.2.1). The rationale behind this is to determine after how many searches the agents' performance converges. The number of searches has a substantial impact on the planning time the agent needs before deciding on its next action. The fewer searches are needed for a good policy, the better, as this translates to a lower planning time.

4.2 Experiment design decisions

A number of noteworthy experiment design decisions were made that are important to consider. These are highlighted and motivated in the following subsections.

4.2.1 Episode definition and repetitions

Every experiment consists of 100 repetitions (runs) of episodes consisting of 1000 actions each, if no terminal state is reached earlier. The episodes are independent of each other - no data is persisted. Each episode starts with the same car state (centered in the lane, relative yaw angle of zero) and the same driver model state (attentive, same duration until distracted). The accelleration and braking are controlled by an external controller, keeping the speed constant at 80 km/h during the entire episode. An episode can end prematurely if a terminal state is reached. A state is terminal if the car is off track, which is considered to be the case if the center of the car lies within more than 20 cm beside the left or right lane markings.

1000 actions represent 100 seconds of simulated driving time. The simulation is not executed in real time. For the time of the planning of the agent, the simulation *waits* for the agent; the simulation does not proceed further until the agent has decided on an action.

4.2.2 Choice of evaluated number of searches

10	100	200	300	400	500	750	1000	1500	2500	5000	7500	10000

Table 4.1: The 13 different values for the number of searches during planning that are considered in the experiments.

Table 4.1 shows the 13 values that are used in the experiments for the number of searches during planning. The computational complexity of the planning process grows exponentially with the number of searches. Therefore, it was decided to increase the relative distance between the values as the values increase. This prevent a precise determination of the convergence point. To further narrow down the convergence point, additional experiments would be required. In

practice, a maximum planning time is often defined after which planning ends, no matter how many searches have been performed (Silver & Veness, 2010). This is not the case for the experiments in this thesis.

4.2.3 Initial belief

The initial belief of the agent consists of 1000 particles that each consist of a randomly sampled attentive driver model state and a car state that is slighly modified but close to the true car state. This may seem like cheating at firs but providing completely random car states to the agent would be like asking it to search for a needle in a haystack. The state space is just too large. The main objective for the agent is to cope with the unpredictability of and partial information about the driver's attentiveness.

The agent knows that the driver is initially attentive but not for how long. The duration until the next change of attentiveness occurs is randomly sampled but adheres to the usual dynamics of the driver model; the duration may not be larger than the maximum or smaller than the minimum defined in Section 3.1.2.2. The car states for the initial belief are created by slighly modifying the real initial car state. For every of the 1000 potential initial states, the lane centeredness and the angle of the real car state are increased or decreased by a randomly chosen amount of up to five percent in either direction. Thereby, the agent does not have perfect knowledge of the car's current lateral position but knows how far it is on the track and how fast it is going.

4.2.4 Decreasing the steering frequency

By default, TORCS expects drivers to input new steering actions every 0.02 seconds. However, for the experiments, the action frequency for the agent and the driver model was decreased to 0.1 seconds. The simulation progresses during this time (see Section 3.1.1.1). This means that every combined action is repreated for 0.1 seconds. There are two reasons for this: First, this enables the evaluation of the agent's performance with a longer driving time. Especially for large number of searches, the planning time for every action is rather large. Getting data for 100 seconds of driving time with an action frequency of 0.02 would require five times the current computation time. This is not feasible with the limited recourses for this thesis. Second, lowering the action frequency makes the driving task more difficult as it is easier to overshoot when the same action is repeated often. The agent has to plan more carefully. Moreover, the agent has to look further ahead as there are more situations in which the action that leads to the highest immediate reward is not the best to choose to maximize long-term return.

4.2.5 Driver action discretization

The actions of the driver are discrete (see Section 3.1.2.1). Obviously, this is not very realistic. The discretization of the driver's actions decreases the complexity

of the problem sufficiently. It is conceivable that POMCP could be applied even if the driver's actions were continuous. Continuous driver actions, rather than continuous agent actions, do not have the effect of limiting the search tree depth because of a width explosion (see Section 3.2.2). However, they make it a lot harder to predict the driver's actions precisely. None of the agens is able to plan with continuous driver actions.

4.2.6 Controling randomness to ensure comparability

Every run has a different seed for the random number generator used for all randomness in the driver model. Every run comes with different driver behavior. However, all experiments share the same seeds for the runs. For example, this means that during run one of every experiment, the driver model state changes are at the same times and lead to the same durations for attentive and distracted periods. The driver actions can differ though since they are dependent on the car's state, which is also influenced by the agent's actions. The reason for this setup is twofold: First, varying driver behavior for different runs verifies the robustness of the agents to varying situations. Second, sharing the seeds between different experiments ensures the comparability between experiments. The driver model used for the generative model of the agent and the driver model representing the actual driver do *not* share the same seed.

4.3 Hyperparameter optimization

Besides the number of searches, there are two additional hyperparameters of POMCP that have a strong influence on the planning performance: The search horizon and the exploration constant. The exploration constant is used to enhance the value of rarely-tried actions in order to facilitate exploration and stop the agent from overfitting. The search horizon (or discount horizon) describes how many actions the agent looks into the future during planning, if no terminal state is reached earlier.

Six different exploration constants and three different search horizons are evaluated in a grid search for the best combination. Each combination is evaluated by the average cumulative return from an experiment with 20 runs, up to 1000 actions each, and 1000 searches per planning step. The second driver model is used for all experimens (steering overcorrection, no action noise; see Section 3.1.2.1). The grid search is only performed for this selected scenario because of limited computing and time resources. However, the experiments should be sufficient to indicate the performance differences stemming from the hyperparameter combinations.

4.4 Performance metrics

The performance of the agents is measured in two ways: First, there is the cumulative reward an agent receives in an experiment for a particular scenario.

Second, the terminal states an agent encounters during the experiment is considered. Terminal states entail that the lane keeping task was failed. It is to be expected that all agents will lead to similar cumulative rewards with an increasing amount of searches during planning. The key is the number of searches they require in order to achieve this reward. A lower number of searches means more efficient, faster planning. Furthermore, the optimality of the agents' policies when converged will be evaluated by comparing the agents' performances with two baselines. First, as lower bound, the driver model driving without assistance. Second, as upper bound, an agent with perfect knowldge of the state who always acts optimally.

Chapter 5

Results

This chapter presents the results from the experiments. First, in Section 5.1 the performance of the driver without assistance, and of an agent that acts optimally, are established as lower and upper performance bounds respectively. In Section 5.2, the results of the hyperparameter optimization are presented. Then, the influence on the performance of the number of searches during planning is evaluated in Section 5.3, with the aim of identifying the point of convergence for each agent. The cumulative reward but also the terminal states reached during experiments are taken into account. The results are grouped according to the different versions of the driver model. Lastly, in Section 5.4 the progression of the mean lane centeredness is shown for an increasing number of searches.

5.1 Lower and upper performance bound

The benchmark for the performance of the agents is the performance of the driver without assistance system as lower bound, and the performance of an agent that always reacts optimally to the driver's actions as upper bound. For both baselines, 50 runs with up to 1000 actions each, if no terminal state is reached earlier, are performed for each of the three driver models.

In the case of the independent driver, no run was completed successfully. At some point in any run, the driver becomes distracted and fails to adjust to a change of the course of the road, leading to lane departure. Table 5.1 shows that the more complex the driver model is, the worse is its performance. The simple driver model, and the driver model with steering overcorrection lead to a few runs with a relatively high number of successful actions before a lane departure occurs. After becoming distracted, the driver only repeats it's last attentive action. In some cases this means that the distracted driver just continues to steer straight which is less likely to lead to lane departure on the highway track than consecutively steering left or right.

The agent reacting optimally to the driver's actions has full knowledge about the car state, as well as the driver's next action. It always chooses the action that leads to the best possible combined action. It finishes every run successfully and leads to average cumulative rewards of 999.3 for all driver models.

	Mean reward	Min #actions	Max #actions
Simple driver	54.39	17	347
Over correction	51.26	17	233
Over correction and noise	31.08	14	74

Table 5.1: Independent driver performance

5.2 Hyperparameter optimization

Hyperparameter optimization is performed for agents with all three action configurations and the driver model with steering overcorrection but without noise. Grid search is used to search for the combination of search horizon and exploration constant that leads to the highest average cumulative reward after 20 runs, with up to 1000 actions each, and 1000 searches per planning step.

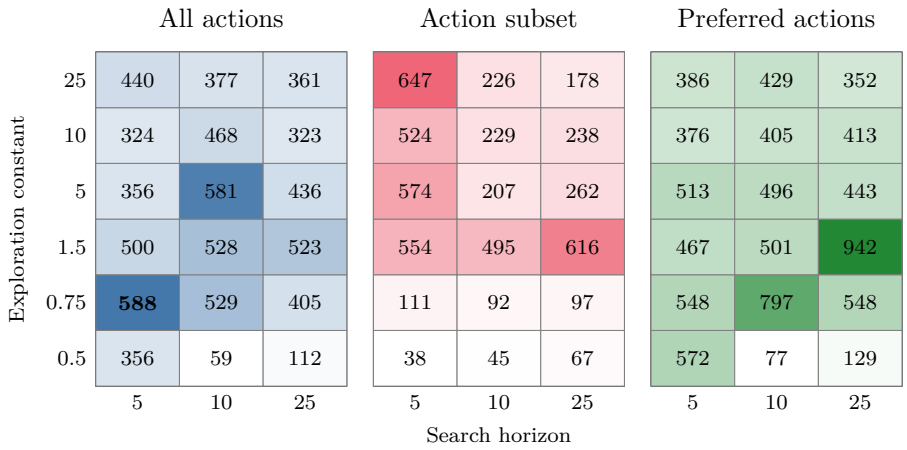

Figure 5.1: Average cumulative rewards for combinations of search horizon and exploration constant for all three action configurations (20 runs with up to 1000 actions each and 1000 searches per planning step; driver model with steering overcorrection but without noise).

For the agent with a full, unweighted action space, two combinations lead to a sufficiently higher average cumulative reward than the others: The combination of a search horizon of 5 actions with an exploration constant of 0.75 leads to a reward of 587.67, and the combination of planning 10 actions ahead with an exploration constant of 5 results in a reward of 581.40. The shallower the search horizon, the less planning time is needed at every planning step as fewer actions need to be simulated. Thus, at the same performance, a lower search horizon is preferable. The combination of a search horizon of 5 actions and an exploration constant of 0.75 is used for the further experiments.

In the case of the agent restricted to using a subset of the action space, the combination of a search horizon of 5 actions and an exploration constant of 25 yields the highest average cumulative reward of 646.83. Only the combination of a search horizon of 25 actions with an exploration constant of 1.5 comes relatively close with a reward of 616.47. As a lower search horizon is preferable, the setup for the further evaluation for this agent is a search horizon of 5 actions with an exploration constant of 25.

The best combination of search horizon and exploration constant for the agent with preferred actions is 25 actions and 1.5 respectively. This combination

yields an average cumulative reward of 942.05 which is sufficiently higher than the return of any other combination. Consequently, this is the combination used for this agent in subsequent experiments.

5.3 Convergence behavior

The agent's challenge is threefold: First, it must accurately determine where the car is located on the track. Its observations are accurate, however, because of the discretization, multiple actual positions map to the same observation (See Section 3.1.1.3). Second, it needs to estimate whether the driver is attentive or not. Third, it must decide on an appropriate action choice, taking into account future consequences of the chosen action. In order to achieve this, at each planning step, the agent can search ahead by simulating experiences starting from a sampled belief state. Performing more searches means evaluating more possible scenarios. In turn, more evaluated scenarios enable the agent to form a better policy. However, after some amount of searches, the information gain from additional searches decreases and performance is expected to converge (Silver & Veness, 2010). The number of searches is directly related to the planning time. Planning time is a limiting factor for the application of a planner. Thus, knowing after how many searches the performance converges, and therefore being able to choose the minimal number of searches to perform for a good result, is critical. Below, the convergence behavior of the three evaluated agents is assessed for the scenarios of the simple driver model, the driver model with steering overcorrection after regaining attentiveness, and the driver with overcorrection and noise.

5.3.1 Simple driver model

For the experiment with the simple driver model, using the full action space, utilizing only the subset of minor steering actions, and employing preferred actions lead to similar convergence behavior. Already with just 200 searches during planning, the average cumulative reward is above 600 for all agents. However, as it can be seen in Table 5.2, the number of runs that lead to a terminal state is high. In the runs resulting in a terminal state, the agents are able to assist the drivers well at first but suffer from particle deprivation after some time (see Section 3.2.3). Then, their belief deviates noticably from the true states of the environment and the driver. The agents are not able anymore to make accurate assumptions about the state of the environment and whether the driver is attentive or not. Thus, they are rendered unable to decide on the right actions to keep the car centered.

The agent using the full range of actions already causes only four terminal states with 300 searches, whereas the other two agents need more searches to perform well. The best result for all three agents is achieved with 1500 searches. No run results in a terminal state. The agent with the full action space receives an average cumulative reward of 957.83, the agent with a reduced action space yields 981.99, and the agent using preferred actions gains 973.88.

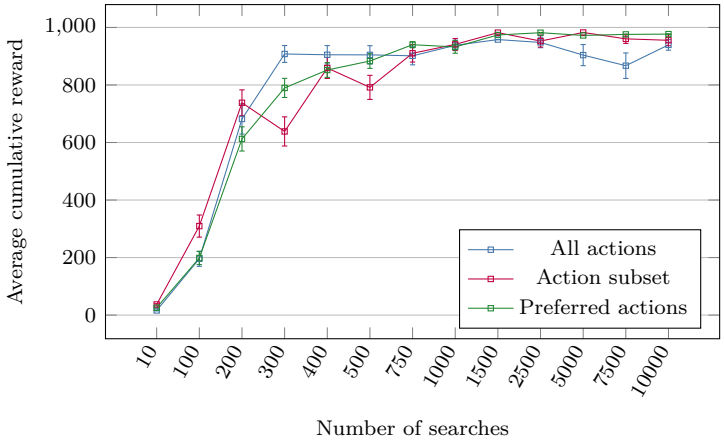

Figure 5.2: Performance comparison of POMCP when utilizing all actions, an action subset, or preferred actions with a simple driver model. Each point shows the mean cumulative reward from 50 runs with 1000 actions each, if no terminal state is reached earlier.

For more searches, the average cumulative rewards are similar for the agents with a subset of actions and preferred actions. They seem to have converged. In contrast, the agent with a complete action space encounters four terminal states in the trial with 5000 searches and six in the experiment with 7500 seaches. These terminal states are reached early on during the run - executing less than 50 actions before. They are caused by an extreme action of the agent that leads to the opposite of optimal steering behavior. The actions completely overrule the driver's actions who is even concentrated in three of the ten occasions. The reason for choosing these extreme actions is a poor belief that strongly deviates from the true states of the environment and the driver. The agent with preferred actions is less likely to perform extreme actions, and the agent with a subset of minor actions cannot do so.

The agent restricted to use only a subset of minor actions reaches terminal states in two states in the experiments with 2500, 7500, and 10000 searches. These are not caused by particle deprivation. In all cases, the car is in a road bend and the driver is distracted, steering into the wrong direction. The agent performs its best possible action by steering as much as possible in the opposite direction than the driver. However, because the agent's range of actions is severely limited, the combination of the agent's and driver's actions is not enough to keep the car in the lane.

Using preferred actions results in only one terminal state for experiments with 1500 searches or more. The reason, like for the terminal states reached by the agent without weighted actions, is particle deprivation. There are multiple occasions where a distracted driver steers into the wrong direction in a road bend and the agent is able to correct the steering by effectively reversing the

driver's actions.

	10	100	200	300	400	500	750	1000	1500	2500	5000	7500	10000
All	50	50	22	4	4	3	3	1	0	1	4	6	1
Subset	50	48	27	26	11	18	6	5	0	2	0	2	2
Preferred	50	50	32	13	8	4	1	3	0	0	1	0	0

Table 5.2: Number of terminal runs by the number of performed searches at each planning step in the experiment with a simple driver model for agents with all three action space types.

5.3.2 Steering overcorrection

A driver that overcorrects after regaining attentiveness by steering too strongly in her first attentive action presents a greater challenge for the agents. When choosing its next action, an agent also need to account for the driver's possible overcorrection. The amount of overcorrection is stochastic (see Section 3.1.2). Attentive drivers are less predictable when they overcorrect. Rather than just performing the optimal steering action, different overcorrection intensities can lead to a variety of actions at the same position. The complexity of the planning problem is higher. Generally, more searches are needed in order to evaluate a greater variety of possible states from the belief while planning.

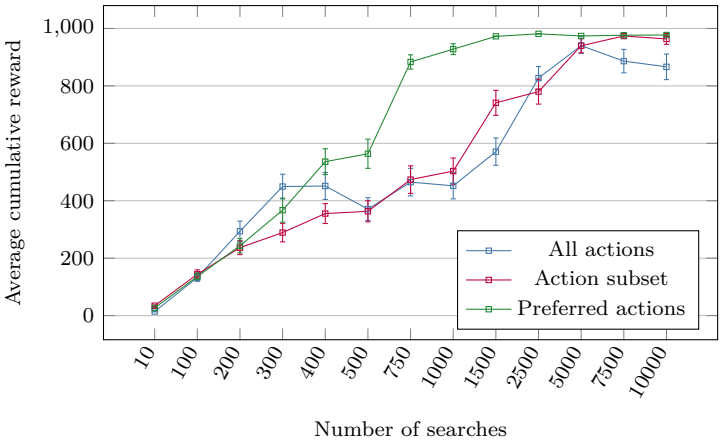

Figure 5.3: Performance comparison of POMCP when utilizing all actions, an action subset, or preferred actions with a driver model that over-corrects when it regains attention. Each point shows the mean cumulative reward from 50 runs with 1000 actions each, if no terminal state is reached earlier.

As it can be seen in Figure 5.3, the agent using preferred actions converges sufficiently earlier towards a good policy than the other two agents. This is mainly caused by a high number of terminal states reached during evaluation runs of the

	10	100	200	300	400	500	750	1000	1500	2500	5000	7500	10000
All	50	50	49	45	42	46	42	41	35	13	4	5	6
Subset	50	50	49	49	48	49	42	39	23	17	3	1	1
Preferred	50	50	49	42	35	29	10	3	0	0	0	0	0

Table 5.3: Number of terminal runs by the number of performed searches at each planning step in the experiment with a driver model with steering over correction for agents with all three action space types.

agents with full and reduced action spaces (see Table 5.3). For example, with 750 searches, the agent using preferred actions receives an average cumulative reward of 883.43 with only 10 runs ending in a terminal state, while the other agents each reach terminal states in 42 runs. The agent with an unrestricted action space yields a return of 464.89, and the one with a restricted action space gains 473.67. In contrast to the experiment with the simple driver model, the terminal states are caused almost exclusively by particle deprivation after a formerly distracted driver becomes attentive again and overcorrects. If the agent did not account for this possibility properly, no matching node for the observation resulting from the overcorrection can be found in the agent's planning tree. In this case, the agent cannot recover and continues using uniformly random action selection (see Section 3.2.1), which usually leads to a terminal state after just a few actions. The agent with preferred actions is able to form an accurate belief of the environment's true state using fewer searches than the other two agents.

5.3.3 Steering overcorrection and noise

The noise added to the driver's actions is added with the goal to make the driving more realistic, and thereby also more difficult to plan with. However, the performance of the agents in the experiment with over correction and noise is very similar to the experiment with overcorrection but without noise. Surprisingly, all agents even receive slightly higher average cumulative rewards and show a similar convergence behavior. The agent using preferred actions converges to a reward of roughly 960 with 1000 searches or more. The agent with a full action range reaches peak performance at 5000 searches with a return of about 860, staying at roughly the same level subsequently. The agent with the small action space converges at around 960, with 7500 searches and more. The actual convergence probably occurs with somewhere between 5000 and 7500 searches but no experiment was conducted with a number of searches in between.

Table 5.4 shows that the number of terminal states reached during experiment runs are comparable for the two agents with unweighted actions. For the agent using preferred actions, the number of terminal states reached at 750 searches is twice as high. In most of these cases, the cause for this is a combination of a strong overcorrection with a high driver action noise. This combination is unlikely but possible with the random nature of the process. Despite this deviation, the data is very similar to the experiment without noise.

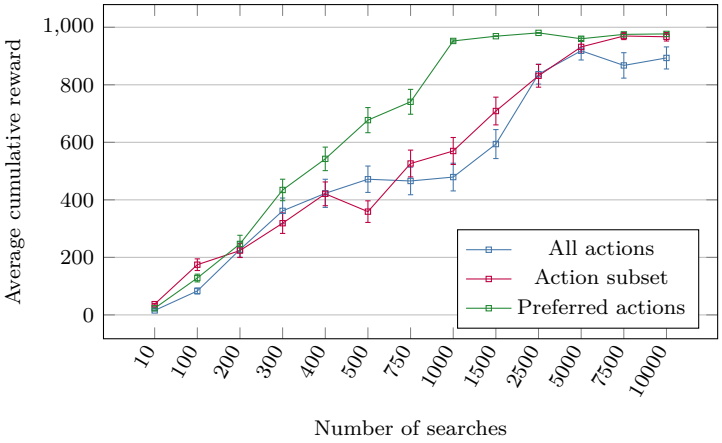

Figure 5.4: Performance comparison of POMCP when utilizing all actions, an action subset, or preferred actions with a driver model that over-corrects when it regains attention and performs noisy actions. Each point shows the mean cumulative reward from 50 runs with 1000 actions each, if no terminal state is reached earlier.

	10	100	200	300	400	500	750	1000	1500	2500	5000	7500	10000
All	50	50	50	44	42	40	40	40	30	15	4	6	5
Subset	50	50	49	47	46	47	37	35	25	14	5	1	1
Preferred	50	50	48	43	37	26	20	1	0	0	3	0	0

Table 5.4: Number of terminal runs by the number of performed searches at each planning step in the experiment with a driver model with steering over correction and noise for agents with all three action space types.

5.4 Mean lane centeredness

The reward reflects how well the agents manage to keep the car centered in lane and angled to the road trajectory. From the last section, it is clear that the number of simulations has a strong impact on the agents' performance. However, the last section only showcased this based on the average cumulative rewards. Figure 5.5 shows the average absolute lane centeredness (distance to the lane center, no matter if left or right of lane center) at every time step in the last experiment with driver action noise and a driver that oversteers after regaining attentiveness. The runs ending in terminal states are taken into account until the terminal state occurs. For the remaining actions, they are ignored in the graphs. It is therefore important to consider them (see Table 5.4).

With just 200 searches, most runs end in terminal states. What is striking is that the average lane centeredness is quite volatile and often drifts off into the extreme. Neither of the three agents is consistently capable of keeping the car

centered in the lane. The graph for 500 searches already suggests an improvement. There are less extreme values and the standard errors are lower. The agents with unweighted actions appear to perform better than the one with preferred actions at first glance. However, still almost all runs of the two agents with unweighted actions, and only about half of the runs of the agent with preferred actions lead to terminal states. The performance of the agent with preferred actions is arguably better. Using 1000 searches marks the start of convergence for the agent with preferred actions. The average lane centeredness is consistently lower than with 500 searches throughout the experiment. The agent using a reduced set of actions performs better than with 500 searches, leading to less variation in the lane centeredness and fewer terminal states reached during the experiment. The lane centering of the agent with the full action range is virtually unchanged. When 10000 searches are performed during planning, all agents have converged to their peak performance.The two agents using unweighted actions agents achieve similar results in this case. The agent with preferred actions slightly outperforms the others but appears to have a higher variance in its lane centeredness.

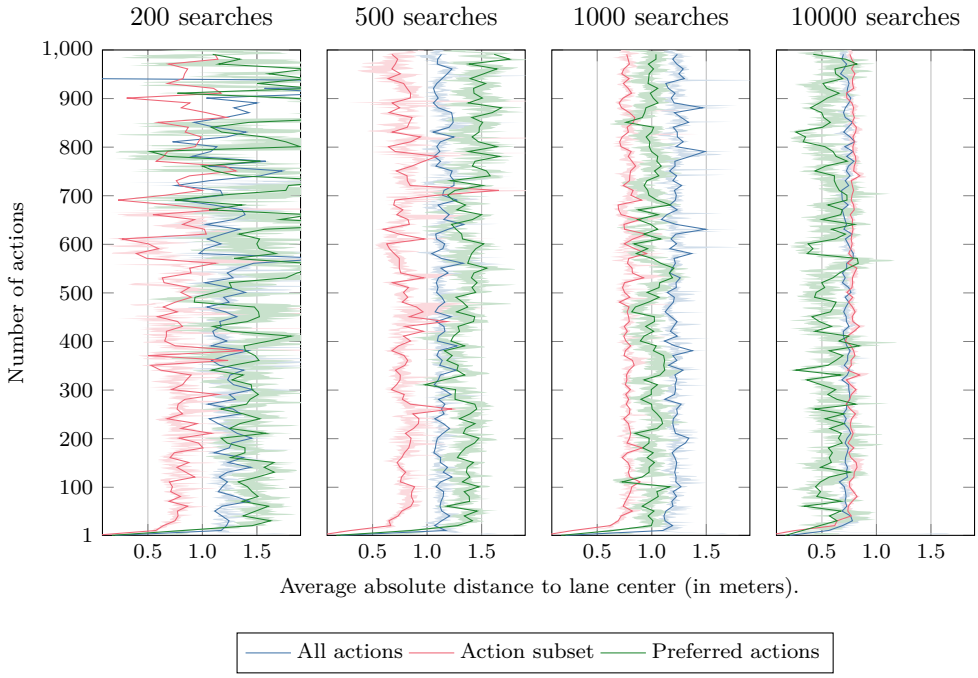

Figure 5.5: Mean lane centeredness for the different agents during experiments with 1000 actions, with oversteering and driver action noise, using 200, 500, 1000, or 10000 searches while planning. In principle, a lower distance is favorable. Nevetheless, it can be better to deviate from the lane center at times in order to reach an overall better performance. The shaded area shows the standard error. *Note: This figure is intended to showcase the difference in lane centeredness between experiments with different numbers of searches for a particular agent. It is not suited to compare the agents with each other without taking into account the different number of terminal runs during the experiments (see Table 5.4). The two agents without preferred actions appear to have lower mean distances than the agent using preferred actions in the first graphs. However, many of their runs end in terminal states. After the terminal state is reached, these runs do not produce additional data and are therefore not reflected anymore in these graphs.*

Chapter 6

Discussion

In this chapter, the results that were presented in the previous chapter are analyzed. The reasons for the performance differences but also the similarities of the agents are explained in Section 6.1. Furthermore, Section 6.2 elaborates on limitations of the chosen solution approach.

6.1 Analysis of the results

6.1.1 Hyperparameter optimization

The results for the hyperparameter optimization are particularly interesting for two reasons: First, the extended experiments are based on the resulting hyperparameters that are deemed best. Second, the results are quite different for the three agents.

A combination of a relatively low exploration constant and a shallow search horizon leads to the best results for the agent with the full unweighted action set. This combination has the effect that the agent constructs a search tree with a maximum depth of only five actions, while it explores relatively little. The agent with the reduced action space yields the best result with a combination of a shallow search horizon but a large exploration constant. During planning, it also only considers five potential future actions but is inclined to explore considerably more. The agent using the full action space starts to exploit early during planning on actions that have shown to be rewarding initially. In contrast, the agent with a reduced action space considers actions more often before starting to exploit on the ones which show better returns. The difference in the agents' behaviors makes sense considering the different number of actions they can choose from during planning. The agent utilizing the full action space has to be more selective. Otherwise, it would *waste* a large amount of its searches to explore the many potential actions and would not be able to generate a meaningful belief at any node.

For the agent using preferred actions, The best results are achieved with a deep search horizon and a low exploration constant. The agent plans ahead 25 actions into the future, with relatively little exploration. The increased search horizon has the effect that the planning complexity, and thereby the planning time, is significantly greater than for the other two agents. It is very important to take this into account. The potential applicability to driving with a real human is highly dependent on the planning time.

6.1.2 Convergence behavior

All agents can lead to good policies if they perform a high number of searches. For the simple driver model, convergence occurs significantly earlier for all agents than with the less predictable driver models. For the simple driver model, all three agents perform quite similar. The agent using the full unweighted action space converges first but the other two agents reach slightly higher results later on. From 1500 searches onward, for the agents using the full action space, both weighted and unweighted, the terminal states reached during runs are caused by particle divergence. The agent using the reduced action set encounters terminal states because it cannot counteract a distracted driver's wrong actions, precisely because its action space is reduced.

For the more complex driver model implementing steering overcorrection, using preferred actions leads to a substantially earlier convergence and to fewer terminal states. The performance of the agent using preferred actions converges somewhere around 1500 searches. From this point on, no terminal states are reached anymore. The other two agents still lead to terminal states during some runs for all evaluated number of searches. The return for episodes where no terminal state is reached is similar to the simple driver model experiments. The agents driving policies are therefore not worse. What is worse is their ability to estimate the current state of the driver. The terminal states result almost exclusively from particle deprivation after a formerly distracted driver becomes attentive again and overcorrects. This was expected since the complexity of the planning task increases with the introduction of oversteering. Only the agent using preferred actions is able to avoid particle deprivation.

The experiments for the third driver model with steering overcorrection and additional driver action noise, lead to similar resuls as the experiments without noise. The most likely reason for this is the discretization applied to the driver's actions (see Section 3.2.2). Low noise that is added to an action is often lost during discretization. The resulting action can therefore be the same action as it would have been before adding the noise. However, there are cases where a combination of a strong overcorrection with a high driver action noise leads to unexpected situations and therefore particle deprivation, even for the agent with preferred actions.

Main problem: Particle deprivation The main cause for suboptimal behavior and reaching terminal states is particle deprivation. The more complex the problem, the more searches are needed to guarantee that a wide array of possible future scenarios are covered. As long as their belief represents the environment's and driver's states well, all agents lead to a good lane keeping performance. However, even with a large number of searches, the agents without preferred actions reach terminal states in some runs, independent of which driver model is used. The only agent that consistently avoids reaching terminal states in all experiments with more than 5000 searches is the agent using preferred actions.

Particle deprivation occurs when an agent's belief strongly diverges from the true state. At some point, none of the observations it receives during planning

will match the real observation anymore. The agent has lost track of the true state completely. For the node representing the real observation, the belief tree stores no particles. From this point on, the agent is effectively clueless about the true state and continues to use random action selection.

Estimating the car's state is not problematic As the car states in the initial belief are very similar to the true car state, and the simulation engine is deterministic (same state and action always lead to the same next state), the car states in the belief do not diverge significantly from the true state during the experiments. The problem lies in the estimation of the state of the driver model. If the agent wrongfully beliefs that the driver is attentive or distracted and the real observations match the observations from planning for some time, the agent's belief will converge to the wrong state. Particle injection (see Section 3.2.3 is implemented to circumvent this to some degree. However, the method is no guaranteed cure for particle deprivation.

Using preferred actions lowers the chance of particle deprivation. Due to the use of preferred actions, the agent does not start with equal initial values at new nodes during rollouts. Instead, the actions are weighted with domain knowledge (see Section 3.2.4). The likelihood of selecting an action for a rollout is bound to its intensity, with less severe steering actions being preferred as the need for strong steering is scarce on a highway track. Assuming the underlying assumption of the introduced domain knowledge is valid, and the return is confirmed to be higher for a preferred action during initial searches, then exploration is kept to a minimum. If the reward does not drop sufficiently, the agent is allowed to exploit on the preferred action. It is like lowering the threshold of trustworthyness for preferred actions; they need less initial confirmation than others. Thereby, a preferred action, if successful initially, is evaluated relatively often, even with fewer searches. Consequently, nodes connected with the preferred actions hold a more comprehensive belief. More driver model states will be considered and therefore particle deprivation is prevented.

Intuitively, one would expect the agent with the reduced action set to lead to a similar improvement when it comes to particle deprivation as the agent using preferred actions. The reduced number of actions means that the fixed number of searches is distributed over fewer actions. Thereby, more potential driver models could be covered per action, which should reduce the chance of particle deprivation. However, during the experiments the agent with the reduced action set did not lead to significantly lower terminal states reached due to particle deprivation than the agent using the full action set. A plausible explanation for this could be the large exploration constant for the agent with the reduced action set. The high value for the exploration constant leads to intensive exploration. Thereby it could counteract the effect of the reduced action space. An additional test with a smaller exploration constant would be necessary to verify this assumption. However, this test has not been performed as part of this thesis.

6.1.3 Mean lane centeredness

Increasing the number of searches during plnning leads to better lane centering for all agents. This was to be expected as the ability to judge the driver's state correctly increases with the number of searches. Moreover, the higher the number of searches, the lower seems t be the variation. With 10000 searches, the car can mosly be kept between 0.5 and 1.0 meters from the lane center, which is a promising result. The agent with preferred actions performs better than the others, but exhibits a higher variance in its lane centeredness at 10000 searches. A possible explanation could be that it accounts better for the times it has to purposefully deviate from the center of the lane, in order to achieve better values subsequently. However, further experiments would be necessary to verify this assumption.

6.2 Limitations

There are multiple factors that limit the applicability of the solution approach taken in this thesis to a more realistic setting or even a real world scenario. The most striking limitations of the solution approach are elaborated in the following subsections.

6.2.1 Long planning time

One of the most striking shortcoming of the online solution approach taken in this thesis is the long planning time the agents require for the forward search they perform during planning. Anything beyond a few fractions of a second would be too much in a real world driving scenario. The agents that are analyzed in this thesis require many searches during planning in order to avoid particle deprivation. Only the agent using preferred action was able to consistently avoid terminal states for all driver models, and that only with 7500 searches or more. The other agents may require even more forward searches. Especially for the agent with preferred actions, due to the deep search horizon, performing a large number of searches means having to plan for a long time. Performing 7500 searches with a search horizon of 25 means the agent has to simulate a total of 187500 actions.

It may be possible to speed up the computation and therefore reduce the amount of planning time that is needed. There are two main ways in which improvements can be made. First, the generative model can be made more efficient. Second, the POMCP algorithm could be parallelized or potentially even replaced by a more efficient algorithm. Both options are discussed in Section 7.2.2.

6.2.2 Dependency on a reliable generative model

For the experiments conducted in this thesis, the same driving simulator and driver model that were used as a simulation of a real driving scenario, where

also used as the generative model for th agent. The transition and observation probabilities underlying the generative model were therefore an exact replication of the dynamics of the agent's environment (car and driver). POMCP is based on the assumption that such a generative model exists which can sample the true transition and observation probabilities. However, in a real-life scenario, or even just while using a more realistic driver model, such a generative model might not be available. It might be possible to use a generative model that just approximates the dynamics of the environment. This has not been evaluated and likely leads to problems if the approximation is not very accurate.

6.2.3 Action and observation space discretization

The action and observation space for the agents are discrete. POMCP is not suited to be used with continuous action and observation spaces (see Section 3.2.2). Steering actions and a car's sensory information are naturally continuous. The results indicate that a discretization of the observation space can be successful. However, the more limited the number of bins used for the discretization, the lower is the precision. On the one hand, increasing the resolution of the discretization can lead to a considerable elevation of the planning complexity. More distinct observations may be encountered while constructing the search tree. A low precision of the observations, on the other hand, does not allow the agent to act precisely, leading to suboptimal behavior. Limited steering precision is also caused by discretizing the action space. If the agent can only choose from a limited number of actions, it cannot steer accurately. Its assistance might be jumpy. This does not allow for a smooth driving experience. Section 7.2.4 discusses briefly how this obstacle could be overcome.

Furthermore, the actions of the driver are discretized. In any realistic scenario, this just could not be the case as it reduces the driver's precision which is obviously unacceptable. The state space is continuous. Therefore, theoretically, POMCP can work with continuous driver actions, as long as they are discretized for the observations. However, none of the agents is remotely successful if continuous driver actions are used. It becomes impossible to account for all potential driver actions while planning. A wide array of sampled actions would probably be enough to allow for a good estimation. However, a low discretization resolution for the observations leads to a big range of different driver actions that are represented by one observation. Planning becomes too imprecise. Using a continuous observation space could therefore potentially enable the agent to plan with continuous driver actions as well. Further experiments would be necessary to verify this assumption.

6.2.4 Driver does not learn or adapt

Research suggests that drivers adapt their driving behavior when they are assisted by a lane keeping assistant (Miller & Boyle, 2017). The driver model used in the experiments does not evolve over time. It does not adapt to the behavior of the agent. Even if the agent would steer into the wrong direction multiple times in a

row, the driver model does not adapt to the agent by adjusting its behavior and steering stronger to acocunt for the agent's wrong actions. The driver also does not develop any reliance on the agent - it does not reduce its steering efforts, trusting the agent to correct it, like it would be conceivable with real humans.

Chapter 7

Conclusion and future outlook

Appendices

Bibliography

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, vol. 47no. 2, 235–256. https://doi.org/10.1023/A:1013689704352

Bai, H., Hsu, D., & Lee, W. S. (2014). Integrated perception and planning in the continuous space: A POMDP approach. *Int. J. Rob. Res.*, vol. 33no. 9, 1288–1302. https://doi.org/10.1177/0278364914528255

Brechtel, S., Gindele, T., & Dillmann, R. (2013). Solving Continuous POMDPs: Value Iteration with Incremental Learning of an Efficient Space Representation. *ResearchGate*. https://www.researchgate.net/publication/256078956_Solving_Continuous_POMDPs_Value_Iteration_with_Incremental_Learning_of_an_Efficient_Space_Representation

Cai, P., Luo, Y., Hsu, D., & Lee, W. S. (2018). HyP-DESPOT: A Hybrid Parallel Algorithm for Online Planning under Uncertainty. *arXiv*, 1802.06215. https://arxiv.org/abs/1802.06215v1

Coquelin, P.-A., & Munos, R. (2007). Bandit Algorithms for Tree Search. *arXiv*, cs/0703062. https://arxiv.org/abs/cs/0703062v1

Espié, E., Guionneau, C., Wymann, B., Dimitrakakis, C., Coulom, R., & Sumner, A. (2005). Torcs, the open racing car simulator.

Garg, N. P., Hsu, D., & Lee, W. S. (2019). DESPOT-Alpha: Online POMDP Planning with Large State and Observation Spaces. *undefined*. https://www.semanticscholar.org/paper/DESPOT-Alpha%3A-Online-POMDP-Planning-with-Large-and-Garg-Hsu/62a1c3b8468a3416fb3189c1203c713d66ee766d

Goldhoorn, A., Garrell, A., Alquézar, R., & Sanfeliu, A. (2014). Continuous real time pomcp to find-and-follow people by a humanoid service robot, In *2014 ieee-ras international conference on humanoid robots*. https://doi.org/10.1109/HUMANOIDS.2014.7041445

Graves, D., Nguyen, N. M., Hassanzadeh, K., & Jin, J. (2020). Learning predictive representations in autonomous driving to improve deep reinforcement learning. *arXiv*, 2006.15110. https://arxiv.org/abs/2006.15110v1

Hoerger, M., & Kurniawati, H. (2020). An On-Line POMDP Solver for Continuous Observation Spaces. *arXiv*, 2011.02076. https://arxiv.org/abs/2011.02076v1

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, vol. 101no. 1, 99–134. https://doi.org/https://doi.org/10.1016/S0004-3702(98)00023-X

Kass, S., Cole, K., & Stanny, C. (2007). Effects of distraction and experience on situation awareness and simulated driving. *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 10, 321–329. https://doi.org/10.1016/j.trf.2006.12.002

Kővári, B., Hegedüs, F., & Bécsi, T. (2020). Design of a Reinforcement Learning-Based Lane Keeping Planning Agent for Automated Vehicles. *Appl. Sci.*, vol. 10no. 20, 7171. https://doi.org/10.3390/app10207171

Kochenderfer, M., Wheeler, T., & Wray, K. (2021). *Algorithms for decision making* [Unpublished manuscript. Retrieved from https://algorithmsbook.com/]. Unpublished manuscript. Retrieved from https://algorithmsbook.com/.

Luo, Y., Bai, H., Hsu, D., & Lee, W. S. (2018). Importance sampling for online planning under uncertainty. *Int. J. Rob. Res.*, vol. 38no. 2-3, 162–181. https://doi.org/10.1177/0278364918780322

Miller, E. E., & Boyle, L. N. (2017). Driver adaptation to lane keeping assistance systems: Do drivers become less vigilant? *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 61no. 1, 1934–1938. https://doi.org/10.1177/1541931213601963

Papadimitriou, C. H., & Tsitsiklis, J. N. (1987). The complexity of markov decision processes. *Mathematics of Operations Research*, vol. 12no. 3, 441–450. https://EconPapers.repec.org/RePEc:inm:ormoor:v:12:y:1987:i:3:p:441-450

Pineau, J., Gordon, G., & Thrun, S. (2006). Anytime point-based approximations for large pomdps. *Journal of Artificial Intelligence Research*, vol. 27, 335–380. https://doi.org/10.1613/jair.2078

Pineau, J., Gordon, G., & Thrun, S. (2003). Point-based value iteration: An anytime algorithm for pomdps, In *Proceedings of 18th international joint conference on artificial intelligence (ijcai '03)*.

Ross, S., Pineau, J., Paquet, S., & Chaib-draa, B. (2008). Online planning algorithms for pomdps. *Journal of Artificial Intelligence Research*, vol. 32, 663–704. https://doi.org/10.1613/jair.2567

Schoon, C. (1994). Road design standards of medians, shoulders and verges. SWOV Institute for Road Safety Research. https://www.swov.nl/publicatie/road-design-standards-medians-shoulders-and-verges

Shani, G., Pineau, J., & Kaplow, R. (2013). A survey of point-based pomdp solvers. *Autonomous Agents and Multi-Agent Systems*, vol. 27no. 1, 1–51. https://doi.org/10.1007/s10458-012-9200-2

Silver, D., & Veness, J. (2010). Monte-carlo planning in large pomdps (J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, & A. Culotta, Eds.). In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, & A. Culotta (Eds.), *Advances in neural information processing systems*, Curran Associates, Inc. https://proceedings.neurips.cc/paper/2010/file/edfbe1afcf9246bb0d40eb4d8027d90f-Paper.pdf

Smallwood, R. D., & Sondik, E. J. (1973). The optimal control of partially observable markov processes over a finite horizon. *Operations Research*, vol. 21no. 5, 1071–1088. https://doi.org/10.1287/opre.21.5.1071

Smith, T., & Simmons, R. (2004). Heuristic search value iteration for pomdps, In *Proceedings of the 20th conference on uncertainty in artificial intelligence*, Banff, Canada, AUAI Press.

Spaan, M. T., & Vlassis, N. (2005). Perseus: Randomized point-based value iteration for pomdps. *Journal of Artificial Intelligence Research*, vol. 24, 195–220. https://doi.org/10.1613/jair.1659

Sunberg, Z., & Kochenderfer, M. (2018). Online algorithms for pomdps with continuous state, action, and observation spaces.

Ye, N., Somani, A., Hsu, D., & Lee, W. S. (2017). Despot: Online pomdp planning with regularization. *Journal of Artificial Intelligence Research*, vol. 58, 231–266. https://doi.org/10.1613/jair.5328

Young, K. L., Lenné, M. G., & Williamson, A. R. (2011). Sensitivity of the lane change test as a measure of in-vehicle system demand. *Appl. Ergon.*, vol. 42no. 4, 20828672, 611–618. https://doi.org/10.1016/j.apergo.2010.06.020