Utrecht University

Jokke Mats Jansen

Title placeholder

Thesis submitted for the degree of Master of Science

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

1.1 Motivation

Fully autonomously driving cars have the potential to rule out human driving error, which is at least a contributing factor to most accidents today. However, many social and technical obstacles have yet to be overcome until fully autonomous cars become market-ready (**autonomous_driving_book**). Cars will likely not be able to drive fully autonomously in the near future, continuing to require some degree of human supervision and intervention (**human-needed**, and **human-needed-2**). Until fully autonomous cars are available, increasinlgy autonomous driving assistance systems will be employed transitionally (**autonomy-future**). Only specific tasks, such as lane-keeping or cruise control, are delegated to an assistance system, while the driver retains the responsibility for other tasks.

However, delegating tasks to an assistance system could limit the driver's autonomy. A loss of autonomy can turn driving into a monotonous and tedious supervisory task. As a result, drivers may distract themselves more frequently (**driver-distraction**), have reduced reaction times (**reaction-time**), suffer from a lower situational awareness (**situational-awareness**), and can place too much trust in the assistance system (**over-trust**). Moreover, in the case of conditional automation, the time required for the driver to shift back to driving from a non-driving task may be too long in critical situations (**takeover-time**). To mitigate these issues, **shared-control-haptics** propose employing a shared control scheme: The driver and the assistance system continuously share the control authority of the car, cooperating in the driving task. The driver is kept in the loop. Thereby, the driving task stays enjoyable, and drivers are prevented from relying too heavily on the assistance systems.

A large part of driving accidents is associated with driver distraction (**distracted_nhtsa**). Therefore, it seems reasonable to make the extend of the assistance dependent on the driver's level of attention. Whenever a driver is inattentive or distracted, an assistance system needs to be particularly sensitive. **disracted-lane-keeping-1** and **disracted-lane-keeping-2** show that increasing the intensity of lane-keeping assistance when the driver is distracted increases lane-keeping performance, while the driver is not influenced in her driving while being attentive.

However, the detection of driver distraction is complex. Methods like eye movement tracking (**eye-movement**), head tracking (**head-tracking**), and driver gaze monitoring (**gaze-monitoring**) have been proposed and proven to be successful. However, they all rely on continuous video recordings and analysis of the driver. This constitutes a privacy intrusion and may not be desirable or

impractical (for example, in low-light situations).

This thesis presents an alternative approach: A Partially observable Markov decision process (POMDP) is used to model a shared-control lane-keeping scenario. A driver is assisted by a lane-keeping assistance system that estimates the driver's attentiveness online, based on the driver's past behavior. Explicitly measuring the driver's distraction is not required.

1.2 Related work and contributions

POMDP is a framework to model sequential decision problems (one decision influences the next) in a partially observable environment (**pomdp-definition**). An agent interacts with the environment. After performing an action, the agent receives observations about the environment and a reward signal. It uses the observations it receives over time to estimate the environment's state. Using its estimate of the true state of the environment, the agent can decide which actions to perform next to maximize the long-term reward.

POMDPs are well suited to model driving problems in which the internal psychological state of a human is important but unknown (such as her intend, goals, drowsiness, or attentiveness). Research mostly focuses on fully autonomous cars assessing the behavior of other traffic participants. By employing a POMDP to model the uncertainty about humans' internal states, improved driving policies were achieved in three, widely studied scenarios: pedestrian avoidance (**despot-crowd**, and **pomdp-pedestrian-avoid-2**), intersection crossing (**pomdp-intersection**, **att_intersec**, **pomdp-intersection-2**, and **pomdp-intersection-3**), and lane changes (**pomdp-lane-changes**, **att_intersec**, **pomdp-lane-changes-2**, **tactical-decision**, and **pomdp_towards_human**).

The aforementioned studies demonstrate that a POMDP is suitable for dealing with the uncertainty related to a person's internal psychological state and that this can be beneficial for making the right decisions when driving. However, the focus lies on estimating the state of external drivers rather than considering the state of the driver in a shared control scenario. **hitl_pomdp** introduce a framework to model shared control driving scenarios with a human in the loop using POMDPs. The benefits of the framework are exemplified by assisting a potentially drowsy driver with lane-keeping: The assistance system is able to reason about the driver's drowsiness effectively and can handle noisy or erroneous observations. A driver model is used to simulate human behavior, and a simple car model is employed to simulate the car's dynamics in the experiments. However, the approach relies on the availability of observations that strongly correlate with the driver's drowsiness (observation whether the driver's eyes are closed or not).

Research suggests that driver distraction can lead to a change in driving performance. Distracted drivers show certain behaviors, such as lane position deviations, infrequent steering wheel movements, and reduced reaction times (**driver-distraction-review**). **dist-det-perf** show that monitoring driving performance measures (steering wheel angle, heading angle, and lateral deviation)

alone can be sufficient to detect driver distraction accurately. Driven by those findings, we propose to estimate the driver's attentiveness in an online manner based on on driver's past driving behaviors.

In this thesis, an assisted lane-keeping scenario with a potentially distracted human in the loop is modeled as a POMDP. The requirement from **hitl_pomdp** of observations that correlate very closely with the driver's psychological state is relaxed. The only observations available to the assistance system are the steering actions of the driver, the car's yaw angle, and its lateral deviation. Moreover, the number of considered discrete steering actions is substantially increased. The driving dynamics are simulated using a realistic driving simulator rather than a simple mathematical model.

1.3 Problem overview and research questions

The problem examined in this thesis is assisted lane keeping with shared control by a potentially distracted human driver and an agent acting as an assistance system. The agent assists the driver in keeping the car centered in its lane. The driver becomes distracted periodically. During distraction episodes, the driver may act suboptimally. The driver's behavior is simulated using a driver model. The driver's attentiveness is unknown to the agent. It has to infer if the driver is distracted from the driver's past steering activity and sensory observations about the car's state. The dynamics of the car are simulated using the driving simulator The Open Racing Car Simulator (TORCS). A POMDP model is employed to account for the uncertainty about the driver's attentiveness.

The following research questions arise:

1. How can a lane-keeping scenario with shared control between a potentially distracted human driver and an agent be modeled using a POMDP?
2. Can the agent estimate the driver's distraction using driver performance measures alone, allowing it to take appropriate actions when the driver becomes distracted?
3. Is the solution approach viable for a real-world scenario, and if not, what limitations are there, and what are potential methods to solve them?

1.4 Outline

The remaining content of the thesis is organized as follows:

?? introduces the basic theoretical POMDP concepts which serve as a foundation for the thesis.

?? presents the methodology, including the definition of the POMDP used to model the shared control lane-keeping task and the chosen solution approach.

?? describes the experiments that we perform to assess the performance of the solution approach.

?? showcases the results from the experiments.

?? analyzes the results and elaborates the limitations of the solution approach.

?? highlights the conclusion, contributions, and challenges for a potential future real-world application.

Chapter 2

Theoretical foundation

This chapter introduces the basic theoretical concepts that serve as a foundation for the problem definition and solution approach of this thesis. The problem we study is assisted lane keeping with shared control by a potentially distracted human driver and an agent. The agent acts as a lane-keeping assistant. The driver's attention and the exact position of the vehicle are unknown to the agent. The task involves sequential decision making, where every prior decision influences the following ones. Section **??** 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 **??** introduces the 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 **??** discusses the key challenges involved in solving POMDP. Section **??** provides an overview of POMDP solvers.

2.1 Sequential decision making

Lane-keeping of a car is a sequential decision-making task. Every steering action 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. Figure **??** illustrates the process. At every time step t, the environment is in a certain state s_t, fully observable by the agent. The agent interacts with the environment by performing an action a_t that determines the next state s_{t+1} 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 does not need to be deterministic but can be probabilistic, accounting for randomness in the environment. After performing an action a_t, leading to state s_{t+1}, the agent receives a numerical reward r_{t+1} (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. If 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.
Assisting a human driver in the lane-keeping task is essentially a sequential

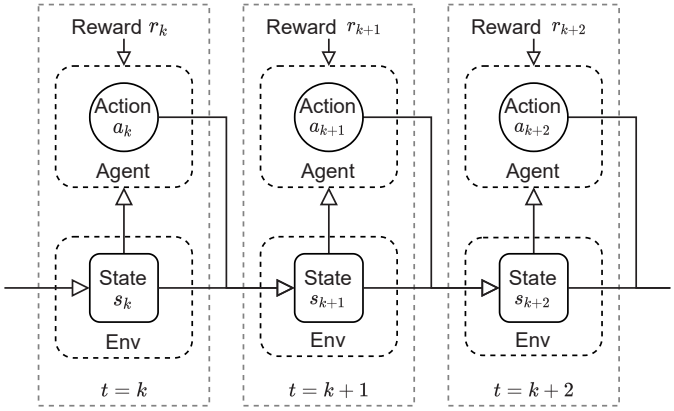

Figure 2.1: Markov decision process (MDP)

decision-making task. However, the agent assisting the human driver does not know about the driver's internal psychological state. This state includes everything that influences the driver's behavior, such as the driver's goals, intentions, emotions, and perceptions, but is not visible from the outside world. We focus on the driver's attention to the driving task when referring to her hidden psychological state hereafter. 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 to act adequately. A Partially observable Markov decision process (POMDP) is a generalization of an MDP that allows planning under uncertainty. Even without observing the complete state of the agent's environment, a POMDP enables the agent to estimate the environment's true state using the partial information it observes. In the next section, we will give a formal definition of a POMDP.

2.2 Partially observable Markov decision process (POMDP)

A POMDP is a generalization of an 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. We follow the description in **pomdp-definition** and 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 a certain 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 to choose adequate actions. At any time step t, it takes 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. The probability of being in state s given history h is denoted as $b(s, h)$.

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

The belief is a sufficient statistic for the agent to form a decision about its next action (**pomdp-belief**). Thus, only the belief needs to be kept and can be recursively updated whenever the agent performs an action and receives a new observation. 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 $\pi : b \rightarrow a$. 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 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 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 (**pomdp_curses**). On the one hand, there is the so-called curse of history: Finding an optimal policy means searching through the space of all 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. Finding an optimal policy can be relatively easy for short histories. However, 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 $|S|$. For a POMDP with a large state space or time horizon, finding an optimal policy is computationally infeasible (**pomdp_complex**). For this reason, approximate algorithms are often applied. We summarize those approximate algorithms in section **??**.

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 **??** is not computable without

knowing the probability distributions explicitly. However, exact updates are too complex for problems with large state spaces in general (**pomcp**).

Some solution approaches circumvent the problem of unknown transition and observation probability distributions by using a generative model to sample state and observation transitions. This generative model can, given the current state and action, stochastically generate a successor state, reward, and observation. Thereby, it implicitly defines the transition and observation probabilities, even if they are not explicitly known. In this thesis, we follow this idea and employ a generative model to solve this challenge in our practical task. The generative model is described in detail in section **??**.

2.4 Algorithms to approximately solve large POMDPs

2.4.1 Offline and online POMDP solving

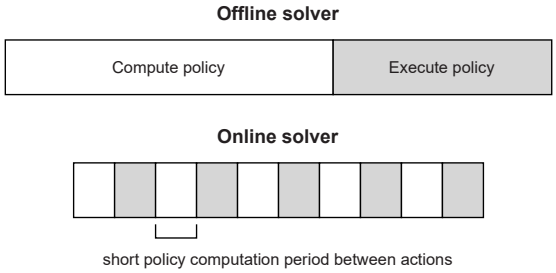

Figure 2.2: Comparison of offline and online solving procedure

There are two general approaches to solve POMDP: offline and online (see figure **??**). Offline solvers compute the optimal policy prior to execution for all possible future scenarios. Their advantage is that once the policy is determined, policy execution is fast as there is only 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 POMDPs can be good, computing the policy may take a very long time. Furthermore, even small changes in the dynamics of the environment require a full recomputation (**online_pomdp**). Online solvers interleave planning and plan execution. At every time step, only the current belief is considered for the computation of 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 for offline planning is required. The amount of available online planning time at each time step limits the performance.

2.4.2 Overview of aproximate POMDP solvers

As discussed in section **??**, finding an exact optimal policy is only feasible for small discrete POMDPs. Larger POMDPs are usually solved approximately. A wide range of offline and online approximate solvers is available. An extensive survey of possible approaches is out of scope for this thesis. A good overview of different methods is given in the chapter "State uncertainty" of the book "Algorithms for Decision Making" (**decision_making_book**). We focus on introducing a selection of especially noteworthy approaches to solving POMDP in this section.

For discrete POMDP, the most effective 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 (**pomdp-point-based-value**). State of the art methods include Perseus (**pomdp_perseus**), and Heuristic Search Value Iteration (HSVI) (**solver_hsvi**). Perseus chooses belief states by randomly sampling trajectories from an initial belief (**pomdp_perseus**). It is sufficiently more compute efficient than PBVI but can suffer from a slow convergence behavior (**pbvi-survey**). HSVI improves on Perseus potential slow convergence in large domains by constructing a belief search tree, maintaining the order of visited beliefs to be backtrack value updates bottom-up (**solver_hsvi**). Even the most advanced offline solvers reach their limit when dealing with POMDPs with large state spaces. As the state space for the shared control lane-keeping task is large, offline solvers are not considered further in this thesis.

When it comes to online solvers, the paradigm of searching for a good policy locally for the current belief state makes them sufficiently more efficient. The general approach is to construct a search tree with the current belief as root, evaluating all possible further actions and observations (**decision_making_book**). This tree can be used to efficiently generate a good approximately optimal action to perform at the current belief. The methods mainly differ in how the tree is constructed. After an 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.

Constructing the search tree using Monte Carlo sampling has recently led to promising results for online solving of large POMDPs (**pomcp**). Including all possibilities in the search tree is not feasible for deep time horizons because of the curse of history and the curse of dimensionality (see section **??**). Monte Carlo tree search (MCTS) methods address this issue by using a generative model to sample state transitions and observations (see section **??**). 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.

2.4.3 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 **pomcp**. The exploration is controlled using the Upper Confidence Bounds 1 (UCB1) (**ucb1**) algorithm for action selection. The belief search tree is constructed by sampling possible action-observation trajectories. The key idea is to approximate the belief space using the same set of states sampled for the search tree construction. POMCP's belief representation eliminates the need for expensive belief update calculations. POMCP has successfully been applied to solve large POMDPs. Thus, the POMCP algorithm was selected as the solver for our problem. In section **??**, we explain how we employed POMCP to solve the shared control lane-keeping task.

The Determinized sparse partially observable tree (DESPOT) algorithm by **despot** 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 (**pomcp-worst-case**) caused by the UCB1 algorithm's tendency to overfit. DESPOT circumvents this by using regularization in the value function. First, a suboptimal policy is searched using heuristic search (**solver_hsvi**) 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 (**hyp-despot**). DESPOT-α further improves DESPOT's capability to handle very large observation and state spaces (**despot-a**). And DESPOT-IS applies importance sampling to account for very rare events that are hard to sample (**despot-is**). 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.

2.4.4 Solving continuous POMDP

The scenario of lane-keeping with a human in the loop that we examine is naturally continuous; steering actions, car state, and sensory information about the car's position are all composed of continuous values (see section **??**). Hence, a solver that can handle continuous POMDP is required.

The aforementioned algorithms POMCP and DESPOT can natively handle continuous state spaces (**pomcp_continuous**). It is unlikely that two identical continuous states are inserted into the collection of states representing the agent's belief. Therefore, the individual relative frequency of a state in the belief, representing its probability in the discrete case, 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 given by the number of belief states close to it.

However, POMCP and DESPOT cannot be applied directly with PODMPs that have continuous action and observation spaces. Discretization is required (**pomcp_continuous**). Action and observation discretization is performed in this thesis to be able to use POMCP. The details are discussed in section **??**.

Various solvers have been developed that are suitable to solve POMDP with continuous action and observation spaces without relying on discretization. To the best of our knowledge, they all require the observation probabilities Z to be known explicitly and are therefore not applicable to the problem addressed in this thesis. Yet, such approaches may be advantageous in future work if the problem of unknown observation probabilities can be circumvented. Two notable solvers are POMCPOW (**online_pomdp_cont**), and LABECOP (**online-cont-pomdp-2**). POMCPOW is an extension of POMCP using progressive widening. It uses weighted belief updates and limits the number of observations considered during planning. LABECOP is based on MCTS as well but avoids limiting the number of considered observations.

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 and how it was applied for the defined task are presented. Section **??** begins with an overview of the three components comprising the problem: The driver model representing the human driver, the driving simulator, and the agent assisting the driver. The simple driver model that we use to simulate human driving behavior is specified in section **??**. Section **??** describes how The Open Racing Car Simulator (TORCS) is used to simulate the dynamics of a car driving on a highway. 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 **??**.

3.1 Overview

The problem addressed in this thesis is an assisted driving lane-keeping task, where a human driver shares control with an agent over a car. The driver and the agent cooperatively control the steering wheel. The goal is to keep the car centered in its lane. Both the agent and the driver have only lateral control; 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. To fulfill the goal of consistently keeping the car centered in the lane, the agent has to effectively estimate 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. It can then plan ahead and select adequate steering actions.

Rather than experimenting with real humans and a real car, simulation models are used for both the car and the human driver in experiments. Figure **??** 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 (**torcs**) 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. From the perspective of the agent, its environment is composed of both the simulated car and the simulated driver. The simulation modules are described in detail in the following subsections.

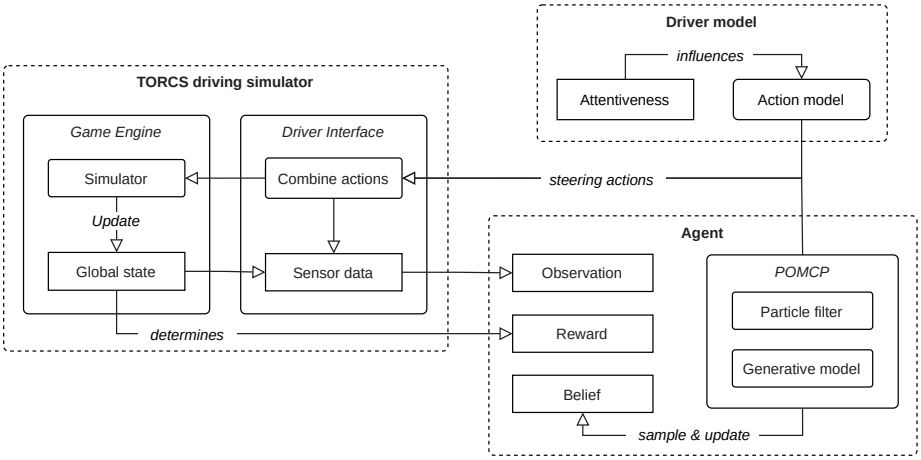

Figure 3.1: Overview of the modules used to represent and solve the shared control lane keeping POMDP

3.1.1 TORCS as a highway driving simulator

The Open Racing Car Simulator (TORCS) is an open-source car driving simulator (**torcs**). 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. It has been widely used by researchers to simulate car driving and to evaluate the performance of autonomous driving agents (for example in **torcs-3**, **torcs-1**, **torcs-2**, and **reward1**).

We use TORCS as a simulator for car dynamics for two purposes. First, to simulate the dynamics of the car the agent and the driver share control over. This simulation is part of the environment of the agent. It represents the car in the real world and would be replaced by an actual car in a realistic setting. TORCS maintains the car's true state and updates it based on the combined steering action. Second, we use the simulation engine as a generative model (see section **??**). The generative model is part of the agent, constituting the agent's model of the world. The agent uses it to sample state and observation transitions for the forward search performed during the online MCTS policy computation (planning).

Typical racing tracks for TORCS have sharp road bends and wide roads with differing widths around the course. This does not reflect a highway driving scenario well. Therefore, a custom highway track is used instead (see figure **??**). The custom track only has moderate road bends and a constant lane width of 3.75m, as it is common in Europe (**lane_width**). The road is completely flat and there are no other cars on it. Moreover, a fixed speed of 80 km/h is set

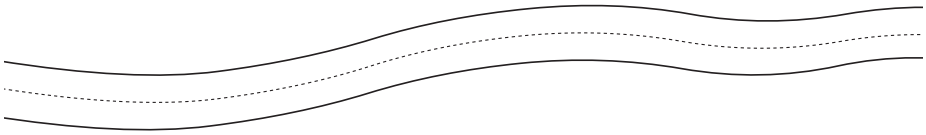

Figure 3.2: Course of the road of a section of the TORCS highway track used for experiments.

during our experiments.

3.1.2 Driver model

In this study, the driver is simulated using a stochastic driver model, instead of performing experiments with an actual human driver. 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. We design a simple driver model to evaluate our approach. 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. 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 reduced situational awareness, and therefore she does not adjust to road changes like an attentive driver would (**driver-awareness**; **driver-awareness2**).

2. **Steering overcorrection:** When a distracted driver diverts strongly from the lane center and then becomes attentive again, the driver suddenly notices the deviation. In this case, drivers tend to perform an overly strong steering correction and overshoot (**oversteering-motivation**). We implemented this behavior in a second, more complex model. The first action of a driver that regains attentiveness is increased by a random amount between 10 and 25 percent. The randomness makes the model less predictable when the driver becomes attentive again.

3. **Steering overcorrection and noise:** The last, most complex scenario introduces action noise. A random noise between five and 20 percent is added to every action the driver takes. The action can thereby become five

to 20 percent stronger or weaker. The overcorrection for the first action after regaining attentiveness, introduced in the last model, is performed as well. The noise is added on top. The noise makes the driver less predictable in every situation.

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

In this section, we formulate the shared control lane-keeping problem as a POMDP. Refer to section **??** for a general definition of a POMDP. In the following subsections, we define the state space and the state transition probabilities, the action space, the reward function, and the observation space with the observation probabilities.

3.2.1 States and transition probabilities

The overall state space S consists of all possible states of the agent's environment. For the agent, the environment is compiled of both the driver and the car. Therefore, the state space we have to consider in the POMDP is the combination of all possible combinations of the state of the car and the state of the driver model. The state transition probabilities T are not explicitly given but implicitly defined by the car dynamics simulated by TORC's simulation engine.

Car state

TORCS data model for the car state is too extensive to list here in full[1]. Table **??** shows the most important attributes. These include the car's position on the track, its current velocity, and its acceleration. Among others, there are additional attributes for the state of transmission and engine, the friction and spin of the wheels, and aerodynamic influences such as drag and lift. The values in the state are continuous.

Long. position	Lat. position	Yaw angle	Velocity	Acceleration
115m (from start)	0 (lane center)	45°	80 km/h (fixed)	0.3

Transmission	Engine	Wheels	Aerodynamics	...
gear, clutch, ...	rpm, pressure, ...	friction, spin, ...	drag, lift, ...	...

Table 3.1: Examplary car state. This is an incomplete table, only listing the most important attributes. For a full state representation, look up the *tCar* struct in the TORCS API documentation.

The lateral position (lane centeredness) and the yaw angle of the car are illustrated in figure **??**. The lane centeredness of the car is defined over a continuous interval of $(-\infty, \infty)$, with values in between -1 (right lane border) and +1 (left lane

[1] See *tCar* struct in the TORCS API documentation

border) denoting that the car is within its lane, and everything beyond standing for an off-track position. A value of zero means the car is centered in its lane. The car's relative yaw angle is given with respect to the track direction and defined over an interval of $[-\pi, +\pi]$ radians. A value of zero indicates that the car is heading in the same position as the track.

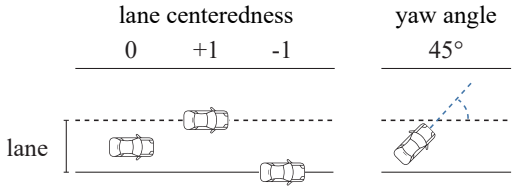

Figure 3.3: Illustration of lane centeredness and yaw angle values

Driver state

The driver state consists of two variables: The current state (attentive or distracted) and the duration it remains in this current state. The duration it remains in a state is randomly chosen but lies between one second (10 actions) and 5 seconds (50 actions). When the time runs out for the current state, the state reverses; an attentive driver becomes distracted, and a distracted driver regains attentiveness. The duration for the next state is randomly chosen again. The process repeats until the experiment is over.

3.2.2 Actions

The action space A consists of all steering actions *the agent* can perform. 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 (reduced action space in table **??**). The motivation behind this is the assumption that strong steering actions are seldomly needed while driving on a highway. Leaving them away might reduce the planning complexity.

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 **??**.

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

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). The relationship between steering actions and steering angles is illustrated in figure **??**.

Table **??** shows the discrete actions for the driver and the agent. The discrete values have been chosen empirically. More actions allow for more precise control.

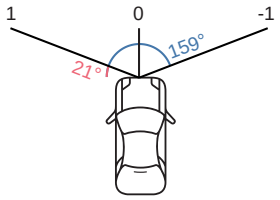

Figure 3.4: Mapping steering actions to steering angles

However, the number of actions is the branching factor for the search tree construction during planning. Thus, it has a strong 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.

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 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.

						Driver's action space								
-2	-1	-0.75	-0.5	-0.25	-0.15	-0.1	0	0.1	0.15	0.25	0.5	0.75	1	2
					Agent's reduced action space									
						Agent's full action space								

Table 3.2: Discrete steering actions for the driver and the agent.

The actions of the driver are naturally continuous. We discretize them by mapping their continuous values to the closest value in the discrete action space, as it can be seen in table **??**.

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 3.3: Driver action discretization. For negative values, the *To* value is exclusive. For zero, both *From* and *To* are inclusive. For positive values, the *From* value is exclusive.

3.2.3 Reward

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 **??**). 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 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 **??** shows the reward function for the agent that incorporates both targets. The relative yaw angle is denoted as θ, and ϕ represents the lane centeredness (see figure **??**). A lane centeredness 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 (**reward1**; **reward2**). 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 indirectly.

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

3.2.4 Observations and observation probabilities

The observation space O includes all possible observations. The observed attributes are displayed in table **??**. The agent observes sensory information about the car's current lane centeredness and relative yaw angle. Moreover, the driver's last action is observed. At any time step, the agent only observes the action of the driver for the last time step. The agent has to estimate the most likely next action of the driver by considering the history of past observations. The observations are discrete. They are discretized by mapping their naturally continuous values to the closest value from the values listed in table **??**. The conditional observation probabilities Z are not explicitly given but implicitly defined by the dynamics of TORCS and the driver model.

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 transitions. 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 the simulation engine of TORCS and an interface to the driving model to simulate

Observation	Description and values
Relative yaw angle	Angle between car direction and track axis direction. The continuous values between $-\pi$ and π radians are discretized using 101 bins: Values: { $-\pi$, $-49/50\pi$, ..., 0, ..., $-49/50\pi$, π }
Lane centeredness	Horizontal distance between the car and the lane center. 0 when the car is centered in its lane, +1 if the car is on the left edge of the lane, and −1 if the car is on the right edge of the lane. Greater numbers than +1 or smaller numbers than −1 indicate that the car is off-lane. The continuous values between $-\infty$ and ∞ are discretized using 103 bins: Values: { right off-lane, -1, $-49/50$, ..., 0, $49/50$, 1, left off-lane }
Driver steering (last time step)	The agent perceives the last input of the human. This is the action of the human at the last time step. The agent does not know which action the human is going to choose simultaneously to its own action. −1 means full right and +1 means full left. The values are discrete. Values: See table **??**.

Table 3.4: Observations for the agent. *See figure* **??** *for an illustration of the yaw angle and the lane centeredness.*

transitions by performing actions starting from arbitrary belief states. The generative model then returns the next state for both driver and TORCS, a reward (using the reward function from section **??**), 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.

3.3 Solution approach using the POMCP algorithm

3.3.1 Solving procedure of POMCP

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 (**pomcp**). Thereby, the curse of dimensionality and the curse of history are *broken* (see section **??**). POMCP 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 the solution might not be very good if the algorithm is stopped early.
POMCP constructs a search tree of histories (see figure **??**). Sequences of states, observations, and rewards are simulated using a generative model. Given a state and action, the generative model returns a successor state, observation, and reward. The simulations allow POMCP to estimate the value of histories without knowing the exact transition dynamics of the POMDP. For each represented

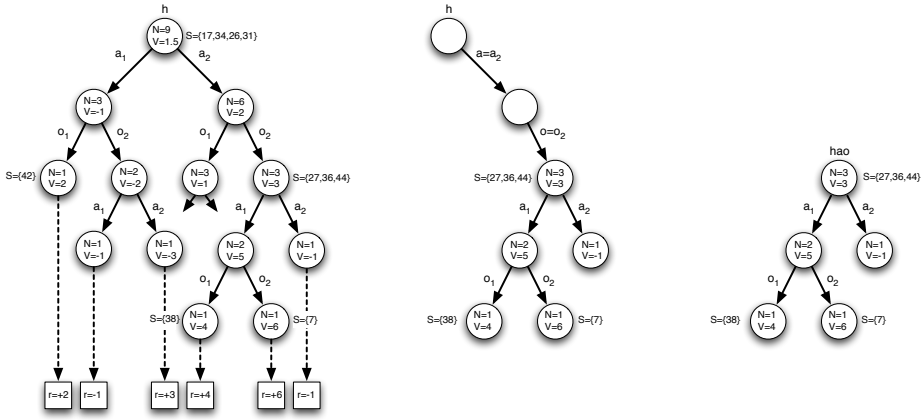

Figure 3.5: A POMCP belief tree with two actions and two observations. The agent simulates action-observation trajectories starting from the current belief at history h (left). The action with the highest mean return from the simulations is executed in the real environment, and the agent receives a real observation o (middle). The tree can then be pruned, as all other histories are rendered invalid. The process repeats from the new history hao (right). *Illustration from:* **pomcp**.

history, a node $T(h) = \langle N(h), V(h) \rangle$ is maintained. $N(h)$ stands for the number of times the history h has been visited, while $V(h)$ stores the history's value. The value is estimated by the mean return from all searches that start at history h.

The belief over the states is approximated by employing an unweighted particle filter approach. The belief is represented by a collection of states. For every history h visited during searches, the belief $B(h)$ is updated to include the simulation state (particle) returned by the generative model. The more likely a state is at a history, the more often it is expected to occur in simulations. As a consequence, the relative frequency of a state in the belief approximates its probability.

Figure **??** illustrates the process of POMCP (Figure **??** shows the pseudocode of the algorithm). If the belief for the current history h_{real} does not contain particles at the beginning of a planning episode ($B(h_{real}) = \emptyset$), the agent has lost track of the environment's state completely. If this happens, we consider the planner to have failed and select actions randomly from this point on. Otherwise, a start state for a search is sampled from the belief at the current history: $s \sim B(h_{real})$.

Searches are divided into two stages. During the first stage, which is active as long as child nodes exist for all children, actions are selected using the Upper Confidence Bounds 1 (UCB1) algorithm (**ucb1**). UCB1 chooses actions by the principle of optimism in the face of uncertainty; the values of actions are increased with an exploration bonus that is higher for rarely tried actions (see equation **??**, c is a constant hyperparameter).

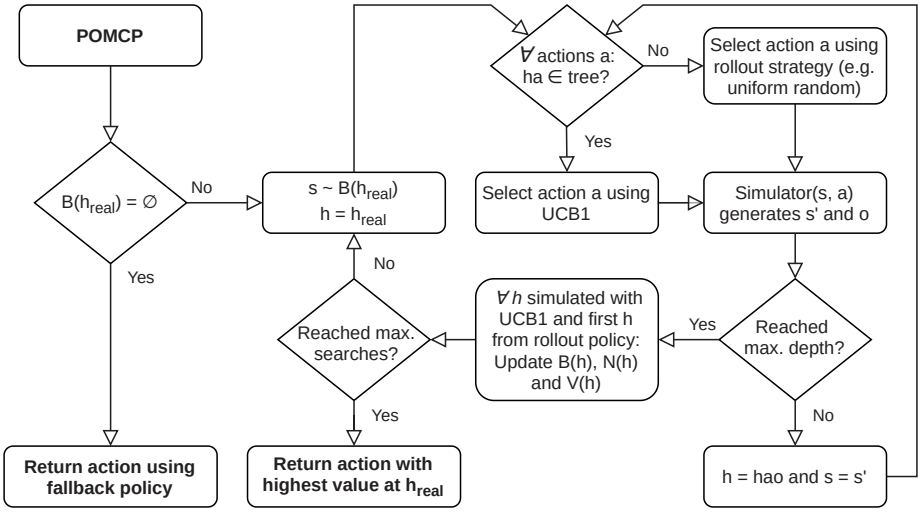

Figure 3.6: Flow chart illustrating the Partially observable Monte-Carlo Planning (POMCP) algorithm

$$V_{UCB1}(ha) = V(ha) + c\sqrt{\frac{logN(h)}{N(ha)}} \tag{3.3}$$

If any history is visited for the first time, the algorithm continues in the second stage: rollout. During this stage, the agent cannot rely on previous experiences to weigh the actions. Hence, actions are selected randomly. Only for the first new history encountered during a rollout, a node is added to the tree. More histories may be simulated, but they are not included in 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 encountered during the search are updated by backpropagating the rewards through the tree.

To select which action to perform in the real environment, a fixed number of searches is performed from the current history. 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 (see figure **??**). Then, the process repeats from the new history.

The number of searches performed during the planning has a substantial impact on the quality of the derived policy. Unfortunately, the computational complexity is exponential with respect to the number of performed searches. However, for a good policy, only a finite number of searches are required. The performance is expected to increase with the number of searches only until convergence occurs (**pomcp**).

Algorithm 1 Partially Observable Monte-Carlo Planning

procedure SEARCH(h)
  repeat
    if $h = empty$ **then**
      $s \sim \mathcal{I}$
    else
      $s \sim B(h)$
    end if
    SIMULATE($s, h, 0$)
  until TIMEOUT()
  return $\underset{b}{\text{argmax}}\ V(hb)$
end procedure

procedure ROLLOUT($s, h, depth$)
  if $\gamma^{depth} < \epsilon$ **then**
    return 0
  end if
  $a \sim \pi_{rollout}(h, \cdot)$
  $(s', o, r) \sim \mathcal{G}(s, a)$
  return r $+ \gamma$.ROLLOUT($s', hao, depth+1$)
end procedure

procedure SIMULATE($s, h, depth$)
  if $\gamma^{depth} < \epsilon$ **then**
    return 0
  end if
  if $h \notin T$ **then**
    for all $a \in \mathcal{A}$ **do**
      $T(ha) \leftarrow (N_{init}(ha), V_{init}(ha), \emptyset)$
    end for
    return ROLLOUT($s, h, depth$)
  end if
  $a \leftarrow \underset{b}{\text{argmax}}\ V(hb) + c\sqrt{\frac{\log N(h)}{N(hb)}}$
  $(s', o, r) \sim \mathcal{G}(s, a)$
  $R \leftarrow r + \gamma$.SIMULATE($s', hao, depth + 1$)
  $B(h) \leftarrow B(h) \cup \{s\}$
  $N(h) \leftarrow N(h) + 1$
  $N(ha) \leftarrow N(ha) + 1$
  $V(ha) \leftarrow V(ha) + \frac{R - V(ha)}{N(ha)}$
  return R
end procedure

Figure 3.7: Pseudocode for the POMCP algorithm. *Source:* **pomcp**.
Notes:

- A
- B

3.3.2 Action and observation space discretization

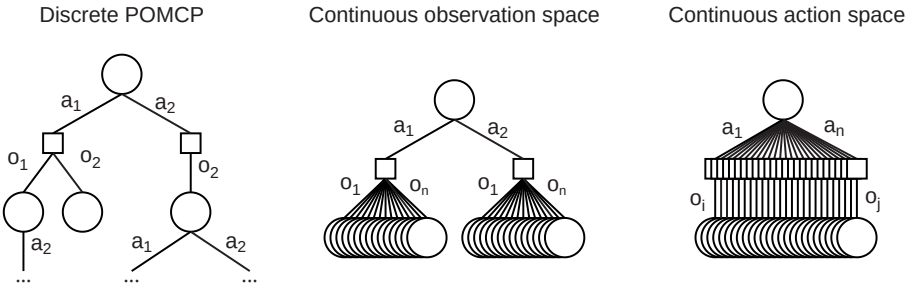

Figure 3.8: Comparison of POMCP belief trees with discrete observations (left) and continuous observations (right) with two actions.

POMCP is not suited to solve continuous POMDP. However, using POMCP with 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. The shared control lane-keeping task naturally has continuous actions and observations. To account for continuous action and observation spaces,

discretization is necessary. Figure **??** shows how POMCP behaves when tasked with solving POMDP with continuous observation or action spaces without discretization. If the observations are continuous, the search tree cannot extend beyond the first observation layer as every observation is unique, and thus, no history will ever be visited twice. In the case of continuous actions, the chance of executing the same 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 (**pomcp_continuous**).

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

3.3.3 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 sometimes converges towards a state that is far from the environment's true state. Particles that differ from the converged state have a low probability while sampling (low relative count). Hence, after every iteration, they become scarcer until they are completely erased from the belief. At this point, the agent is sure to be in an erroneous 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 (**decision_making_book**). While this reduces the accuracy of the belief, it prevents its complete convergence towards a wrong state.

Thus, 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 **??** 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 (**pomcp**), the number of transformed particles 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.

3.3.4 Preferred actions

Domain knowledge narrowly focuses the search on promising states without altering asymptotic convergence.

pomcp improve the performance of POMCP by providing domain knowledge in the form of preferred actions to the agent. Preferred actions are initialized with an initial value, increasing the likelihood to be selected by the UCB1 algorithm. For example, **pomcp** implemented preferred actions for the game Pac-Man, where the player is chased by ghosts in a maze who aim to catch and eat her. By collecting a *power pill*, the player is enabled to eat the ghosts instead. When using preferred actions, the value of moving in directions in which the player sees ghosts is based on the power pill. If the player collected one, the value of moving towards ghosts is increased, if not, it is decreased. Thereby, the agent is made aware of basic domain knowledge, and its performance is increased.

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 a countermeasure when a distracted driver turns strongly in the wrong direction. The idea is to make minor actions more likely to be chosen during searches by giving them a higher initial value than stronger actions. Thereby, less-severe steering actions are selected and tried out first by the UCB1 algorithm. If they lead to good results, the agent can exploit them quicker. If not, the agent will also evaluate less preferred actions.

We employ preferred actions to evaluate their effect on the performance of the agent. The design we use was empirically chosen and is not meant to provide a fully correct representation of driving domain knowledge. 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 **??**. 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 3.5: Preferred actions probabilities and initial values. The positive and negative values *each* have the same probabilities and values.

Chapter 4

Experimental setup

In this chapter, our experiments are introduced. First, an overview of the different evaluated scenarios is given in section **??**. Subsequently, in section **??** some important experiment design decisions are explained and justified. Furthermore, the process of tuning the hyperparameters is discussed in section **??**. Lastly, the performance metrics for the evaluation are presented in section **??**.

All experiments have been performed single-threaded, but in parallel, on a Google Cloud C2 Compute-optimized[1] *virtual machine with 60 CPU 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 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 randomly. The second agent only considers a subset of the action space containing only minor steering actions (see table **??**). This is done because we do not expect strong steering actions to be needed often. 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 **??**). 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 **??**).

The driver model configurations are introduced in section **??**. 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 makes nine scenarios (three agents × three driver models). For each of these scenarios, various experiments are executed with varying numbers of forward searches during planning with the POMCP algorithm (see section **??**). 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

We made a number of noteworthy experiment design decisions 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 50 repetitions (runs) of episodes consisting of 1,000 actions each, if no terminal state is reached earlier. 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. 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 acceleration and braking are controlled by an external controller, keeping the speed constant at 80 km/h during the entire episode.

4.2.2 Experiments do not use real-time simulation

It is important to consider that the simulation is not executed in real-time because this limits the applicability of this study to a real-world scenario. The agent and driver provide a new steering action every 0.1 seconds. For the time in between, their steering action stays constant. 1,000 actions represent 100 seconds of simulated driving time. This means that the car drives 100 seconds in TORCS. However, because the planning time before each action may be substantial, the experiment execution can take much longer in real-time. While the agent plans, the simulation *waits*; it is paused until the agent decides on its next action and executes it. This is not realistic and would not work with a real human and a real car. Section **??** discusses how performance optimizations could be implemented in order to circumvent this problem.

4.2.3 Choice of evaluated number of searches

The number of searches impacts the required planning time significantly. Increasing the number of searches leads to an increased planning time. In the experiments, we evaluate different numbers of searches to assess the performance

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

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

impact. We expect the performance to converge after some amount of searches and want to find out when this convergence occurs. Table **??** shows the various values that are used in the experiments. The computational complexity of the planning process grows relative to the number of searches. Therefore, it was decided to increase the relative distance between the values we try as the values increase. This prevents a precise determination of the convergence point. However, otherwise, the experiments would not be computationally feasible with our resources. To further narrow down the convergence point, additional experiments would be required.

4.2.4 Initial belief

The initial belief of the agent consists of 1,000 particles that each consist of a randomly sampled attentive driver model state and a car state that is slightly modified but close to the true car state. The lane centeredness and the yaw angle of the real car state are increased or decreased by a randomly chosen amount of up to five percent. Thereby, the agent does not have perfect knowledge of the car's current lateral position but has a close initial estimate.

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 **??**.

4.2.5 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 **??**). This means that every combined action is repeated 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 a 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.

4.2.6 Driver action discretization

The actions of the driver are discrete (see section **??**). 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 **??**). However, they make it a lot harder to predict the driver's actions precisely.

4.2.7 Controlling randomness to ensure comparability

Every run has a different seed for the random number generator used for the randomly chosen state durations. Every run has different state duration times. However, the driver state durations are the same for the different agents in the runs. For example, during run one, the driver is attentive or distracted at the same times and for the same durations for all agents. The reason for this setup is twofold: First, varying driver behavior for different runs verifies the robustness of the agents to varying situations. Second, using the same state durations for the different agents 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 assessed by the average cumulative return from an experiment with 20 runs, up to 1,000 actions each, and 1,000 searches per planning step. The second driver model is used for the grid search (steering overcorrection, no action noise; see section **??**). 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 are 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 number 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 the lower bound, the driver model driving without assistance. Second, as the upper bound, an agent with perfect knowledge of the state, who always acts optimally.

Chapter 5

Results

This chapter presents the results from the experiments. First, in section **??** the performance of the driver without assistance, and of an agent that acts optimally, are established as the lower and upper performance bounds respectively. In section **??**, the results of the hyperparameter optimization are presented. Then, the influence of the number of searches during planning on the performance is evaluated in section **??**, 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 **??** 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 the lower bound, and the performance of an agent that always reacts optimally to the driver's actions as the upper bound. For both baselines, 50 runs with up to 1,000 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 **??** shows that the more complex the driver model is, the worse its performance is. 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 its last attentive action. In some cases, this means that the distracted driver 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 1,000 actions each and 1,000 searches per planning step.

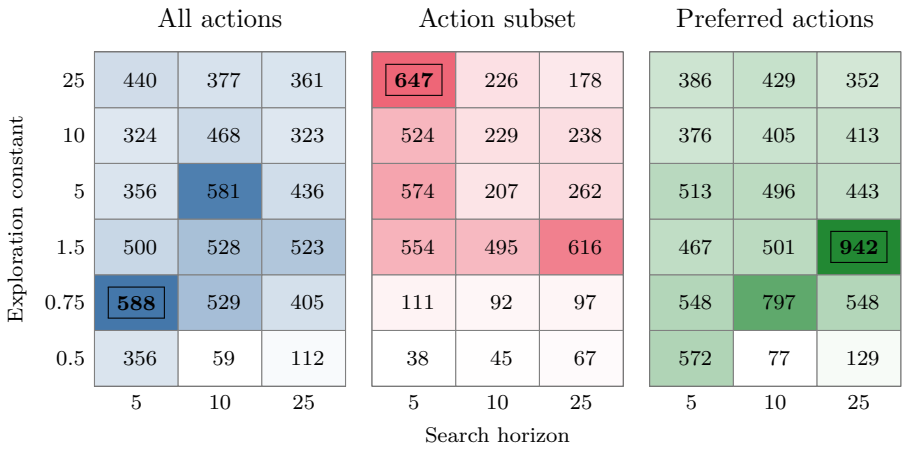

Figure 5.1: Average cumulative rewards for combinations of search horizon and exploration constant for all three action configurations (20 runs with up to 1,000 actions each and 1,000 searches per planning step; driver model with steering overcorrection but without noise). The framed values indicate the combination with the best result that we use for further experiments.

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 ten 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 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 Performance impact of the number of searches during planning

We want to determine after how many searches during planning the lane-keeping performance of the three evaluated agents converges. Performing more searches means evaluating more possible scenarios. In turn, more evaluated scenarios enable the agents to form better policies. However, after some number of searches, the information gain from additional searches decreases, and performance is expected to converge (**pomcp**). We evaluate each of the agents in the scenarios with the simple driver model, the driver model with steering overcorrection after regaining attentiveness, and the driver with overcorrection and noise. A lower number of searches is preferable as each search requires a substantial amount of planning time. The applicability of a planner is limited by the planning time required for good results. 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.

5.3.1 Simple driver model

Figure **??** illustrates the performance differences between evaluations with various different numbers of searches during planning for the experiment with the simple driver model. All evaluated agents lead to similar convergence behavior. Already with just 200 searches, the average cumulative reward is above 600 for all agents. However, as it can be seen in Table **??**, 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 **??**). Then, their belief deviates noticeably from the true states of the car and the driver. The agents are not able anymore to make accurate assumptions about 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 reaches 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 1,500 searches. Then, 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.

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 7,500 searches.

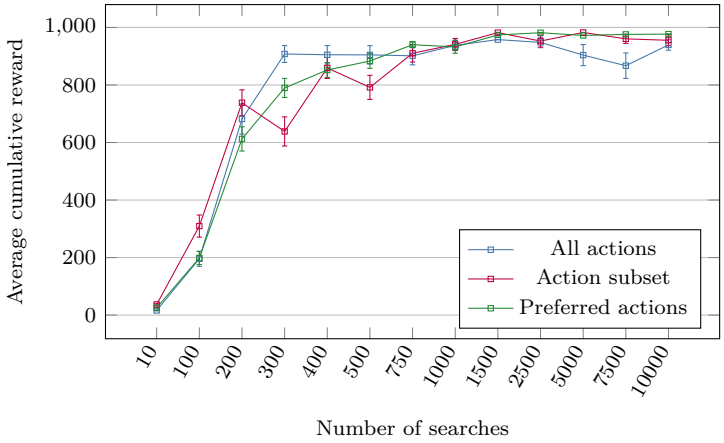

Figure 5.2: Performance comparison of POMCP when utilizing all actions, an action subset, or preferred actions with a simple driver model. Each point shows the mean cumulative reward from 50 runs with 1,000 actions each, if no terminal state is reached earlier.

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 during 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 2,500, 7,500, and 10,000 searches. These are not caused by particle deprivation. In all cases, the car is in a road bend, and the driver is distracted, steering in 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 1,500 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 in the wrong direction in a road bend, and the agent is able to correct the steering by effectively reversing the driver's actions.

Table **??** also includes the average planning times the agents require to perform all their searches in each planning episode. The planning times are mostly influenced by the search horizon. Whereas the agents without preferred

Number of searches		10	100	200	300	400	500	750	1000	1500	2500	5000	7500	10000
All actions	# term. states	50	50	22	4	4	3	3	1	0	1	4	6	1
	∅ plan. time (s)	.02	.06	.12	.14	.16	.19	.28	.37	.55	.88	1.76	2.67	3.59
Action subset	# term. states	50	48	27	26	11	18	6	5	0	2	0	2	2
	∅ plan. time (s)	.02	.05	.09	.13	.16	.19	.31	.37	.56	.89	1.85	2.73	3.58
Pref. actions	# term. states	50	50	32	13	8	4	1	3	0	0	1	0	0
	∅ plan. time (s)	.11	.25	.39	.55	.70	.87	1.35	1.91	3.52	5.96	9.23	16.71	22.36

Table 5.2: For each of the experiments with the simple driver model, this table includes the total number of runs during which a terminal state was reached and the average planning time per planning step in seconds. *Note: .xx is short for 0.xx in the table.*

actions both only look five actions ahead during planning, the agent with preferred actions considers 25 possible future actions in its tree. Therefore, it needs to perform five times the number of simulations during planning, which requires more time. The planning time is needed before executing each action. In the most extreme case, with 10,000 searches, this means that the agent with preferred actions needs to plan for an average of 22.36 seconds of real-time before deciding on the action for the next 0.1 seconds of simulated driving time. The simulation halts during planning. However, in a real-world scenario, this is not feasible.

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 needs to account for the driver's possible overcorrection. The amount of overcorrection is stochastic (see section **??**). 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.

As it can be seen in Figure **??**, 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 agents without preferred actions (see Table **??**). For example, with 750 searches, the agent using preferred actions receives an average cumulative reward of 883.43 with only ten 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 mean 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

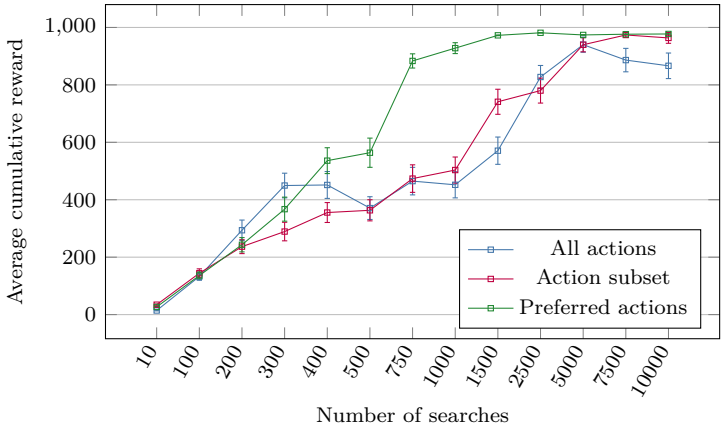

Figure 5.3: Performance comparison of POMCP when utilizing all actions, an action subset, or preferred actions with a driver model that over-corrects when it regains attention. Each point shows the mean cumulative reward from 50 runs with 1000 actions each, if no terminal state is reached earlier.

Number of searches		10	100	200	300	400	500	750	1000	1500	2500	5000	7500	10000
All actions	# term. states	50	50	22	4	4	3	3	1	0	1	4	6	1
	∅ plan. time (s)	.02	.05	.08	.12	.16	.20	.28	.37	.55	.91	1.75	2.72	3.63
Action subset	# term. states	50	48	27	26	11	18	6	5	0	2	0	2	2
	∅ plan. time (s)	.02	.05	.08	.12	.15	.19	.28	.36	.53	.88	1.84	2.69	3.54
Pref. actions	# term. states	50	50	32	13	8	4	1	3	0	0	1	0	0
	∅ plan. time (s)	.10	.24	.40	.54	.68	.88	1.22	1.69	2.51	4.30	9.39	16.15	22.53

Table 5.3: For each of the experiments with the driver model with steering over correction, this table includes the total number of runs during which a terminal state was reached and the average planning time per planning step in seconds. *Note: .xx is short for 0.xx in the table.*

not account for this possibility properly, no matching node for the observation resulting from the overcorrection is contained in the agent's planning tree. In this case, the agent cannot recover and continues using uniformly random action selection (see section **??**), 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.

Moreover, table **??** presents the average planning times needed by the agents. These are comparable with the results for the scenario with the simple driver model.

5.3.3 Steering overcorrection and noise

The noise added to the driver's actions is added with the goal to make the driver's behavior more realistic and thereby also more difficult to plan with. However,

the performance of the agents in the experiment with overcorrection and noise is very similar to the experiment with overcorrection but without noise. Figure **??** displays the performance differences for the agents with various numbers of searches. 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 1,000 searches or more. The agent with a full action range reaches peak performance at 5,000 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 7,500 searches and more. The actual convergence probably occurs with somewhere between 5,000 and 7,500 searches, but no experiment was conducted with a number of searches in between.

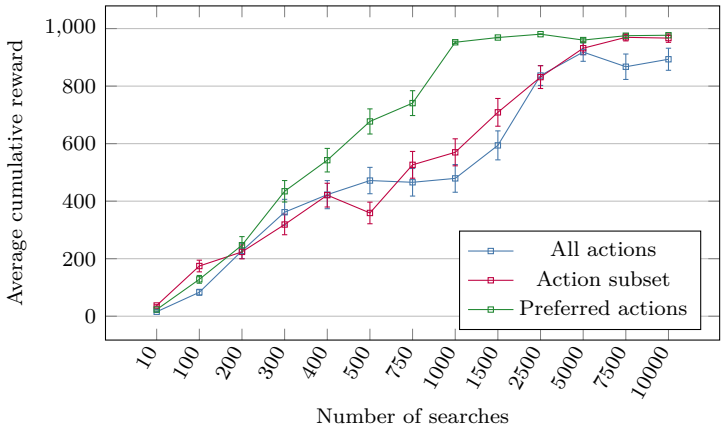

Figure 5.4: Performance comparison of POMCP when utilizing all actions, an action subset, or preferred actions with a driver model that over-corrects when it regains attention and performs noisy actions. Each point shows the mean cumulative reward from 50 runs with 1,000 actions each, if no terminal state is reached earlier.

Table **??** shows the number of terminal states reached during the experiments and the average planning times needed. The numbers of terminal 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. The average planning times are similar to the times from the experiments with the other two driver models.

Number of searches		10	100	200	300	400	500	750	1000	1500	2500	5000	7500	10000
All actions	# term. states	50	50	22	4	4	3	3	1	0	1	4	6	1
	∅ plan. time (s)	.02	.05	.09	.12	.16	.21	.29	.37	.55	.92	1.78	2.73	3.01
Action subset	# term. states	50	48	27	26	11	18	6	5	0	2	0	2	2
	∅ plan. time (s)	.02	.06	.09	.12	.17	.19	.28	.38	.55	.89	1.80	2.69	3.61
Pref. actions	# term. states	50	50	32	13	8	4	1	3	0	0	1	0	0
	∅ plan. time (s)	.12	.24	.39	.55	.68	.84	1.25	1.66	2.53	4.20	9.13	16.66	22.21

Table 5.4: For each of the experiments with the driver model with steering over correction and noise, this table includes the total number of runs during which a terminal state was reached and the average planning time per planning step in seconds. *Note: .xx is short for 0.xx in the table.*

5.4 Mean lane centeredness

The reward reflects how well the agents manage to keep the car centered in the 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 and terminal states reached. In this section, we want to emphasize the difference in lane centeredness that stems from using more searches during planning. More specifically, we consider the mean absolute lane centeredness, which is the average distance to the lane center, regardless of whether the deviation is on the left or right side.

For the experiment with the driver model with driver action noise and steering overcorrection, figure **??** shows the average absolute lane centeredness for an increasing number of searches. We focus on this figure in the analysis. The results for the other two scenarios are similar (see appendix **??** and appendix **??**). 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 **??**).

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 agent with preferred actions at first glance. However, still, almost all runs of the two agents with unweighted actions lead to terminal states, and only about half of the runs of the agent with preferred actions. The performance of the agent with preferred actions is arguably better. Using 1,000 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 10,000 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.

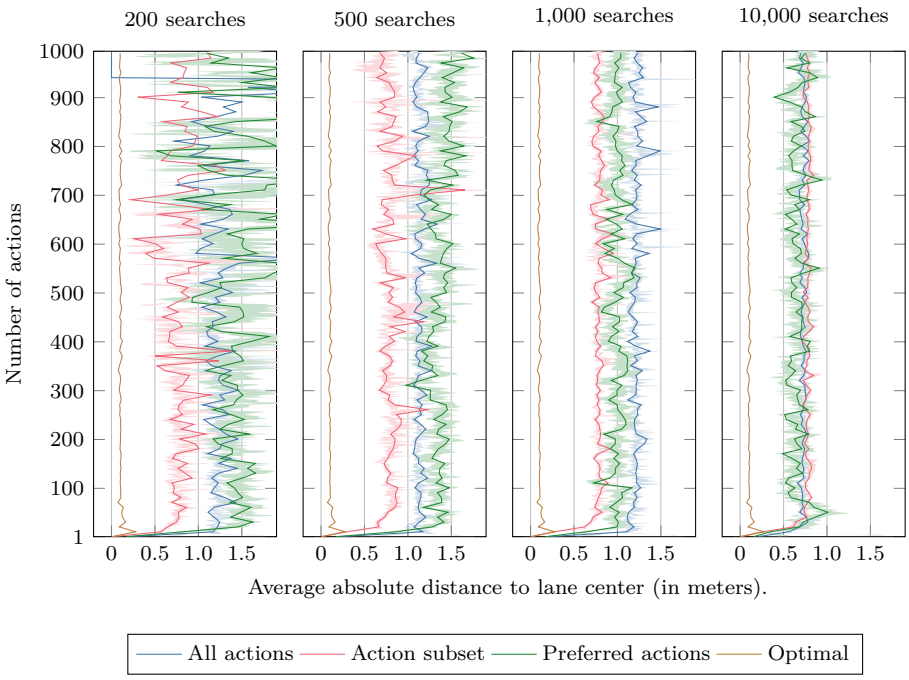

Figure 5.5: Mean lane centeredness for the different agents in experiments with the driver model with oversteering and driver action noise. A lower distance is favorable. The shaded area shows the standard error. The optimal agent does not build a search tree, and thus there is no difference between the four plots for the optimal agent. *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* **??***). 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, we analyze the results that were presented in the previous chapter. The reasons for the performance differences but also the similarities of the agents are explained in section **??**. Furthermore, section **??** elaborates on limitations of the chosen solution approach.

6.1 Analysis of the results

6.1.1 Lower and upper performance bound

Without assistance, the driver is not able to keep the car in its lane in any scenario. This is because the driver is not able to account for changes in the road trajectory when distracted. It is unlikely that an actual human that becomes distracted performs as poorly as our driver models. A realistic driver model was not in the scope of this thesis. However, the incapability of a distracted driver to keep the car in its lane only makes the task harder for the agent. All agents are able to significantly improve the driver's lane-keeping performance.

We determined the upper performance bound using an agent acting optimally (or almost optimally, as the discretization does not allow for very fine-grained control). The optimal agent had full knowledge of the car's position and the driver's attentiveness and achieved average cumulative rewards of 999.3 in all scenarios. Considering the theoretical maximum average cumulative reward of 1,000, this means that an almost perfect return is possible if the agents estimate the driver's attentiveness correctly. Especially with many searches, the agents come close. However, none of the evaluated agents is able to consistently estimate the state of its environment correctly and gain an optimal return.

6.1.2 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 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 number 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 a relatively low 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.

6.1.3 Performance impact of the number of searches during planning

All agents can lead to good policies if they perform a high number of searches. For the simple driver model, convergence occurs significantly earlier than with the less predictable driver models. 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 1,500 searches onward, for the agents using the full action space, both weighted and unweighted, the terminal states reached during runs are caused by particle deprivation. 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 1,500 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 results as the experiments without noise. The most likely reason for this is the discretization applied to the driver's actions (see section **??**). 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 5,000 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 **??** 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 **??**). 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 the preferred action. 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.

6.1.4 Mean lane centeredness

Increasing the number of searches during planning leads to better lane-centering for all agents. This was to be expected as the ability to estimate the driver's state correctly increases with the number of searches. With 10,000 searches, the car can mostly be kept between 0.5 and 1.0 meters from the lane center. The optimally-acting agent is able to consistently keep the distance to the lane center close to zero. In contrast, it is clear that none of the agents is able to estimate the driver's state accurately enough to keep the car centered at all times.

It is difficult to compare this result with the lane-keeping performance achieved in other papers as the experiment setups differ widely. The experiments in different studies differ in their road setup, car speed, action frequency, and car simulation dynamics. However, it is highly likely that commercially available systems today that do not account for the driver's distraction would outperform our approach significantly. Nevertheless, this does not undermine our contribution because the fundamental premise of assisting a potentially distracted driver by solving a POMDP model including only driver performance measures is proven to work. Nevertheless, improvements are necessary.

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 shortcomings of the online solution approach taken in this thesis is the long planning time the agents require for the 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 7,500 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 7,500 searches with a search horizon of 25 means the agent has to simulate a total of 187,500 actions, which takes about 16 to 17 seconds on average (see tables **??**, **??**, and **??**).

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 achieved. First, the efficiency of the generative model can be improved. Second, the POMCP algorithm could be parallelized or potentially even replaced by a more efficient algorithm. Both options are discussed in section **??**.

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 were also used as the generative model for the agent. Therefore, the transition and observation probabilities underlying the generative model were 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 with a more realistic driver model, such a generative model might not be available. It is conceivable that it could be sufficient to use a generative model that approximates the dynamics of the environment. This has not yet been evaluated and likely leads to problems if the approximation is not 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 **??**). 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 lower the number of discretization bins, 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 **??** discusses briefly how this obstacle could be overcome.

Furthermore, the actions of the driver are discretized as well. This reduces the driver's precision which is unacceptable in any realistic scenario. 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 large range of different driver actions that are represented by a single 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 are necessary to verify this assumption. Possible approaches to handle continuous actions are discussed in section **??**.

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 (**behavior_adapt**). 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 in the wrong direction repeatedly, the driver model does not adapt to the agent by adjusting its behavior. 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

This chapter concludes the study of this thesis about distraction-aware lane-keeping assistance with a human in the loop. Section **??** presents the conclusions of this thesis. Section **??** outlines the necessary adaptations required to apply our approach with a more complex driver model or a human driver.

7.1 Conclusion

In this thesis, an agent acting as a lane-keeping assistance system is conceived that estimates the driver's distraction online, sharing control over the car with the driver. The agent does not observe whether the driver is attentive or distracted. For its assessment of the driver's distraction, the agent relies solely on observations about driving performance measures, such as the steering actions of the driver, and sensory information about the position of the car. The problem is modeled as a POMDP to account for the uncertainty about the driver's distraction and the exact positon of the car. The POMCP algorithm (**pomcp**) is applied to solve the POMDP online. To the best of our knowledge, this is the first work that applies online POMDP solving to address the uncertainty about a driver's distraction in a shared control lane-keeping scenario while relying only on commonly available measures as observations.

The main conclusions of this thesis can be summarized as follows:

1. We provide a POMDP model with a continuous state as a representation of the shared control lane-keeping scenario, outlining how both the human driver and the car's dynamic can be simulated.

2. Our model for the human driver is simple. However, our modeling approach is also suitable for the integration of a more sophisticated driver model (see section **??**).

3. We enable an agent to act as a lane-keeping assistant to the driver, taking into account the driver's potential distraction. The POMDP is solved online by applying the POMCP algorithm. Experimental results show that the driving performance is enhanced.

4. Particle deprivation is a common problem with a particle filter approach such as POMCP. Implementing particle injection (see section **??**) and introducing domain knowledge by the use of preferred actions (see section **??**) leads to an improvement.

5. Using the TORCS driving simulator as a generative model during planning with POMCP is not efficient enough for a real-time scenario. The

performance needs to be significantly optimized. Suggestions on how to achieve this are provided in section **??**.

6. The lane-keeping performance of our approach is inferior to traditional lane-keeping assistance systems. The autonomy of the driver is not significantly increased by estimating the driver's distraction. The immediate application of the approach is not advisable. Section **??** outlines opportunities for improvement.

7. Our simulation method allows for a repetition of experiments. Problems can be revisited and analyzed. This is important in the safety-critical domain of automated driving.

7.2 Road toward application with human drivers

In this section, adaptations and improvements that we suggest for the application of our approach with a more complex driver model or a human driver are discussed.

7.2.1 Avoiding particle deprivation

The main problem encountered during planning is particle deprivation. This happens when the belief of the agent diverges substantially from the true state. In our case, this is mostly caused by either taking into account too few possible future trajectories during planning or not successfully identifying and focusing on more probable future scenarios. Taking into account too few possibilities can be circumvented by increasing the exploration. However, this has a negative performance impact. A more promising solution would be to improve the performance of the generative model we use for the search during the planning phase. The next section addresses this issue. Enabling the agent to focus on more probable scenarios can be achieved in two ways: First, one can pass domain knowledge to the agent, as we did by using preferred actions. Second, offering the agent more sophisticated observations could lead to a more accurate belief state.

The domain knowledge we offer our agent, with the chosen implementation of preferred actions, is not very extensive nor accurate. It is based on the assumption that strong steering actions are less likely to be needed. Providing better domain knowledge may improve the agent's action selection during planning substantially. For example, this could be achieved by learning an initial policy offline from driving data and providing it to the agent (see **combining_on_offline**).

It is possible to classify driver distraction using driver performance measures alone (**dist-det-perf**). However, estimating the driver's attention using only her past steering actions and the car's position is difficult. Including additional information may be helpful. This could be additional performance measures, such as acceleration and braking behavior. Additionally, using more sophisticated information is conceivable, such as driver's eye gaze, head position, or even

biometric factors, like her heart rate or brain activity. The likelihood of a good prediction increases with a stronger correlation of the observations with the driver's attentiveness. Nevertheless, it must be considered how likely it is to have access to this data in a realistic driving situation.

7.2.2 Performance optimization

Our experiments have shown that a considerable planning time is required for the agent to achieve acceptable lane-keeping performance. There are two main causes for this. Firstly, the generative model we use for the Monte-Carlo forward simulation during planning is not very efficient. Secondly, the POMCP algorithm itself is not very efficient; forward simulations during planning are sequential.

The generative model uses TORCS for the simulations of the driver states. Thousands of trajectories need to be simulated. The number of performed searches and the search depth determine the volume of simulations performed during the online planning. For example, with 10,000 searches and a search horizon of 25, up to 250,000 individual simulations are performed during each online planning step before the next action can be executed (less than 250.000 if terminal states are reached before the search horizon). We did not perform any sophisticated profiling. Nevertheless, it is very likely that the performance of TORCS could be highly optimized. However, optimizing TORCS would require an extensive amount of work. This may be avoided by either replacing TORCS with a simpler, more efficient driving simulator or by parallelizing the planning. A simpler model may represent the driving dynamics less accurately but can potentially still lead to good results (see **hitl_pomdp** for an example of a simple mathematical model). Parallelization of the planning would enable the use of multiple instances of TORCS simultaneously. Thereby, multiple possible future scenarios can effectively be evaluated at once.

pomcp-parallel show that POMCP can be parallelized by constructing multiple search trees simultaneously during the planning phase and merging them afterward. The planning time is reduced without strongly negatively impacting the solution quality. Another, more potent parallelized MCTS algorithm that could be considered instead of POMCP is DESPOT-α (**despot-a**). It utilizes both CPU and GPU parallelization and could decrease the required planning time substantially while keeping a similar performance or even improving it. However, it requires the observation probabilities Z to be known explicitly for the POMDP (see section **??**). It may be possible to approximate the observation probabilities. A more in-depth investigation of the necessary adaptations would be necessary.

7.2.3 Integrating a sophisticated driver model

The POMCP algorithm is taking a model-based planning approach. The driver models we use in our experiments are simple and not suited to accurately model real human behavior. The POMCP algorithm is dependent on the accuracy of the model used for planning. In our experiments, the same model is used to

represent the driver during evaluation and to simulate drivers for the search tree construction during planning. With a real human driver, this is not possible. If the agent shares control over the car with a human, a sophisticated, accurate model is required for Monte-Carlo sampling during planning. Driver behavior modeling is an active research field. It is not within the scope of this thesis to provide an overview of driver modeling approaches. Extensive reviews of different methods can be found in literature by **model-review-3**, **model-review-4**, and **model-review-5**.

The integration of a more advanced driver model is straightforward. It can be used as a replacement for one of the three driver models we use in our experiments. The only requirement is an interface to be used as a generative model for the agent to sample future states and observations during planning. The exact composition of the driver model state can be arbitrary, as the agent never evaluates the driver model state directly but only stores it in a belief particle to be used in the next forward search.

7.2.4 Continuous action and observation space

We utilize discretization to be able to use POMCP with naturally continuous action and observation spaces (see section **??**). Discretization is necessary for POMCP to work. However, it comes with the drawback of a loss of precision. Even if one would increase the resolution of the discretization - using more bins, some degree of information loss is inevitable. The alternative would be to work with the continuous spaces directly. A switch from POMCP to another POMDP solver would be required.

Various solvers have been developed to work with fully continuous POMDP. Two algorithms worth mentioning are POMCPOW, an extension of POMCP using progressive widening (**online_pomdp_cont**), and LABECOP, which, in contrast to POMCPOW, avoids limiting the number of observations considered during planning (**online-cont-pomdp-2**). However, both methods require the observation probabilities Z to be explicitly known, which is not the case in our scenario. One would need to find a way to circumvent this problem. A potential approach would be to approximate the observation probability distribution.

7.2.5 Use of active probing

Human drivers likely react to the actions of the agent, with potentially different reaction times or behavior for distracted drivers. The driver models used in our experiments did not account for this. **att_intersec** show that actively probing human drivers can reduce the uncertainty about their mental state. Enabling the agent to gather information about the human by actively probing her might help to improve the agent's performance. In a scenario with a human driver, the internal state of the human is very complex and therefore hard to estimate. Active probing, such as intentional small steering actions, even if they are not necessary, could be used to gauge the state of the driver more accurately.

Appendices

Appendix A

Lane centeredness comparison

A.1 Simple driver model

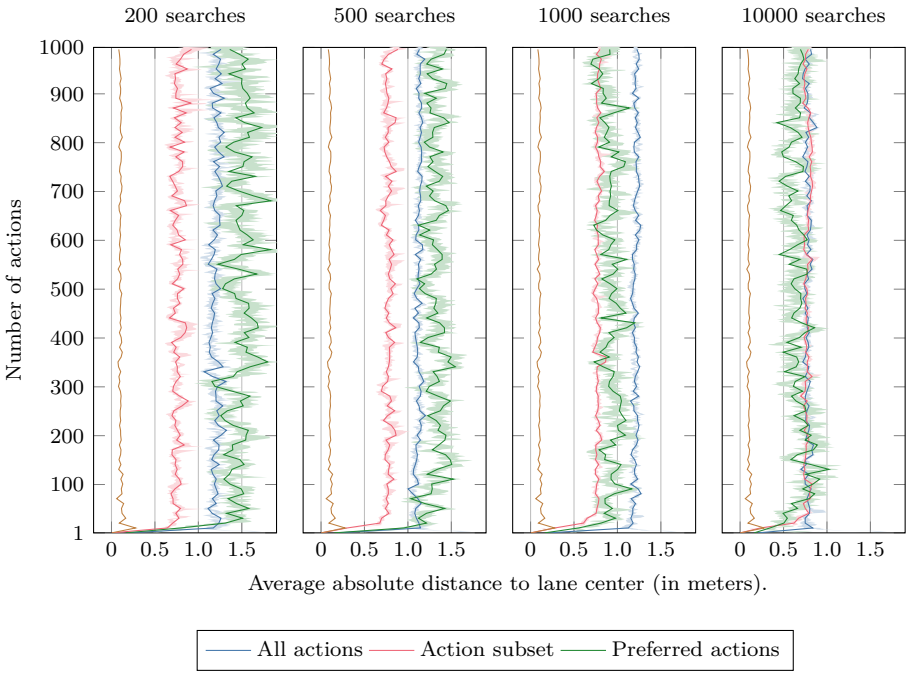

Figure A.1: Mean lane centeredness for the different agents during experiments with the simple driver model. In principle, a lower distance is favorable. The shaded area shows the standard error. The optimal agent does not build a search tree, and thus there is no difference between the four plots for the optimal agent. *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.*

A.2 Driver model with steering overcorrection

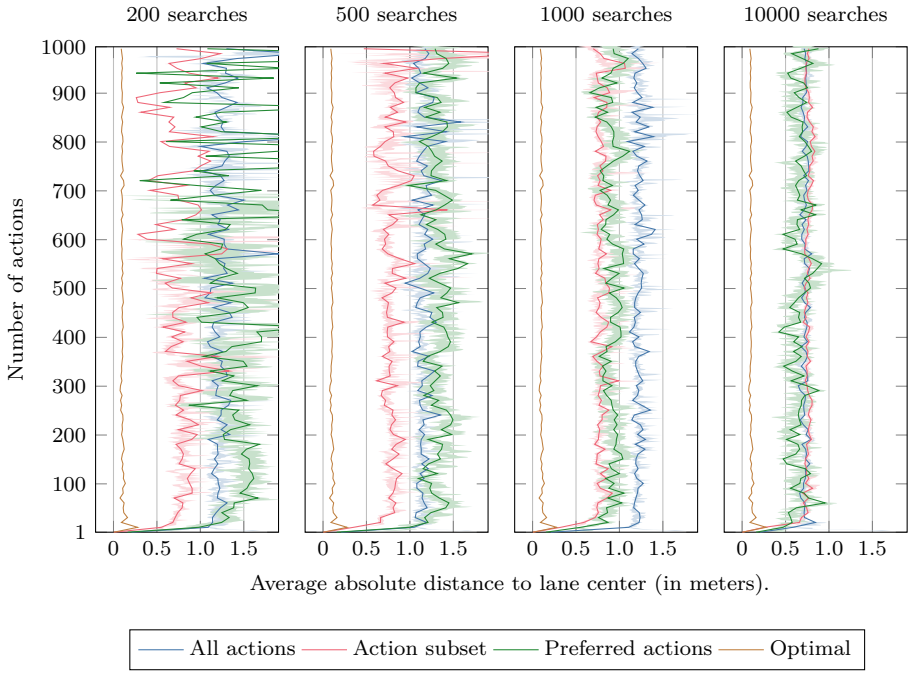

Figure A.2: Mean lane centeredness for the different agents during experiments with 1,000 actions, with oversteering and driver action noise, using 200, 500, 1,000, or 10,000 searches while planning. In principle, a lower distance is favorable. The shaded area shows the standard error. The optimal agent does not build a search tree, and thus there is no difference between the four plots for the optimal agent. *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,*