Karlsruhe Institute of Technology

Motion Planning for Automated Vehicles in Uncertain Environments

Master's Thesis of

Maximilian Beck

Institute of Measurement and Control Systems
Karlsruhe Institute of Technology

Reviewer:	Prof. Dr.-Ing. Christoph Stiller
Second reviewer:	Prof. Dr.-Ing. Eric Sax
Advisor:	Ömer Şahin Taş, M.Sc.

Karlsruhe, April 2021

Declaration / Erklärung

Ich versichere hiermit wahrheitsgemäß, die Arbeit bis auf die dem Aufgabensteller bereits bekannten Hilfsmittel selbständig angefertigt, alle benutzten Hilfsmittel vollständig angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Maximilian Beck
Karlsruhe, 12.04.2021

Acknowledgement / Danksagung

An dieser Stelle möchte ich allen danken, die mich während meiner Masterarbeit und des zweiten Corona-Lockdowns im Winter 2020 und Frühjahr 2021 unterstützt und motiviert oder auf eine andere Art und Weise zum Gelingen dieser Arbeit beigetragen haben.

Vielen Dank an Prof. Christoph Stiller vom Institut für Mess- und Regelungstechnik und Prof. Eric Sax vom Institut für Technik der Informationsverarbeitung für die Betreuung dieser Arbeit.

Ein besonderer Dank gilt Ömer Şahin Taş für seine Zeit, die er sich stets genommen hat, um meine vielen Fragen zu beantworten, für die vielen Treffen, Telefonate, Zoom Meetings, und für die vielen Tipps sowohl zu dieser Arbeit als auch für meine persönliche Zukunft. Danke, Sahin!

Zum Schluss möchte ich mich ganz besonders bei meinen Eltern und meiner Familie bedanken, die mir mein Studium ermöglicht und mich in all meinen Entscheidungen immer unterstützt haben.

Karlsruhe, April 2021 *Maximilian Beck*

Abstract

Autonomous vehicles operate in complex environments. In fact, due to limited sensor range, noisy sensor data, occlusions, or unknown intentions of close-by human drivers, the environment is highly uncertain and only partially observable. Motion planning algorithms must tackle these challenges to generate safe and human alike behavior for automated vehicles. In this thesis, the motion planning problem is modeled as partially observable Markov decision process (POMDP), considering intended routes of other vehicles as a hidden state variable in the state space. This POMDP is then solved with a Monte-Carlo tree search based online solver, which uses weighted particle filter updates for state estimation and tree search. The particle filter weights are calculated based on features that effectively combine map information with true or simulated observations of the scene. In this way the planner is able to infer the intentions of other drivers online and directly uses this knowledge for subsequent planning steps. The algorithm will then be evaluated on interactive urban intersection scenarios (e.g. roundabout scenarios) with real driving data.

Kurzfassung

Autonome Fahrzeuge werden in sehr komplexen Umgebungen betrieben. Aufgrund von begrenzter Sensorreichweite, verrauschten Sensordaten, Verdeckungen oder unbekannten Intentionen von anderen menschlichen Fahrern sind diese Umgebungen sehr unsicher und nicht vollständig beobachtbar. Bewegungsplaner müssen mit diesen Herausforderungen umgehen, um ein sicheres und dem Menschen sehr ähnliches Fahrverhalten zu generieren. In dieser Arbeit wird deshalb das Bewegungsplanungsproblem als einen teilweise beobachtbaren Markov Entscheidungsprozess (POMDP) modelliert, bei dem die beabsichtigte Route von anderen Fahrzeugen die nicht-beobachtbare Variable im Zustandsraum ist. Dieses POMDP wird dann mit einem Algorithmus gelöst, der auf Monte-Carlo Baumsuche basiert und einen gewichteten Partikelfilter sowohl für die Zustandsschätzung als auch für die Baumsuche verwendet. Die Partikelgewichte werden dann mit Merkmalen berechnet, die simulierte oder wirkliche Beoabachtungen der Verkehrsszene mit Karteninformationen kombinieren. Auf diese Weise ist der POMDP Planer im Stande von den Beobachtungen online auf die Intention von anderen Fahrern zu schließen und das so gewonnene Wissen direkt für nachfolgende Planungsschritte zu nutzen. Die Evaluation des POMDP Planers erfolgt dann in interaktiven, urbanen Kreuzungsszenarien (wie z.B. einem Kreisverkehr) mit aufgezeichneten Verkehrsdaten.

Contents

Acronyms

ABT	Adaptive Belief Tree
FIB	Fast Informed Bound
HSVI	Heuristic Search Value Iteration
IPFT	Information Particle Filter Tree algorithm
KDE	Kernel Density Estimation
MCTS	Monte-Carlo Tree Search
MDP	Markov Decision Process
PBRS	Potential-based Reward Shaping
PBVI	Point-based Value Iteration
pdf	Probability Density Function
PF	Particle Filter
PFT-DPW	Particle Filter Trees with Double Progressive Widening algorithm
PFT	Particle Filter Tree algorithm
POMCP	Partially Observable Monte-Carlo Planning algorithm
POMCPOW	Partially Observable Monte-Carlo Planning with Observation Widening
POMDP	Partially Observable Markov Decision Process
ρPOMDP	POMDP with belief-dependend rewards
QMDP	POMDP solver assuming full knowledge about the state after the first action
SARSOP	Successive Approximations of the Reachable Space under Optimal Policies
SIR	Sequential Importance Resampling
SIS	Sequential Importance Sampling
UCB	Upper Confidence Bound
UCT	Upper Confidence Bound for Trees algorithm

Nomenclature

POMDPs

$\mathbb{S}$	state space (continuous)
$\mathcal{S}$	state space (discrete)
$\mathbb{A}$	action space (continuous)
$\mathcal{A}$	action space (discrete)
$\mathbb{O}$	observation space (continuous)
$\mathcal{O}$	observation space (discrete)
$\mathcal{T}$	transition model
$\mathcal{R}$	reward model
$\mathcal{Z}$	observation model
γ	discount factor
π	policy
V	value function
Q	Q-value function
$()^*$	optimal quantity
s	state
a	action
o	observation
h	action-observation history
b	belief state or belief distribution
$\hat{b}$	approximated belief state
τ	belief update function
ρ	belief-dependent reward function

Probability and Information Theory

p or Pr	probability density function
$\mathcal{N}$	Normal distribution
$\mathcal{H}$	Entropy

Algorithms

$\mathcal{G}$	generative model or black box simulator
d	tree depth
N	visitation count for tree nodes
M	counter for how often a history has been generated
R	cumulative reward
B	particles approximating a belief at tree nodes
W	particle weights for weighted particle approximations
C	children of a tree node
ha	history appended by the next action
hao	history appended by the next action and observation
T	entire search tree

Driving POMDP model

$\mathcal{O}$	global Cartesian coordinate system
$\mathcal{F}_{r_k}$	Frenet coordinate system of route r_k of vehicle k
V_0	ego-vehicle
V_k	other vehicle in the scene
$N_{\mathcal{V}}$	number of other vehicles in the scene
$N_{\mathcal{R}_k}$	number of route options of another vehicle k
$\mathbf{s}_{V_0}$	state of the ego vehicle
$\mathbf{s}_{V_k}$	state of another vehicle k
$\mathbf{o}_{V_0}$	observation of the ego vehicle
$\mathbf{o}_{V_k}$	observation of another vehicle k

1. Introduction

1.1. Motivation

In recent years, the development of more and more sophisticated Advanced Driver Assistance Systems (ADAS) has increased the level of autonomy in driving. Those systems are aimed to increase safety and comfort, but still require fully attentive human drivers. However, in specific operational design domains (ODD) such as on highways it is possible that the car takes over control and the driver will be notified only in those situations were manual control is needed [Ros17]. According to the SAE taxonomy, this is defined as level 3 out of six levels of driving automation, ranging from no automation (level 0) to full driving automation (level 5) [SAE14]. So far those systems — even if technically feasible and mature, have not been deployed on the roads because of legal issues. Those issues have been resolved with the adoption of an international regulation for automated lane keeping systems (ALKS) by the United Nations Economic Commission for Europe (UNECE) which becomes effective in 2021 [Dil20] and allows level 3 autonomous driving on highways.

Current research and development focuses on level 4 or higher autonomous cars which are capable of handling complex urban environments with multiple pedestrians, bicycles and other vehicles in near-collision scenarios. The goal is to design a system, which shows a human-like driving style in urban traffic along with regular human-driven vehicles. Such autonomous cars may have a high impact on people's life, not only due to an increase in comfort and efficiency in driving, but also due to a significant reduction in the number of accidents and deaths in traffic. Moreover, the potential availability of fully autonomous transportation systems enables completely new opportunities for mobility solutions.

For designing Automated Driving Systems (ADS), a general approach is to divide the driving problem in several subproblems and solve each subproblem separately. The result is a model- or system-based approach in which each module defines an interface to other modules, processes the input data and provides data for the next module. The information flow of this approach from the sensors (e.g. Lidar, Cameras, Radar, etc.) that perceive the environment to the actors (e.g. steering, brake, throttle) that manipulate the state of the vehicle is depicted in Figure 1.1(a). This is a typical pipeline that starts with feeding raw sensor inputs to localization and object detection modules, followed by behavior prediction and a planning or decision-making module. At last, the control module generates motor commands for the actors based on the planned behavior of the ego-vehicle.

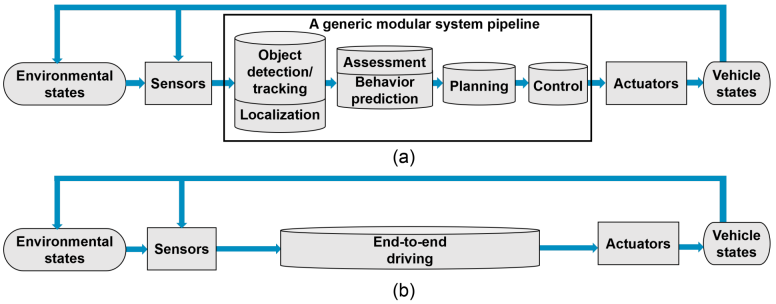


Figure 1.1.: Information flow diagrams of two different approaches to automated driving: (a) Model-based approach, (b) End-to-end learning approach (from [YLCT20]).

Another approach is to use End-to-end driving methods, that learn human-style driving by either direct supervised deep learning where the training data is generated with an expert human-driver or deep reinforcement learning which learns the optimal way of driving by receiving reward signals from interacting with the world depending on the quality of the selected actions. Those systems generate actions for the ego-vehicle directly from sensory inputs as can be seen in Figure 1.1(b). One drawback of this approach is the huge amount of training data necessary to train those models, since the training-data should contain as many edge cases as possible. However, it is impossible to record all edge cases for training, and it cannot be guaranteed that the learned models generalize to situations which could not be included in the training data. Hence, with this approach, urban driving has not been achieved so far [YLCT20].

In contrast, the technical feasibility of automated driving in urban environments with the modular approach has been shown with various experimental vehicles in recent years (e.g. [ZBS+14]). However, their driving style is still very different from human drivers and is often described as overly defensive or conservative. The reason for this is that autonomous cars have to deal with many uncertainties resulting from sensor noise or range limitations in perception, from uncertain prediction due to hidden or unobservable variables (e.g. intentions or goals of other drivers) or from unknown possible interaction of other traffic participants with the ego-vehicle. To maintain safety, optimization-based motion planners avoid taking risky actions in those ambiguous scenarios, since they do not plan with uncertain information [ZBDS14]. More recent approaches try to consider uncertainties in the optimization-based planning approaches [TS18]. However, the result is often a rule-based behavior, which depends on different maneuver options in an urban scenario.

1.2. Goal Description

In this thesis, the focus lies on behavior generation of the ego-vehicle by using motion planning techniques that take the aforementioned uncertainties into account, without including any prior knowledge in form of behavior rules. Hence, the framework of Partially Observable Markov Decision Processes (POMDPs), which is well-known for sequential decision-making under uncertainty, will be used to model the driving scenarios. Instead of planning with the current state of the world (e.g. state of the ego- and all other vehicles) and its prediction obtained from the prediction module, POMDPs allow planning with the current belief state, which is a probability distribution about all possible states. The uncertainty of the belief state is reflected in the shape of the distribution, such that if the belief is distributed uniformly over many possible states, i.e. several states are equally likely, the planning algorithm is uncertain about its environment.

In order to generate feasible behavior for the automated vehicle even in such a case, the online POMDP solver uses its internal POMDP model and multiple random simulations to determine how the environment and ego-vehicle might evolve and selects a motion plan, which is optimal according to a prespecified reward model. The result is a POMDP planner, which integrates prediction and planning into a single, combined problem.

Hence, the first goal of this thesis is to formulate a POMDP model, that takes the aforementioned uncertainties in the environment of automated vehicles into account and when solved provides a feasible and interactive solution to the motion planning problem for urban scenarios.

The second goal is to provide an algorithm that is capable to compute a solution to this POMDP online and in near real-time.

Contribution

Within these goals, the first contribution of this work is a real-time capable software framework written in C++ and Python for solving POMDPs online. This framework uses an online MCTS algorithm applicable to POMDPs with continuous state and observation spaces and allows to actively consider the value of information for decision-making [FT20].

The second contribution is the extension of a POMDP model formulation from previous research [HSB+18]. It is extended to support dynamic map changes in every time step and a more sophisticated motion and interaction model for the other vehicles is used. Moreover, the novel POMDP model improves intention estimation of other drivers and is adapted for the use with weighted particle filters and the Particle Filter Tree algorithm.

Finally, the framework is used to compute policies for the novel POMDP model and the resulting POMDP motion planner is evaluated. It is demonstrated, that the planner is able to plan for sufficiently long horizons in near real-time and the impact of important parameters on solution quality and runtime is investigated.

Limitation

The online POMDP solver used in this thesis and the software framework supports belief dependent rewards, such as rewards based on information gain. Even though it is supported and a promising idea, so far no belief dependent rewards, and hence no active information gathering is implemented for the driving POMDP model.

The reason is, that for this, the question of how the belief dependent rewards could be calculated must be addressed. As the state space of the POMDP model includes various unrelated variables (e.g. the route intentions or positions of different other vehicles), this is not a trivial task, since it is not clear based on which quantity the rewards could be calculated. The method presented in previous works to compute the negative entropy from weighted particle sets based on kernel density estimation (KDE) cannot be applied in a straight-forward way [FT20]. Thus, including belief dependent rewards based on information measures is left for future work.

1.3. Outline

The remainder of this work is structured as follows: Chapter 2 introduces the fundamentals of decision making and state estimation. The section of state estimation focuses on particle filters since they are the method of choice for updating the belief state in POMDP solvers. Subsequently, in Chapter 3 at first the most important offline algorithms for solving POMDPs are explained, before then state of the art MCTS based online solvers are

introduced. After that, Chapter 4 presents the Information Particle Filter Tree (IPFT) algorithm, which is used for the experiments in this work. In Chapter 5, the planning problem is stated, the POMDP formulation is explained and some implementation details are highlighted. Then, the POMDP planner is evaluated in a simulation environment and the results are presented in Chapter 6. Finally, the work is concluded and future research directions are given in Chapter 7.

2. Fundamentals of Decision Making and State Estimation

In this chapter, the mathematical formulation of a sequential decision-making problem in a stochastic environment is addressed. Sequential decision-making means, that based on the state (e.g., the velocity or position) an agent (e.g., the ego vehicle) is in, it needs to select or decide for an action in order to transition to the next state multiple times in a row. Regarding automated driving, the actions could be acceleration values or steering angles for example.

The stochastic fashion of the environment comes in when the same action in the same state does not always, but only in probability, yield the same result state. If this state is fully observable to the agent, the problem can be modelled as Markov Decision Process (MDP), which will be explained in more detail in section 2.1.
Many times the current state of the world is not fully observable, which means that some state variables cannot be directly measured, but can only be inferred from the history of states, as it is the case for the intentions of other drivers for example. Then, the MDP can be extended to a Partially Observable Markov Decision Process (POMDP), where the agent does not know exactly in which state it is, but only receives observations from the world which may let the agent draw conclusions about the true, unobservable state. Section 2.2 will deal with the latter case of state uncertainty.
For a more in-depth introduction to the topic, the reader is referred to [RN10], [Koc15] and [TBF06].

2.1. Markov Decision Processes

In a MDP an agent chooses an action a_t based on the state s_t at time t. After receiving a reward r_t according to the reward model $\mathcal{R}(s_t, a_t)$, the probability of transitioning to a specific successor state s_{t+1} is given by the transition model $\mathcal{T}(s_{t+1}|s_t, a_t)$. It is hereby assumed, that the next state only depends on the current state and action and not on any previous state or action. This is called the Markov assumption or property.
To sum up, a MDP consists of the following components:

- $\mathcal{S}, \mathbb{S}$: A state space $\mathbb{S}$ or set of states $\mathcal{S}$, with s_0 being the initial state.
- $\mathcal{A}, \mathbb{A}$: An action space $\mathbb{A}$ or a set of actions $\mathcal{A}$.
- $\mathcal{T}(s_{t+1}|s_t, a_t)$: A transition model that determines the probability of reaching state s_{t+1} given that action a_t is chosen in state s_t.

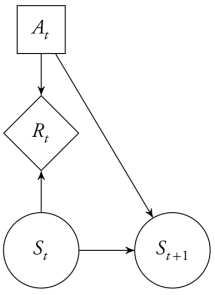


Figure 2.1.: Probabilistic graphical model of a Markov decision process (from [Koc15, p.78]).

- $\mathcal{R}(s_t, a_t)$: A reward function returning the current reward for taking a_t in state s_t.

Figure 2.1 shows a Markov decision process in a decision diagram. States are depicted as circles, actions as squares and rewards as diamonds. With the above definition of the components, a MDP is defined as the 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, where γ is the discount factor as explained in the next paragraph.

Policy and Value

According to the formulation of an MDP, the goal of an agent is clearly to choose actions, which yield a high reward. But how can such a solution, which (often) results in high rewards, be represented?
A deterministic plan, i.e. a specific action sequence from the start state is not suitable, since state transitions happen stochastically. That means that the same action sequence can result in completely different state sequences, and hence in completely different rewards. Therefore, a solution to an MDP is called a policy (one could also say a reactive plan), which determines which action an agent chooses in each state at time t:

$$a_t = \pi(s_t) \tag{2.1}$$

The difference now is, that the agent not just blindly follows a predefined action sequence, but also considers the probabilistic outcome of each action (the state transition), before the next action is selected. This is called closed-loop planning in contrast to the case of open-loop planning, where no state information is considered [Koc15, pp. 84-89].
As a policy π determines which action to choose, an agent must now decide which policy to follow, starting in the current state s. To do this, different policies can be compared by their value or expected utility $V^\pi(s)$. [1]
The value $V^\pi(s)$ of executing a policy π starting in state s is the expectation of the infinite sum over all discounted future rewards taking actions according to policy π:

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}\left(s_t, a_t = \pi(s_t)\right)\right] \tag{2.2}$$

The discount factor γ is a number between 0 and 1 and describes the agent's preference for current rewards over future rewards. When γ is close to 0, it means that immediate rewards are worth more than rewards in distant future. In case of infinite horizon problems (as in equation (2.2)) a $\gamma < 1$ ensures a finite value, provided that all rewards are finite.
Now, having computed the value of some (or all) policies, there are always one or more

[1] The value and utility are measures which assign single numbers to a policy or state in order to express the agents preferences for some policies and the desirability of a state. [RN10, Sec.16]

optimal policies, which have higher values than all the others. A rational agent, which maximizes the value or expected utility, will always choose an optimal policy π^*

$$\pi^* = \arg\max_{\pi} V^{\pi}(s). \tag{2.3}$$

Hence, the optimal policy defines an action-selection strategy, which yields the highest expected utility.

Value Iteration

There are different ways to compute the optimal policy, which differ in their computational complexity and accuracy in value. One basic algorithm to iteratively compute the optimal values of all states is value iteration. Since this involves solving nonlinear equations iteratively, value iteration is rather computational complex.
The core of value iteration is the so-called Bellman equation, after Richard Bellman [Bel57]:

$$V^*(s_t) = \max_{a_t \in \mathcal{A}(s_t)} \left(\mathcal{R}(s_t, a_t) + \gamma \sum_{s \in \mathcal{S}} \mathcal{T}(s_{t+1} = s | s_t, a_t) V^*(s) \right), \tag{2.4}$$

where $V^*(s)$ is the optimal value function.
Because of the max-operator the equation (2.4) is clearly nonlinear and states, that the optimal value of a state s_t is the immediate reward for taking the optimal action a_t of all possible actions in that state plus the discounted value of the next state.
The value iteration algorithm solves this equation iteratively. Hence, line 6 of Algorithm 2.1 is called a Bellman update.

Algorithm 2.1 Value iteration

```
1: function ValueIteration()
2:     k ← 0
3:     V_0(s) ← 0 for all states s                                   ▷ s := s_t
4:     repeat
5:         for each s ∈ S do
6:             V_{k+1}(s) ← max_a [R(s,a) + γ Σ_{s'} T(s'|s,a) V_k(s')]   ▷ s' := s_{t+1}
7:             k ← k + 1
8:     until convergence
9:     return V_k ≈ V*
```

A common termination condition, showing that the value function has converged, is $\|V_k - V_{k-1}\| < \delta$. Here, $\|\cdot\|$ denotes the max norm, where $\|V\| = \max_s |V(s)|$. In order to speed up the convergence, instead of initializing the value function with zero, it could also be initialized with a bounded function close to the optimal value function.
Now, after having computed V^*, the optimal policy can be extracted by

$$\pi^*(s_t) \leftarrow \arg\max_{a_t} \left(\mathcal{R}(s_t, a_t) + \gamma \sum_{s_{t+1}} \mathcal{T}(s_{t+1} | s_t, a_t) V^*(s_{t+1}) \right). \tag{2.5}$$

Another way for computing the optimal policy is policy iteration, which directly improves the policy in each step. In that way it can be avoided to compute the optimal value function exactly and therefore it is not needed to solve the nonlinear Bellman equations. But, as value iteration, neither is policy iteration applicable to larger problems with many states. Thus, approximations for both algorithms, which reduce the computational cost have been developed [Koc15].

Q-Value

Another useful quantity when dealing with MDPs is the Q-value (function) $Q(s, a)$, which describes the value of taking action a in state s and following the optimal policy afterwards:

$$Q(s_t, a_t) = \mathcal{R}(s_t, a_t) + \gamma \sum_{s_{t+1}} \mathcal{T}(s_{t+1}|s_t, a_t) V^*(s_{t+1}) \tag{2.6}$$

The relation to the value function is given by

$$V(s_t) = \max_{a_t} Q(s_t, a_t). \tag{2.7}$$

Notice the difference to the value function $V(s)$ introduced in (2.4) before, which describes the value of a state, whereas the Q-value gives the value of taking an action in that specific state. Hence, the Q-value is used to compare different actions, which will be especially useful when dealing with solvers for POMDPs, covered in the next chapters.

2.2. Markov Decision Processes with State Uncertainty

As introduced in the beginning of this chapter, Partially Observable Markov Decision Processes (POMDPs) can deal with partially observable environments, i.e. the agent is uncertain about the state it is in.
Thus, it cannot execute the action $\pi(s)$ based solely on the state, but also needs to consider the certainty or probability of being in state s for selecting the next action. This enables the agent not only to take actions, which maximize the value, but also to choose information gathering actions in order to enhance its knowledge about the state. To simply consider only the most likely state does not suffice.
Mathematically, POMDPs are a generalization of MDPs, because in addition to the state space $\mathbb{S}$ / state set $\mathcal{S}$, action space $\mathbb{A}$ / action set $\mathcal{A}$, transition model $\mathcal{T}$ and reward model $\mathcal{R}$, a POMDP is extended by:

- $\mathcal{O}, \mathbb{O}$: An observation space $\mathbb{O}$ or set of observations $\mathcal{O}$.
- $\mathcal{Z}(o_{t+1}|s_t, a_t, s_{t+1})$: An observation model that determines the probability or probability density of receiving observation o_{t+1} given that in state s_t the action a_t was taken and the state transition to s_{t+1} happened.
  Note that $\mathcal{Z}$ is sometimes defined as $\mathcal{Z}(o_{t+1}|s_t, a_t)$, $\mathcal{Z}(o_{t+1}|a_t, s_{t+1})$ or $\mathcal{Z}(o_t|s_t)$ [PGT06, RPPCd08, TBF06, Koc15].

The POMDP problem structure in Figure 2.2 is an extension to the MDP structure from Figure 2.1. Because of this similarity the MDP (white symbols in Figure 2.2), which is included in a POMDP is also referred to as *underlying MDP* [SPK13, p. 6]. Analog to MDPs, a POMDP is defined as 7-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, \mathcal{Z}, \gamma)$.

Belief

Since the agent has no access to the true states, but only to the observations given by the observation model, the action a_t depends on the complete action-observation history h_t the agent has encountered so far at time t.

$$h_{0:t} = h_t = \{a_0, o_1, a_1, o_2, \ldots, a_{t-1}, o_t\} \tag{2.8}$$

As time goes on, this full history trace can get very long, so that it is not practicable to plan in each time step with the complete action-observation sequence. Instead, the history

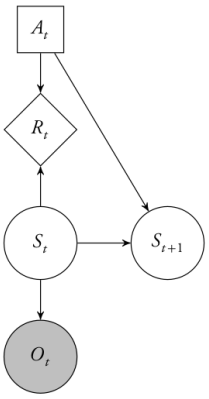


Figure 2.2.: Probabilistic graphical model of a POMDP with observation model $\mathcal{Z}(o_t|s_t)$ (from [Koc15, p.135]).

at time t can be summarized by a posterior probability distribution over all states, called belief state or belief distribution [PGT06, RPPCd08]:

$$b_t(s) = \Pr(s_t = s|h_t, b_0), \tag{2.9}$$

where b_0 is the initial belief state, i.e. distribution over all states.
If the state space is discrete and finite, $b_t(s)$ can also be written as belief vector $\boldsymbol{b}_t = (p_0, p_1, \ldots, p_{|\mathcal{S}|-1})^\top \in \mathbb{R}^{|\mathcal{S}|}$ with p_i describing the probability of being in state i. Due to the constraint, that all state probabilites must sum to one, the space of possible belief vectors, i.e. the belief space $\mathcal{B}$ is sometimes also called belief simplex, denoted Δ [PGT03].
Smallwood and Sondik showed that the belief state $b_t(s)$ is a sufficient statistic for the history [SS73]. As a result, the agent bases its next action only on the current belief state (cf. the MDP policy in Eq. (2.1)):

$$a_t = \pi(b_t) \tag{2.10}$$

For sequential action selection, starting from the initial belief b_0, the belief needs to be updated recursively in every time step. This is done with the belief state update function $\tau(b, a, o)$, where

$$b_t = \tau(b_{t-1}, a_{t-1}, o_t) \tag{2.11}$$

This equation formulates the (recursive) Bayesian filter problem. Depending on the structure of the state space (continuous or discrete), the form of the belief distributions and the restrictions on the transition and observation model regarding linearity, the belief update function may have different implementations [TBF06, Koc15].
In case of discrete and finite state spaces the belief update function may look like [RPPCd08]:

$$\begin{aligned} b_{t+1}(s_{t+1}) &= \tau(b_t, a_t, o_{t+1})(s_{t+1}) \\ &= \frac{1}{\Pr(o_{t+1}|b_t, a_t)} \mathcal{Z}(o_{t+1}|a_t, s_{t+1}) \sum_{s \in \mathcal{S}} \mathcal{T}(s_{t+1}|s_t = s, a_t) b_t(s), \end{aligned} \tag{2.12}$$

where $\Pr(o_{t+1}|b_t, a_t)$, the probability of observing o after taking action a in belief b, is used as normalizer such that b_t remains a probability distribution:

$$\Pr(o|b, a) = \sum_{s' \in \mathcal{S}} \mathcal{Z}(o|a, s') \sum_{s \in \mathcal{S}} \mathcal{T}(s'|s, a) b(s) \tag{2.13}$$

Note that in these equations it is assumed that the states, actions and observations are elements of the finite sets $\mathcal{S}$, $\mathcal{A}$, $\mathcal{O}$. Other implementations of the Bayes filter and a short introduction to the filtering problem are covered in Section 2.3.

Belief-State MDP

With the definitions of $\tau(b, a, o)$, $\Pr(o|b, a)$ and by defining the reward function dependent on the belief as

$$\rho(b, a) = \sum_{s \in \mathcal{S}} b(s)\mathcal{R}(s, a), \tag{2.14}$$

the POMDP can be viewed as observable MDP over belief states (called a belief MDP), where $\Pr(o|b, a)$ specifies the probability of moving from b to $\tau(b, a, o)$ by taking action a [Hau00].
It can be shown, that an optimal policy for this belief MDP is also optimal for the underlying POMDP, which means that one can solve a POMDP by solving its corresponding belief MDP on the belief-state space. This reformulation does not necessarily simplify the solution of the POMDP, since, if the physical state space of the POMDP is discrete with $|\mathcal{S}|$ states, the belief-state space of the belief MDP is now continuous with $|\mathcal{S}| - 1$ dimensions. For continuous state spaces, this leads to infinite dimensional belief state spaces.
The value iteration algorithm as introduced in Section 2.1 is not capable of solving such belief MDPs. Thus, this algorithm is adapted to POMDPs next.

Value iteration for POMDPs

The value function V^* of the optimal policy π^* is still the solution to the Bellman equation (cf. Eq. (2.4)), which is now formulated for belief MDPs:

$$V^*(b) = \max_{a \in \mathcal{A}} \left(\rho(b, a) + \gamma \sum_{o \in \mathcal{O}} \Pr(o|b, a) V^* \left(\tau(b, a, o) \right) \right) \tag{2.15}$$

The Q-value (function) $Q(b, a)$ is defined analog to MDPs (see Eq. (2.6)) as the value of taking action a in belief state b:

$$Q(b, a) = \rho(b, a) + \gamma \sum_{o \in \mathcal{O}} \Pr(o|b, a) V \left(\tau(b, a, o) \right) \tag{2.16}$$

With the value iteration algorithm for POMDPs introduced by Smallwood and Sondik [SS73], it is possible to compute the solution to this equation optimally for specified finite T horizon POMDP value functions V_T (notice, with the value iteration algorithm defined in Section 2.1 we were able to compute the value of infinite horizon MDPs).
The algorithm starts with the initial value function for horizon $T = 1$, by computing the optimal one-step conditional plan, that determines which action is optimal for the first time step.

$$V_1(b) = \max_{a \in \mathcal{A}} \rho(b, a) \tag{2.17}$$

Then, the algorithm recursively computes the value function V_T at horizon T, which maximizes the sum of all future rewards the agent receives in the next T time steps for any belief state b, from the value function V_{T-1} at horizon $T - 1$. This value function update is also called Bellman update or Bellman operator H (for POMDPs) (cf. Eqs. (2.4) and (2.15)):

$$\begin{aligned} V_T(b) &= \max_{a \in \mathcal{A}} \left(\rho(b, a) + \gamma \sum_{o \in \mathcal{O}} \Pr(o|b, a) V_{T-1} \left(\tau(b, a, o) \right) \right) \\ &= H V_{T-1}(b) \end{aligned} \tag{2.18}$$

Put differently, the algorithm extends the $T - 1$ step conditional plan by another step. The conditional plan lists all possibilities of previous actions and observations and specifies

in each case which action to take. That means, with each step the number of different conditional plans grows exponentially with the possible number of observations $|\mathcal{O}|$.
Similar to the MDPs (cf. Eq. (2.5)) the optimal policy for a finite horizon T is simply to choose the action maximizing $V_T(b)$:

$$\pi_T^*(b) \leftarrow \arg\max_{a \in \mathcal{A}} \left(\rho(b,a) + \gamma \sum_{o \in \mathcal{O}} \Pr(o|b,a) V_{T-1}\left(\tau(b,a,o)\right) \right) \tag{2.19}$$

Smallwood and Sondik showed that the optimal value function for finite-horizon POMDPs can be represented by multiple hyperplanes and is therefore piecewise linear and convex. Each hyperplane can be expressed by a $|\mathcal{S}|$-dimensional vector, called α-vector. With the set of α-vectors given by $\Gamma_T = \{\alpha_0, \alpha_1, \ldots, \alpha_m\}$, the value function for horizon T can be written as the maximum over a set of hyperplanes:

$$V_T(b) = \max_{\alpha \in \Gamma_T} \sum_{s \in \mathcal{S}} \alpha(s) b(s) = \max_{\alpha \in \Gamma_T} \boldsymbol{\alpha}^\top \boldsymbol{b}, \tag{2.20}$$

where $\boldsymbol{\alpha}, \boldsymbol{b} \in \mathbb{R}^{|\mathcal{S}|}$.
In this notation each α-vector corresponds to a different conditional plan, which differs from other plans by any distinct action choice at some point in the future. Hence, as already mentioned before, the exponential growth in the number of conditional plans is equivalent to the increasing number of α-vectors, the longer the horizon gets. Fortunately, many α-vectors are dominated by others (the corresponding hyperplane lies below others for all belief points), such that the dominated vectors can be pruned away without affecting the solution. To fully understand value iteration for POMDPs, the best way is to study an illustrative example. For that, the reader is referred to [TBF06, RN10, Koc15].

Challenges of solving POMDPs

In the previous paragraphs, it was already observed, that with increasing generality of the problem formulation as POMDP instead of MDP, the computational complexity of POMDP solving algorithms increases. In the literature, there have been named especially two distinct but interdependent reasons, why solving a POMDP is challenging: The curse of dimensionality and the curse of history [PGT06]. Both curses have been encountered in this thesis already.
The curse of dimensionality describes the problem, that in a POMDP with n physical states, an optimal policy must be found over all belief states in a $(n-1)$-dimensional continuous space (since the probabilities a belief assigns to each state must sum to 1), whereas the curse of history names the exponential growth in the number of distinct possible action-observation histories the longer the planning horizon gets.
The exact value iteration algorithm for POMDPs explained above suffers from both curses, which prevents the algorithm from being applicable to larger, practical POMDP problems [PT87]. To overcome the curses, research has focused on approximate solution algorithms, which try to reduce the computational cost by computing only an approximation of the true optimal value function. The most popular ones will be presented in Chapter 3. It will be clear that the real-time requirement is still hard to meet for those algorithms.

2.3. State Estimation for Belief Updating

In this section, the focus lies on filtering, since it is of great importance for belief updating in POMDPs. Therefore, the general Bayes filter algorithm is discussed in Subsection 2.3.1. An introduction to Gaussian filters and particle filters, which are two important instances of the Bayes filter, follows in Subsection 2.3.2 and 2.3.3.

The textbooks on state estimation, especially for robotic applications, which are used throughout this section are [Sär13], [TBF06] and [Bar17]. They also provide detailed proofs and derivations of the equations stated in this section.

In equation (2.9) the belief distribution was introduced as a surrogate for the full action-observation history to describe the information an agent has about the state it might be in. The function $\tau(b_{t-1}, a_{t-1}, o_t)$ was used to update the belief distribution, after new actions and observations have been encountered.
In general, this problem of calculating the joint posterior distribution $p(s_{0:t}|h_{0:t})$ of all state sequences given the full action-observation history is called state estimation problem. A state sequence is denoted as $s_{0:t}$, i.e. the set of states $s_{0:t} = \{s_k, k = 0, ..., t\}$ encountered up to time step t. Theoretically, this posterior can be computed by straightforward application of Bayes' rule

$$p(s_{0:t}|h_{0:t}) = p(s_{0:t}|o_{1:t}, a_{0:t-1}) = \frac{p(o_{1:t}|s_{0:t}, a_{0:t-1})\, p(s_{0:t})}{p(o_{1:t})}, \tag{2.21}$$

where $p(s_{0:t})$ is the prior distribution defined by $p(s_0)$ and the (dynamic) transition model, $p(o_{1:t}|s_{0:t}, a_{0:t})$ is the observation model, i.e. the likelihood of the observations and $p(o_{1:t})$ is the normalization constant defined by

$$p(o_{1:t}) = \int p(o_{1:t}|s_{0:t}, a_{0:t-1})\, p(s_{0:t})\, ds_{0:t} \tag{2.22}$$

and $a_{0:t-1}$ is the action sequence up to time $t-1$ [Sär13, p.9].
If, as in the POMDP case, the actions and observations arrive sequentially in time and the posterior is recomputed in each time step, the term sequential state estimation is used. Since it is computationally inefficient to recalculate the full posterior distribution over all time steps every time a new action-observation pair arrives, only marginal distributions over one time step k are used.
Depending on the time step k with respect to the range of available observations state estimation problems can be divided into filtering, prediction and smoothing (see Figure 2.3) [Sär13, p.11]:

- Filtering: Computation of the marginal distribution of the current state s_k given the current action-observation history $h_{0:k} = \{a_0, o_1, a_1, o_2, \ldots, a_{k-1}, o_k\}$:

$$p(s_k|h_{0:k}) = p(s_k|o_{1:k}, a_{0:k-1}), \quad k = 1, ..., t \tag{2.23}$$

- Prediction: Computation of the marginal distribution of the future state s_{k+n} after the current time step k:

$$p(s_{k+n}|h_{0:k}) = p(s_{k+n}|o_{1:k}, a_{0:k-1}), \quad k = 1, ..., t, \quad n = 1, 2, ... \tag{2.24}$$

- Smoothing: Computation of the marginal distribution of the state s_k given a certain history $h_{0:t} = \{a_0, o_1, a_1, o_2, \ldots, a_{t-1}, o_t\}$ with $t > k$:

$$p(s_k|h_{0:t}) = p(s_k|o_{1:t}, a_{0:t-1}), \quad k = 1, ..., t \tag{2.25}$$

2.3.1. General Bayes Filter

The most general algorithm for the belief update function $\tau(b_{t-1}, a_{t-1}, o_t)$ is the Bayes filter algorithm (Algorithm 2.2), which is in fact a reformulation of equation (2.21).

$$b_t = \tau(b_{t-1}, a_{t-1}, o_t) = \textsc{BayesFilter}(b_{t-1}, a_{t-1}, o_t) \tag{2.26}$$

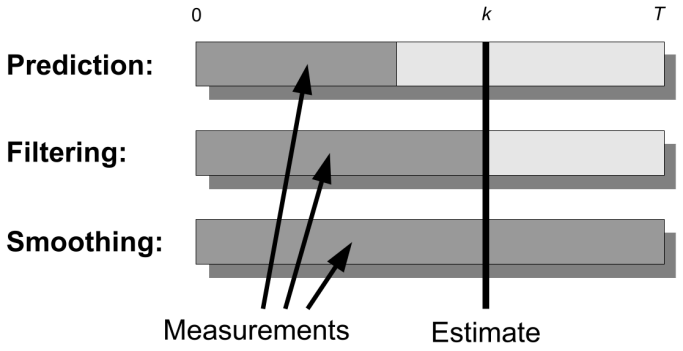


Figure 2.3.: The different state estimation problems (from [Sär13, p.11]).

Algorithm 2.2 General Bayes filter algorithm (see [TBF06, p.27])

1: **function** BAYESFILTER($b_{t-1}(s_{t-1}), a_{t-1}, o_t$)
2: **for each** state s_t **do**
3: $\overline{b}_t(s_t) = \int \mathcal{T}(s_t|s_{t-1}, a_{t-1})\, b_{t-1}(s_{t-1})\, ds_{t-1}$ ▷ prediction step
4: $b_t(s_t) = \frac{1}{\eta}\, \mathcal{Z}(o_t|a_{t-1}, s_t)\, \overline{b}_t(s_t)$ ▷ measurement update step
5: **return** $b_t(s_t)$

It recursively computes the marginal posterior belief distribution b_t for the current time step (now following the POMDP notation again) denoted as t from the belief b_{t-1}, the action a_{t-1} and the latest observation o_t.

The algorithm has two essential steps: the prediction step and the measurement update step.
In the first step, the prediction step, the belief $\overline{b}_t(s_t)$ after taking action a_{t-1} in the prior belief b_{t-1} is calculated by integrating (summing) over all possible previous states s_{t-1}. This is equivalent to simulating the belief b_{t-1} one time step forward using the transition model $\mathcal{T}$. Due to the random noise in $\mathcal{T}$ the resulting translated belief $\overline{b}_t$ is usually deformed and broadened compared to b_{t-1}.
Then, in the measurement update step, the intermediate belief $\overline{b}_t$ is multiplied by the probability of observing o_t defined by the observation model $\mathcal{Z}$. To ensure, that $b_t(s_t)$ remains a probability, the result is normalized by the factor $1/\eta$, where η is given by

$$\eta = \int \mathcal{Z}(o_t|a_{t-1}, s_t)\, \overline{b}_t(s_t)\, ds_t. \tag{2.27}$$

This step results in the belief b_t and usually narrows the density $\overline{b}_t$ since by using the latest observation more knowledge about the true state is incorporated.

For initialization of the filter algorithm one needs to specify an initial belief b_0 describing the knowledge about the state at time $t = 0$. If the value of s_0 is known with certainty, b_0 should be initialized with a point mass distribution, which is one for s_0 and zero anywhere else. In case of no prior knowledge about s_0 at all, a uniform distribution about all possible states might be used for b_0.

If the algorithm could be executed, the result would be the optimal Bayesian solution to the problem of recursively calculating the exact posterior density. Unfortunately, the general Bayes filter (Algorithm 2.2) can only be implemented for very simple estimation problems, because one needs to be able to carry out the integration in line 3 and the multiplication in line 4. This is only the case if strong assumptions about the shape of the belief distributions are made (see Sec. 2.3.2) or the state space is finite, such that the

integral becomes a finite sum (see Eq. (2.12)) [TBF06, ch. 2.4.1].
Another approach is to avoid computing exact posterior belief distributions by using a finite number of values as an approximation (see Sec. 2.3.3).

2.3.2. Gaussian Filters

Gaussian filters assume that the belief over the continuous state space is represented by (multivariate) normal or Gaussian distributions

$$b(s) = \mathcal{N}(s|\mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^N \det \Sigma}} \exp\left(-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right), \tag{2.28}$$

where $s \in \mathbb{R}^N$ is the state vector, $\mu \in \mathbb{R}^N$ the mean vector and $\Sigma \in \mathbb{R}^{N\times N}$ is the symmetric, positive-semidefinite covariance matrix. In this way, a full belief state can be described only by a vector μ and a matrix Σ, which is very useful for efficient implementations of such filters.

If the class of problems is further restricted to linear transition and observation models with added Gaussian noise, i.e.

$$\begin{aligned} s_{t+1} &\sim \mathcal{T}(s_{t+1}|s_t, a_t) = As_t + Ba_t + q_t, \\ o_{t+1} &\sim \mathcal{Z}(o_{t+1}|a_t, s_{t+1}) = Cs_{t+1} + r_t, \end{aligned} \tag{2.29}$$

where $q_t \in \mathbb{R}^N \sim \mathcal{N}(0, Q)$ and $r_t \in \mathbb{R}^N \sim \mathcal{N}(0, R)$ with the covariance matrices Q and R, the problem can be solved exactly by using the Kalman filter. The important property of the normal distribution a Kalman filter makes use of is, that any normal distributed random variable passed through a linear model remains normal distributed, such that the parameters of the posterior distribution (lines 3 and 4 of Algorithm 2.2) can be computed in closed form.

In practice the models are rarely linear, such that one often has to deal with nonlinear models. Hence, the extended and unscented Kalman filters relax the linearity assumption. They can be applied to problems with nonlinear transition and observation functions f and g with additive noise:

$$\begin{aligned} s_{t+1} &\sim \mathcal{T}(s_{t+1}|s_t, a_t) = f(s_t, a_t) + q_t, \\ o_{t+1} &\sim \mathcal{Z}(o_{t+1}|a_t, s_{t+1}) = g(s_{t+1}) + r_t, \end{aligned} \tag{2.30}$$

where q_t and r_t are defined as above. Due to the nonlinearities, the posterior belief can no longer be calculated in closed form, since after a filter update the true belief can be transformed to be non-Gaussian [Bar17, p. 101]. As a result, the goal of the extended and unscented Kalman filter is to approximate the true posterior belief (which is no Gaussian anymore) by a Gaussian distribution with parameters μ and Σ as good as possible.

The two filters differ in how they treat the nonlinearities during the filter update. The extended Kalman filter uses the (first order) Taylor expansion to linearize the functions f and g about the current state estimate mean μ and then use the Jacobian matrices in the linear Kalman equations. With this method the mean is passed through the nonlinear function f exactly, while the covariance is passed approximately through the linearized version of the function. The problem is that the estimated mean μ often is not equivalent to the true mean, which can produce large approximation errors [Bar17, p. 109].
The unscented Kalman filter avoids the linearization method used in the extended Kalman filter by relying on a transformation of the input probability density function (pdf), called sigmapoint or unscented transformation. With this transformation, the input Gaussian pdf is represented by a fixed number of points (the sigmapoints), which are directly passed

through the nonlinearity, such that the parameters of the input Gaussian can be re-estimated from the sigmapoints afterwards. Doing so, the Jacobians of the nonlinear functions are no longer needed.

However, even if there exist methods that can cope with nonlinearities (such as extended and unscented Kalman filters) the assumption that the posterior belief distribution is Gaussian is very limiting. For some tracking problems where the belief is focused around the true state with a small uncertainty margin this assumption might hold, but for many global estimation problems with very distinct hypotheses the posterior might have multiple peaks at very different positions, such that an approximation with an unimodal Gaussian is not sufficient.

As the problem of decision-making with POMDPs is similar to global estimation problems, because the agent is expected to choose between actions that rely on just these different hypotheses about the environment, Kalman filters can generally not be used in POMDPs. Instead, the method of choice are particle filters which are well-suited to handle complex multimodal beliefs and nonlinear observation and transition models.

2.3.3. Particle Filters

Particle filters are - just like the Gaussian filters from the previous section - a variant of the general Bayes filter algorithm (Algorithm 2.2). In contrast to Gaussian filters they do not make any assumption about the shape of the posterior probability density such that the belief can be highly complex and multimodal. Additionally, particle filters can handle any type of nonlinearity (no differentiability or continuity requirement) and do not even need to know the mathematical form of the nonlinear function — in practice it can be any software function [Bar17, p.108]. In these general problem formulations, particle filters clearly outperform other filtering methods [AMGC02] such that they are applied in numerous areas [DdFG01].

The particle filter achieves this flexibility by approximating the belief b_t, i.e. the posterior probability density function $p(s_t|h_{0:t})$ with a set of N_s weighted state samples $\hat{b}_t = \{s_t^i, w_t^i\}_{i=1}^{N_s}$, where $\{s_t^i, i = 1, ..., N_s\}$ is a set of support points with associated weights $\{w_t^i, i = 1, ..., N_s\}$. The weights are normalized, such that $\sum_i w_t^i = 1$. This discrete weighted approximation can be written as

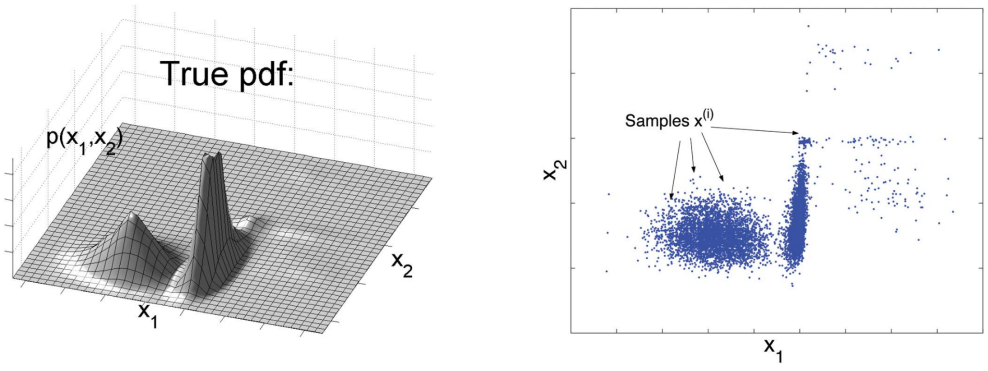
$$b_t = p(s_t|h_{0:t}) = p(s_t|o_{1:t}, a_{0:t-1}) \approx \hat{b}_t = \sum_{i=1}^{N_s} w_t^i \, \delta(s_t - s_t^i). \tag{2.31}$$

A sample (s_t^i, w_t^i), which is also called particle (hence the name particle filter), can be regarded as a hypothesis of being in that state with the weight indicating how likely the hypothesis is. Figure 2.4 shows how a sample approximation of a probability density function looks like. One can observe that the probability density value is proportional to the sample density. Note that the weight of the particles is not depicted here, i.e. the particles are considered to be unweighted (all particles have the same weight $w_t^i = \frac{1}{N_s}, i = 1, ..., N_s$).

In fact, assuming that the posterior probability density function $b_t = p(s_t|h_{0:t})$ has the shape as depicted in Figure 2.4(a), the goal of any particle filter is to find a set of particles $\hat{b}_t = \{s_t^i, i = 1, ..., N_s\}$ as plotted in Figure 2.4(b) such that the likelihood of a particle s_t^i being in the set $\hat{b}_t$ is proportional to that posterior pdf:

$$s_t^i \sim p(s_t|h_{0:t}) \tag{2.32}$$

Unfortunately, it is not possible to sample directly from $p(s_t|h_{0:t})$, since the probability density function might be highly complex and – more important – is unknown in advance



(a) 2D pdf (b) samples from the 2D pdf

Figure 2.4.: Samples drawn from a 2D probability density function (from [CGM07])

(i.e. at time steps $k < t$). Instead, the particle filter recursively computes the current particle set $\hat{b}_t$ from the previous one $\hat{b}_{t-1}$, following the general Bayes filter. In particular, every particle from $\hat{b}_{t-1}$ is simulated one step further in time using the random transition model, the taken action a_{t-1} and the obtained observation o_t.

In general, methods that mainly use multiple random simulations are called Monte-Carlo methods. Therefore, due to the sequential manner of the particle filter, the term sequential Monte-Carlo is used frequently in the literature to describe this class of algorithms. The field is reviewed by numerous surveys and textbooks and the ones used in this section are [AMGC02, CGM07, DJ09, Gus10, TBF06, Sär13].

Importance Sampling

In order to create a particle set $\hat{b}_t$ approximating a probability density function where it is not possible to sample from at time $t-1$, the particle filter makes use of a general technique called importance sampling.

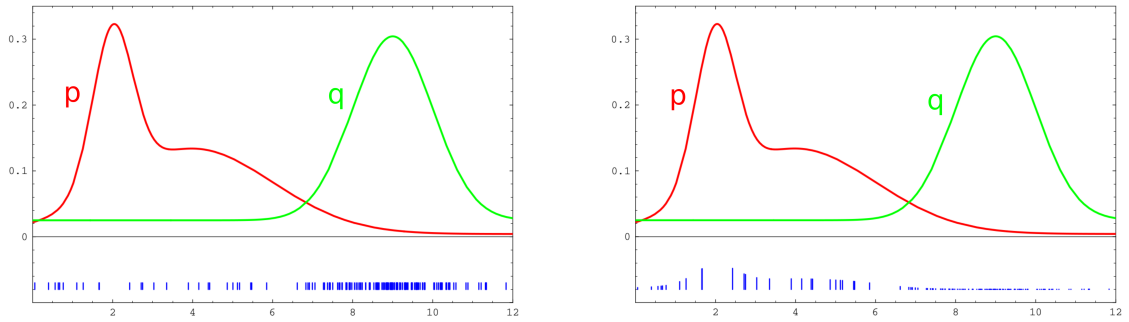
In general, importance sampling is used to approximate an arbitrary target density $p(\cdot)$, where it is difficult to sample from, by a sample set $\{x^i, w^i\}_{i=1}^{N_s}$. It is assumed, that $p(x)$ can be evaluated at any x and there exists a density $q(\cdot)$ with a support larger than $p(\cdot)$, called importance or proposal density, which can also be evaluated at any x and from which it is simple to generate or draw samples from.
Then, the x^i are generated according to $q(x)$ ($x^i \sim q(x)$) and the normalized weights w^i are used as correction factor making sure that the sample set really approximates $p(x)$ instead of $q(x)$. The weighted approximation of $p(\cdot)$ is then given by

$$p(x) \approx \sum_{i=1}^{N_s} w^i \, \delta(x - x^i), \tag{2.33}$$

where the $w^i = \frac{\tilde{w}^i}{\sum_{j=1}^{N_s} \tilde{w}^i}$ are the normalized weights computed from the unnormalized weights $\tilde{w}^i$ defined as

$$\tilde{w}^i = \frac{p(x^i)}{q(x^i)}, \qquad i = 1, ..., N_s. \tag{2.34}$$

Figure 2.5 shows the weighted samples as blue dashes on the x-axis, where the length corresponds to the importance weights.



(a) samples drawn from $q(\cdot)$ (b) weighted sample approximation of $p(\cdot)$

Figure 2.5.: Using importance sampling to approximate the density $p(\cdot)$ (from [TBF06, p.101])

Sequential Importance Sampling

Now, the importance sampling method will be applied to the general state estimation problem posed at the beginning of Section 2.3 in equation (2.21). The goal here is to approximate the posterior density $p(s_{0:t}|h_{0:t})$ by a set of weighted state sequence particles $\{s^i_{0:t}, w^i_t\}^{N_s}_{i=1}$. This can be done by choosing the importance density in a way such that the state sequences can be sampled sequentially, i.e. by starting from an initial state sample set, the state sequences are extended by one time step t in each iteration. That means the proposal density is chosen to factorize such that

$$q_{0:t}(s_{0:t}|h_{0:t}) = q_{0:t-1}(s_{0:t-1}|h_{0:t-1})\, q_t(s_t|s_{t-1}, o_t, a_{t-1}). \tag{2.35}$$

The samples $s^i_{0:t} \sim q_{0:t}(s_{0:t}|h_{0:t})$ can then be obtained by adding to each existing sample state trajectory $s^i_{0:t-1} \sim q_{0:t-1}(s_{0:t-1}|h_{0:t-1})$ the new state sample $s^i_t \sim q_t(s_t|s^i_{t-1}, o_t, a_{t-1})$.

Moreover, the multiplicative decomposition of $q_{0:t}(s_{0:t}|h_{0:t})$ also enables recursive computation of the unnormalized weights $\tilde{w}^i_t$ at time t from the normalized weights w^i_{t-1} by using the following update rule:

$$\tilde{w}^i_t = \frac{p(s^i_{0:t}|h_{0:t})}{q_{0:t}(s^i_{0:t}|h_{0:t})} \propto w^i_{t-1} \times \eta \frac{\mathcal{T}(s^i_t|s^i_{t-1}, a_{t-1})\mathcal{Z}(o_t|a_{t-1}, s^i_t)}{q_t(s^i_t|s^i_{t-1}, o_t, a_{t-1})}, \tag{2.36}$$

where the symbol $\propto$ denotes proportionality and η is a scaling factor, which takes into account the likelihood of the observation o_t given the previous action-observation history $h_{0:t-1}$ and the last action a_{t-1}. In practice, due to the subsequent renormalization of the weights, this factor does not need to be computed.

An implementation of the algorithm starts with the indexed set (e.g. an array) of weighted initial states $\{s^i_0, w^i_0\}^{N_s}_{i=1}$ and then samples in each time step t a new set of states and computes the respective weights, resulting in a new set $\{s^i_t, w^i_t\}^{N_s}_{i=1}$. If all sets are kept in memory, the algorithm indeed generates a set of weighted state sequence particles $\{s^i_{0:t}, w^i_t\}^{N_s}_{i=1}$, where all states with the same index i in the sets form a state sequence and the weight of the sequence corresponds to the weights in the last time step. However, in the case of filtering as explained earlier (see Eq. (2.23)), only the probability distribution over the states at the current time step t matters. Hence, only the previously generated set $\{s^i_{t-1}, w^i_{t-1}\}^{N_s}_{i=1}$ needs to be stored. Together with the taken action a_{t-1} and the received observation o_t, the sequential importance sampling (SIS) particle filter summarized in Algorithm 2.3 computes the current weighted particle set $\hat{b}_t = \{s^i_t, w^i_t\}^{N_s}_{i=1}$ approximating the current belief b_t. Therefore, SIS is an instance of the general Bayes filter (Algorithm 2.2).

Algorithm 2.3 Sequential Importance Sampling Particle Filter

1: **function** SEQUENTIALIMPORTANCESAMPLING($\hat{b}_{t-1} = \{s^i_{t-1}, w^i_{t-1}\}^{N_s}_{i=1}, a_{t-1}, o_t$)
2: **for** $i = 1, ..., N_s$ **do**
3: Propagate/draw particles ▷ prediction
$$s^i_t \sim q_t(s_t | s^i_{t-1}, o_t, a_{t-1})$$
4: Compute weight ▷ measurement update
$$\tilde{w}^i_t = w^i_{t-1} \frac{\mathcal{T}(s^i_t | s^i_{t-1}, a_{t-1}) \mathcal{Z}(o_t | a_{t-1}, s^i_t)}{q_t(s^i_t | s^i_{t-1}, o_t, a_{t-1})}$$
5: Normalize weights
$$w^i_t = \frac{\tilde{w}^i_t}{\sum_{j=1}^{N_s} \tilde{w}^i_t}, \qquad i = 1, ..., N_s$$
6: **return** $\{s^i_t, w^i_t\}^{N_s}_{i=1}$

The SIS algorithm suffers from a major drawback: the particle degeneracy problem. This problem describes the case that after a few iterations all the weight is concentrated on a small portion of the particles. The other particles will have negligible weights. This leads to the algorithm requiring high computational effort to update particles whose contribution to the approximation to $p(s_{0:t}|h_{0:t})$ is almost zero. The reason for this is that the state space is high dimensional since one sample $s^i_{0:t}$ represents an entire path history of state variables up to time t. As the dimension even increases with each time step it is impossible to approximate such a high dimensional probability density with particle set of fixed and practically realizable size.

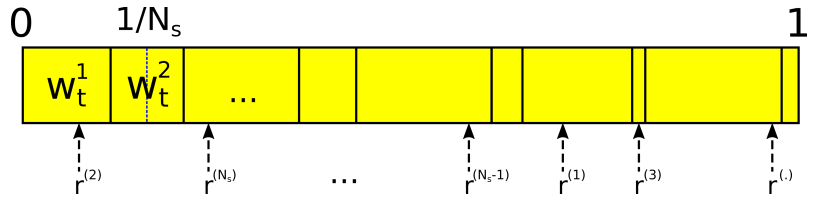
However, in the filtering case, where the focus is on approximating the low dimensional marginal $p(s_t|h_{0:t})$ resampling can be used to alleviate the particle degeneracy problem.

Resampling

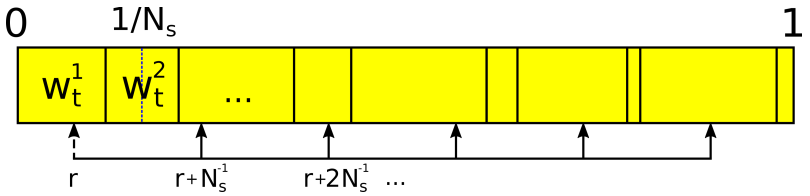
The basic idea of resampling is to replace particles with small importance weights in the set by particles with large importance weights, i.e. unlikely state hypotheses by more likely ones. This is done by generating a new set of (more likely) particles $b^*_t = \{s^{i*}_t, w^{i*}_t\}^{N_s}_{i=1}$ from the previous particle set with normalized weights $\{s^i_t, w^i_t\}^{N_s}_{i=1}$, where the particles s^{i*}_t are drawn from $\{s^i_t, w^i_t\}^{N_s}_{i=1}$ with the weight w^i_t as probability of selection. After this resampling step, all weights are reset to $w^{i*}_t = \frac{1}{N_s}$.

There exist different resampling schemes in the literature, that vary in the way how the new particles are drawn from the previous set [DC05]. To explain two of the resampling schemes, it is assumed that the normalized weights form N_s subintervals in the interval $[0, 1]$ as depicted in Figure 2.6, where each interval boundary (i.e. the vertical black bars) is computed as sum of all particle weights up to the current weight. The simplest method is then to draw N_s random numbers $r^i, i = 1, 2, ..., N_s$ from a uniform distribution over the interval $[0, 1]$ (since all weights are normalized and sum to 1) and add those particles to the new set in whose intervals the random numbers r^i fall. This scheme is called multinomial resampling and is shown in Figure 2.6(a). Due to the multiple random draws, the variance of multinomial resampling is usually quite large. Therefore, often other techniques such as residual or stratified resampling are considered [AMGC02, DC05].

Due to its ease of implementation, the $O(N_s)$ runtime and its low sampling variance, the most widely used method is systematic resampling. Instead of making N_s random draws from the interval $[0, 1]$, only a single uniformly distributed random number r in the interval $[0, \frac{1}{N_s}]$ is generated. Based on this random number, the particles are selected from the set as depicted in Figure 2.6(b) to form the resampled particle set b^*_t. The general concept of systematic resampling as depicted in Algorithm 2.4 is to compute the cumulative distribution of the particle set by adding up all N_s particle weights. It is then



(a) Multinomial resampling scheme



(b) Systematic resampling scheme

Figure 2.6.: Comparison of multinomial and systematic resampling

checked, whether the random number $U = r + (m-1) \cdot N_s^{-1}$ falls into the distribution and if so, the particle is added to the set and the cumulative distribution is updated. The result of the resampling step is an unweighted particle set, since all particles added to the final set are assigned the weight $\frac{1}{N_s}$.

Algorithm 2.4 Systematic Resampling

1: **function** RESAMPLE($b_t = \{s_t^i, w_t^i\}_{i=1}^{N_s}$)
2: $b_t^* \leftarrow \emptyset$
3: $r = \text{rand}(0; N_s^{-1})$ ▷ draw random number from uniform distribution
4: $c = w_t^{[1]}$
▷ construct the cumulative distribution function (CDF)
5: $i = 1$
6: **for** $m = 1$ to N_s **do**
7: $U = r + (m-1) \cdot N_s^{-1}$ ▷ move along the CDF
8: **while** $U > c$ **do**
9: $i = i + 1$
10: $c = c + w_t^{[i]}$ ▷ update the CDF
11: add s_t^i to b_t^*
12: **return** $b_t^* = \{s_t^{i*}, w_t^{i*}\}_{i=1}^{N_s}$

Sequential Importance Resampling

Sequential Importance Resampling (SIR), originally called Bootstrap filter [GSS93] combines Sequential Importance Sampling (Algorithm 2.3) with the Resampling algorithm introduced above with the goal to mitigate the particle degeneracy problem. To do this, the resampling algorithm (Algorithm 2.4) is executed after the weight normalization step in line 5 of the SIS algorithm (Algorithm 2.3) in each iteration of the particle filter. In the short-term, this resampling or *selection* adds additional Monte Carlo variance, but in the long-term it avoids accumulation of error. Thus, the filter algorithm becomes much more stable [CGM07].
However, it is still possible, that due to an unlucky series of random numbers or a small particle set size the particle set becomes degenerate, i.e. the weight is concentrated on a few particles and only those are selected during resampling. This phenomenon is also called *particle deprivation*, *particle depletion* or *particle impoverishment*. To avoid the case

that after resampling all particles are equal, a small portion of noise could be added to the particle set by replacing some particles by random particles. However, one should only add random noise to the particle set, if other technique for fixing the deprivation problem have failed [TBF06, p.113].

Standard Particle Filter

An important extension to this concept is to do resampling only in case of a degenerated particle set, by using for example the effective sample size as a measure of degeneracy [AMGC02]. This results in the standard particle filter algorithm with general proposal function and optional resampling in every step, stated in Algorithm 2.5.

Algorithm 2.5 Standard Particle Filter

function ParticleFilter($\hat{b}_{t-1} = \{s^i_{t-1}, w^i_{t-1}\}^{N_s}_{i=1}, a_{t-1}, o_t$)
  for $i = 1, ..., N_s$ **do**
    Propagate/draw particles ▷ prediction
      $s^i_t \sim q_t(s_t | s^i_{t-1}, o_t, a_{t-1})$
    Compute weight ▷ measurement update
      $\tilde{w}^i_t = w^i_{t-1} \frac{\mathcal{T}(s^i_t | s^i_{t-1}, a_{t-1}) \mathcal{Z}(o_t | a_{t-1}, s^i_t)}{q_t(s^i_t | s^i_{t-1}, o_t, a_{t-1})}$
  Normalize weights
    $w^i_t = \frac{\tilde{w}^i_t}{\sum_{j=1}^{N_s} \tilde{w}^i_t}, \qquad i = 1, ..., N_s$
  if Resampling necessary **then**
    b^*_t = Resample($\{s^i_t, w^i_t\}^{N_s}_{i=1}$)
  if Additional noise necessary **then**
    add noise to b^*_t
  return b^*_t

3. Solving Decision Making Problems with State Uncertainty

In section 2.2, a first algorithm which solves a POMDP exactly for its optimal policy over a finite horizon was introduced. This algorithm suffered enormously from the two curses of dimensionality and history and is therefore far from practical for larger problems.
Moreover, value iteration computes the value over the complete belief space, no matter whether a belief state is reachable by the agent or not. Even for attainable beliefs, some might only be attained with very small probability. This fact shows that for practical decision-making algorithms it is desired to focus on computing values and policies only for relevant belief states.
Various offline solvers try to realize this desire by using special heuristics in order to focus on computing policies only for the reachable belief. They additionally break the curse of history by approximating the value function with α-vectors only for a finite set of belief points or samples. Some of them will be discussed further in Section 3.1.
For very large problems, however, the approximation of the value function by a set of belief points might be too coarse, which would result in suboptimal policies. In order to circumvent this problem, a potentially better approach could be to use online planning algorithms. They avoid approximating the global value function over all beliefs, by computing only local policies for the current belief of the agent. That means, after every change or update of the agent's belief, the policy needs to be replanned. This results in an interleaved policy computation and execution phase for online planning agents (see Figure 3.1) with the advantage, that such an agent inherently plans only for the reachable belief.
The development of online POMDP algorithms has led to state-of-the-art solvers, which are applicable to automated driving problems. Section 3.2 will examine some of the most popular ones of those state-of-the-art online solvers.

3.1. Offline Solvers

Offline solvers compute an approximation of the optimal value function and store an appropriate representation in memory. An agent then executes a policy based on this precomputed approximate value function and on the current belief the agents finds itself in.
As mentioned in section 2.2 the belief space is a high-dimensional continuous space, which makes it impossible to store an exact representation. Therefore, early approaches focused on representing the approximate optimal value function by a finite set of belief points along with their values [Hau00]. In order to query the value at arbitrary belief points (not

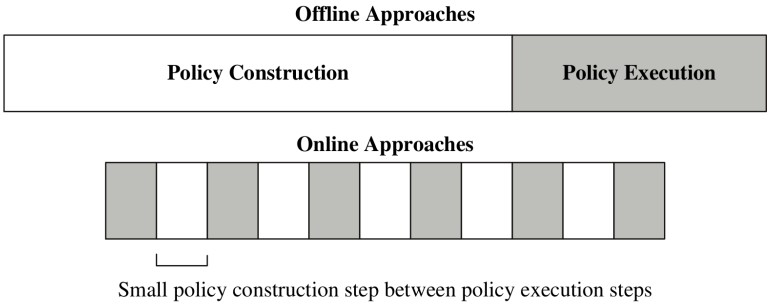


Figure 3.1.: Comparison between offline and online approaches (from [RPPCd08]).

necessarily in the set), an interpolation-extrapolation rule is used to specify the value as a function of the belief points in the set.
The methods suggested differ by how the belief points are distributed over the belief space. Most of them use a grid pattern over the belief space, which can have a fixed-resolution [Lov91] or a non-regular pattern [Hau97].
However, those interpolation-extrapolation schemes are rather computational complex. Thus, more recent approaches use α-vectors (see Eq. (2.20)) to represent the value function instead [SPK13]. For that reason the discussion of offline solvers will focus on the latter type of algorithms.
This section proceeds by first introducing some upper bounds on the optimal value function, which have been proven useful, before then some well-known offline solvers are presented.

3.1.1. Upper Bounds for the Value Function

Knowing upper bounds of the value functions can often be beneficial. Those bounds can be used as good initial guess for more accurate algorithms or help to eliminate suboptimal actions early, e.g. in branch and bound techniques [Koc15]. Finally, they can even be used as approximation of the optimal value function itself.
Two simple methods for computing upper bounds are QMDP [LCK95] and the Fast Informed Bound (FIB) [Hau00, Hau97].

QMDP

The QMDP method generalizes MDP value iteration to POMDPs by temporarily ignoring the observation model. The result is a policy defined over the belief space instead of the state space.
To compute the Q function for the POMDP, use of the underlying Q values of the MDP, consisting of transitions and rewards only, is made. After running the value iteration algorithm (Algorithm 2.1) the Q_{MDP} values can be obtained from equation (2.6). By averaging over a belief state b one can get the respecting $Q(b, a)$ value for the POMDP:

$$Q(b, a) = \sum_{s \in \mathcal{S}} b(s) Q_{MDP}(s, a) \tag{3.1}$$

The agent then takes the action a, which has the highest Q for a specific belief state b.
QMDP is based on the assumption, that any uncertainty in the agent's current belief state will be gone after the next action. As a result, a QMDP policy generally overestimates the true value of a belief state and will not take any information gathering actions [LCK95, TBF06, p.565].

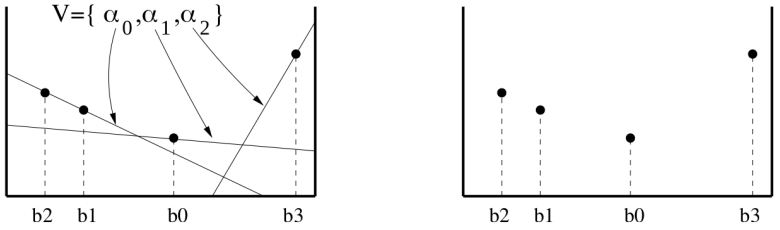


Figure 3.2.: Value function representation of PBVI (left) and grid-based approaches (right) (from [PGT03]).

Fast Informed Bound

In contrast to QMDP, the Fast Informed Bound method (FIB) tries to take the partial observability into account (at least to some degree). Therefore, FIB provides a tighter upper bound than QMDP (i.e. $V_T^{FIB}(b) \leq V_T^{QMDP}(b)$ for all b) [Hau00].
FIB makes use of the α-vector update, just as the exact value iteration for POMDPs, but approximates it, such that the exponential growth in the number of α-vectors is avoided. Instead, the set of α-vectors has always the size of $|\mathcal{A}|$ and can be computed in polynomial time. The update of the α-vector $\alpha_T^a(s)$ at horizon T associated with action a is described as follows:

$$\alpha_T^a(s) = \mathcal{R}(s,a) + \gamma \sum_{o \in \mathcal{O}} \max_{\alpha \in \Gamma_T} \sum_{s' \in \mathcal{S}} \mathcal{Z}(o|a,s')\mathcal{T}(s'|s,a)\alpha(s'), \tag{3.2}$$

where the observation model $\mathcal{Z}(o_{t+1}|a_t, s_t)$ is used.
The α-vectors α_0^a can be initialized to the α-vectors obtained by the value iteration algorithm (Algorithm 2.1) at convergence and equation (2.6), i.e. $\alpha_0^a(s) = Q_{MDP}(s,a)$. The value of a belief for the fast informed bound $V_T^{FIB}(b)$ can then be obtained by equation (2.20) [RPPCd08].

3.1.2. Point-Based Value Iteration

The Point-Based Value Iteration algorithm (PBVI) solves POMDPs approximately by estimating the optimal value function using strictly point-based updates [PGT06, PGT03]. The value function estimate is guaranteed to have bound on the error with respect to the exact solution and will improve over the runtime of the algorithm (provided it is initialized adequately), making the algorithm an anytime algorithm.
However, since PBVI is applicable only to problems with discrete action, observation and state spaces of order 10^3 states, it is not suitable for complex autonomous driving problems, but has been applied successfully to various robotic tasks [PGT06]. Moreover, this algorithm provides useful insights in the theory of approximate POMDP solving and introduced general concepts used also by other offline solvers, which makes it worth to explain PBVI more in detail.

Point-Based Value Backup

As mentioned before, PBVI only computes the value function exactly at a fixed set of belief points $B = \{b_0, b_1, ..., b_q\}$ and assumes that the value function at other belief points close to them in the set have the same action choice and similar values. Hence, the value function will contain at most one α-vector for each belief point and can be represented by the set $\Gamma_T = \{\alpha_0, \alpha_1, ..., \alpha_q\}$. This representation is shown in the left plot of Figure 3.2. One can observe, that the approximation with α-vectors preserves the piecewise linearity and convexity of the value function.

The operation all the point-based offline algorithms in this chapter have in common, is the point-based value backup operation, which is the approximation of one step of exact

value iteration for POMDPs (see Sec. 2.2).
While the Bellman update (2.18) and the α-vector backup in Algorithm 3.6 both compute the value at a belief point b, the α-vector backup additionally computes the gradient of the value function approximation making the α-vector a global approximation over the entire belief space rather than only a local approximation at b.
During backup, the α-vectors corresponding to the belief point set B in Γ_{T-1} are updated, such that the resulting set Γ_T now approximates the value function to the solution of the POMDP for horizon T. The process takes only polynomial time and is described in Algorithm 3.6. The size of the solution set Γ_T remains constant at every iteration. This is a big advantage over exact value iteration, where the solution set grows exponentially.

Algorithm 3.6 Point-based value backup

function BACKUP(B, Γ_{T-1})
  for each $a \in \mathcal{A}$ **do** ▷ Create intermediate set of α-vectors $\Gamma_T^{a,o}$
    for each $o \in \mathcal{O}$ **do**
      for each $\alpha_i \in \Gamma_{T-1}$ **do**
        $\alpha_i^{a,o}(s) = \gamma \sum_{s' \in \mathcal{S}} \mathcal{Z}(o|a,s')\mathcal{T}(s'|s,a)\alpha_i(s'), \forall s \in \mathcal{S}$
      $\Gamma_T^{a,o} = \cup_i \alpha_i^{a,o}$
  $\Gamma_T \leftarrow \emptyset$ ▷ Add only α-vectors maximizing the value to the solution set
  for each $b \in B$ **do**
    $\alpha_b = \arg\max_{a \in \mathcal{A}} \left[\sum_{s \in \mathcal{S}} \mathcal{R}(s,a)b(s) + \sum_{o \in \mathcal{O}} \max_{\alpha \in \Gamma_T^{a,o}} \boldsymbol{\alpha}^\top \boldsymbol{b} \right]$
    if $\alpha_b \notin \Gamma_T$ **then**
      $\Gamma_T \leftarrow \Gamma_T \cup \alpha_b$
  return Γ_T

PBVI Algorithm

The PBVI algorithm uses a series of point-based value backups alternately with a belief set selection or expansion step, where new belief points are added to the existing belief set B in order to improve the global value function approximation.
The main PBVI function is presented in Algorithm 3.7. It accepts an initial belief point set B_{Init}, an initial value Γ_0, the number of desired expansions N and the planning horizon T as input. A typical choice for B_{Init} is the initial belief b_0. The initial value Γ_0 is usually set to a low value, such as $\alpha_0(s) = \frac{\mathcal{R}_{min}}{1-\gamma}, \forall s \in \mathcal{S}$, where $\mathcal{R}_{min}$ is the minimum possible reward for any action a.[1] By doing this, it can be shown that the point-based solution always is a lower-bound on the exact value function [Lov91]. This is plausible since missing computing an α-vector for any belief point can only lower the value function, i.e. in Figure 3.2 missing the vector α_1 would lower the value function now consisting of α_0 and α_2 only.

Moreover, it is guaranteed that the error of the PBVI algorithm $\epsilon_T = \|V_T^B - V_T^*\|_\infty$ for any belief set B and any horizon T is bounded by

$$\epsilon_T \leq \frac{(\mathcal{R}_{max} - \mathcal{R}_{min})\delta_B}{(1-\gamma)^2}, \tag{3.3}$$

where δ_B is the density of a set of belief points B, which is defined to be the maximum distance from any belief in the simplex Δ to a belief point in set B, i.e. [PGT06]

$$\delta_B = \max_{b' \in \Delta} \min_{b \in B} \|b - b'\|_1. \tag{3.4}$$

[1] When an agent always executes a specific action regardless of the current belief, this is referred to as blind policy. Blind policies are usually used as lower bounds on the value function [Hau97, SS05, RPPCd08].

Algorithm 3.7 Point-based value iteration

```
1: function PBVI(B_Init, Γ_0, N, T)
2:     B ← B_Init
3:     Γ ← Γ_0
4:     for N expansions do
5:         for T iterations do
6:             Γ ← BACKUP(B, Γ)        ▷ approximate value function with belief set B
7:         B_new ← EXPLAND(B, Γ)        ▷ select new belief points to be added to B
8:         B ← B ∪ B_new
9:     return Γ
```

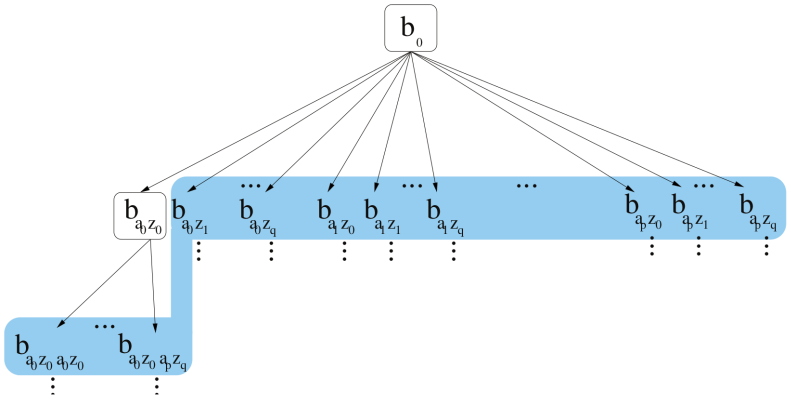


Figure 3.3.: The reachable belief (from [PGT06]).

Belief Point Selection

The bound defined in Equation (3.3) indicates that the belief point set B is crucial for the performance of PBVI. The goal is to keep B as small as possible, while at the same time still having a good approximation of the value function. Therefore, new belief points to be added to B must be selected carefully.
Pineau et al. examine different approaches for belief set expansion including uniform random selection over the entire belief simplex. However, the best performing selection strategy is Greedy Error Reduction (GER), which adds the candidate beliefs from the set of reachable beliefs that reduce most effectively the theoretic error bound (cf. Eq. (3.3)) [PGT06].
The reachable belief can be determined by simulating beliefs already in B one step forward in time using the belief update function $\tau(b, a, o)$. The result is a belief tree containing the immediate descendants of the belief points in B, denoted as $\overline{B}$ and shown in blue in Figure 3.3. After the set $\overline{B}$ is determined, the GER chooses those beliefs with high reachability probability and large error bound.

3.1.3. Heuristic Search Value Iteration

Another point-based POMDP algorithm, which can be applied to larger problems with $\sim 10^5$ states, is Heuristic Search Value Iteration (HSVI) [SS04, SS05]. Similar to PBVI, HSVI also has theoretical convergence guarantees, anytime performance and takes reachability in the belief space into account.

The differences between these two algorithms lie in both, how the new belief points are selected, and how updates of the value function are ordered [PGT06]. Whereas PBVI performs a batch of belief set expansions, followed by a batch of backup operations over

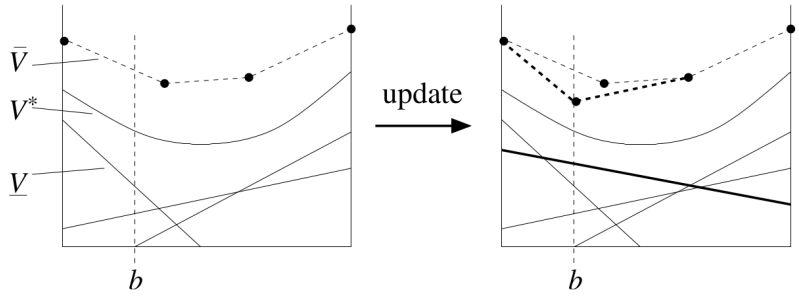


Figure 3.4.: A local update at b (from [SS04]).

all points (see Algorithm 3.7), HSVI updates for each expanded belief only the value of the direct ancestors (parents, grand-parents, etc., all the way back to the initial belief). This is done in multiple exploration trials, where the algorithm repeatedly visits new belief points by choosing actions and observations according to a novel forward search heuristic until a specific termination criterion is met. When the trial has terminated, HSVI updates the beliefs encountered during the trial in reverse order (i.e. the last visited belief is updated first).

Value Function Representation and Local Updates

Unlike PBVI, HSVI stores an upper bound $\overline{V}(b)$ and a lower bound $\underline{V}(b)$ on the optimal value function V^*. The lower bound is represented by a set of α-vectors $\Gamma_{\underline{V}}$ — similar to PBVI. Therefore, the same backup operation is used for local updates (see Algorithm 3.6). The only difference is, that the backup is only applied to a single belief point b instead of a full set of belief points B.
For the upper bound, a vector set cannot be used, since updating the set (i.e. adding another α-vector) would not improve (reduce) the upper bound in the neighborhood of the local update due to the max-operator [SPK13, p. 8]. Therefore, the upper bound is represented by the finite point set $\Upsilon_{\overline{V}}$ consisting of belief/value points $(b_i,\overline{v}_i)$.
To compute the value $\overline{v}$ at an arbitrary belief point b, the projection of b onto the convex hull of $\Upsilon_{\overline{V}}$ must be calculated. This can be done exactly by solving a linear program (LP) or approximately using the so-called saw-tooth (or jig-saw) approximation [Hau00, SS05]. Updates of $\Upsilon_{\overline{V}}$ are performed by adding a new point to the set using the Bellman operator from equation (2.18).
In HSVI a local update of the value function at b means applying an update on both the lower and upper bound. That means a new α-vector will be added to $\Gamma_{\underline{V}}$ during backup of the lower bound and a new belief/value point is inserted in $\Upsilon_{\overline{V}}$ (see Fig. 3.4).

Forward Search Heuristics

The gap between the bounds at a specific belief can be considered as the uncertainty at that belief. The higher the difference, the more uncertain the algorithm is about the optimal value at that belief point. The goal of the search heuristics is to reduce this uncertainty and hence the regret at the initial belief b_0, which is the difference in value between the theoretical optimal policy π^* and the policy π returned by the algorithm:

$$\text{regret}(\pi, b) = V^{\pi^*}(b) - V^{\pi}(b) \tag{3.5}$$

More formally, the goal is to find a policy whose regret at the initial belief b_0 is bounded by a convergence parameter ϵ (i.e. $\text{regret}(\pi, b_0) \leq \epsilon$).
To achieve this, HSVI greedily selects successors that maximize the so-called excess uncertainty of a belief:

$$\text{excess}(b, t) = \left(\overline{V}(b) - \underline{V}(b)\right) - \frac{\epsilon}{\gamma^t}, \tag{3.6}$$

where γ is the discount factor and t is the depth of b (i.e. number of actions from b_0 to b). The excess is derived from the termination criterion of the function EXPLORE, which is true, if the discounted gap between the bounds at belief b is less than ϵ,
i.e. $\gamma^t \left(\overline{V}(b) - \underline{V}(b)\right) \leq \epsilon$.

To reach the termination criterion, i.e. ensure convergence, the action with the greatest upper bound is chosen:

$$a^* = \arg\max_a Q^{\overline{V}}(b, a) \tag{3.7}$$

This choice makes sure, that the eventual sub-optimality of the selected action may be discovered when the a^* upper bound drops below the upper bound of another action. Then the other action is selected, since its optimal value might be higher. By selecting actions according to the highest lower bound, sub-optimality might never be discovered, since future updates can only increase the lower bound and no other actions will be taken.

After action selection, the heuristic chooses the observation that most contributes to the excess uncertainty at the current belief b:

$$o^* = \arg\max_o \left[\Pr(o|b, a^*) \operatorname{excess}(\tau(b, a^*, o), t + 1)\right] \tag{3.8}$$

If the excess uncertainty at the next belief node gets negative, the node satisfies the termination condition and is called finished. Then, the exploration trial ends by updating the lower and upper bounds of all visited nodes up to the root node in reverse order.

In this way, HSVI may visit the same belief point in many trials or even multiple times during the same trial and adds a new α-vector for each visit. This is opposed to PBVI, which stores one vector per belief point $b \in B$ at most. However, the advantage of HSVI is that there is no need to explicitly store a belief point set B as in PBVI, since the search always starts at the initial belief b_0 and discovers the beliefs to be updated during the trial [SPK13, p.19]. Dominated elements that occur during the search in the lower bound vector set and upper bound belief-value point set are pruned periodically [SS04].

HSVI Algorithm

The previously explained search heuristic is used in the EXPLORE function of Algorithm 3.8. Starting from an initial belief b_0 and the convergence parameter ϵ, the HSVI algorithm first initializes the upper and lower bounds and then repeatedly executes exploration trials until the convergence criterion at the initial belief b_0 is met.

The lower bound is initialized using the blind policy method analog to PBVI (see Sec. 3.1.2) and for the upper bound the QMDP method or in the improved version of HSVI the Fast Informed Bound can be used (see Sec. 3.1.1).

After termination of HSVI, the returned lower bound is used to determine the policy (cf. Eq. (2.19)). So, the upper bound is only used for action selection during exploration trials. Theoretically one can use the gap between the bounds to decide whether the algorithm has converged, but experiments have shown that for large domains the gap closes very slowly and remains large, even when the lower bound seems to converge [SPK13, p.47].

Therefore, usually HSVI is used in an anytime fashion, where the quality of the solution improves over runtime. This can be done by making the loop in function HSVI an infinite loop and simply setting ϵ at the top-level call to EXPLORE to a slightly smaller value than the current uncertainty at b_0 in each iteration, i.e. $\epsilon = \zeta(\overline{V}(b_0) - \underline{V}(b_0))$, where $\zeta < 1$ is a scalar parameter [SS04].

Algorithm 3.8 Heuristic search value iteration

function HSVI(b_0, ϵ)
  Initialize $\underline{V}$ and $\overline{V}$
  while $\overline{V}(b_0) - \underline{V}(b_0) > \epsilon$ **do**
    EXPLORE(b_0, ϵ, 0)
  return $\Gamma_{\underline{V}}$ ▷ lower bound as approximation of the value function

function EXPLORE(b, ϵ, t) ▷ follows a single path down the search
  if $\overline{V}(b) - \underline{V}(b) > \epsilon\gamma^{-t}$ **then**
    $a^* \leftarrow \arg\max_a Q^{\overline{V}}(b, a)$ ▷ select action according to $\overline{V}$
    $o^* \leftarrow \arg\max_o [\Pr(o|b, a)\text{excess}(\tau(b, a^*, o), t + 1)]$
      ▷ select observation that maximizes the gap between bounds
    EXPLORE($\tau(b, a^*, o^*)$, ϵ, $t + 1$)
  ▷ After the recursion, update both bounds on the way back up to the initial belief
    $\Gamma_{\underline{V}} \leftarrow$ BACKUP($\{b\}$, $\Gamma_{\underline{V}}$) ▷ see Algorithm 3.6
    $\Upsilon_{\overline{V}} \leftarrow \Upsilon_{\overline{V}} \cup (b, H\overline{V}(b))$

3.1.4. Successive Approximations of the Reachable Space under Optimal Policies

The point-based algorithm Successive Approximations of the Reachable Space under Optimal Policies (SARSOP) is related to HSVI, since SARSOP reuses many core components of HSVI. The main contribution of SARSOP is the improved sampling strategy that tries to sample new belief points from or close to $\mathfrak{R}^*(b_0)$, the subset of beliefs reachable only by following an optimal action sequence. Since this optimal action sequence, which would correspond to the optimal policy, is not known prior to solving the POMDP, SARSOP tries to approximate $\mathfrak{R}^*(b_0)$ [HK08].

In the experiments conducted by Kurniawati et al. SARSOP has outperformed the improved version of HSVI [SS05] in five out of six robotic tasks in terms of speed with similar or even better reward levels making SARSOP one of the fastest offline POMDP solvers to date. The reason for this performance is that SARSOP avoids unnecessary sampling in $\mathfrak{R}(b_0)\backslash\mathfrak{R}^*(b_0)$, as $\mathfrak{R}^*(b_0)$ is usually much smaller than $\mathfrak{R}(b_0)$ [HK08].

SARSOP Algorithm

SARSOP retains the sampled belief points in a tree structure $T_{\mathfrak{R}}$ with the initial belief b_0 as root, which is different from the point set representation used in HSVI. New points are added to the tree in the SAMPLE function by choosing actions and observations according to HSVI's search heuristics and using the belief update function $\tau(b, a, o)$. In this way, the tree only consists of reachable beliefs.
Analog to HSVI, SARSOP also keeps the same upper and lower bound representations (sawtooth approximation for upper and α-vector set approximation of the lower bound) on the optimal value function. Thus, the bounds are initialized in the same way as in HSVI.

The main loop of SARSOP iterates over three main functions SAMPLE, BACKUP and PRUNE until the termination condition is fulfilled, which can be a fixed gap size between the upper and lower bound or a time limit similar to HSVI (see Algorithm 3.9).
The BACKUP operation is the standard α-vector backup as used in other point-based algorithms (e.g. PBVI and HSVI).

Sampling

The SAMPLE function in SARSOP corresponds to the EXPLORE function from HSVI. In fact the forward search heuristics implemented in EXPLORE builds the core of the SAMPLE

Algorithm 3.9 SARSOP

```
1: function SARSOP(b_0)
2:     Initialize V_ and V‾
3:     Insert belief b_0 as root of tree T_R
4:     while termination condition not satisfied do
5:         SAMPLE(T_R, Γ_V_)
6:         Choose nodes from T_R
7:         for each chosen node b do
8:             BACKUP({b}, Γ_V_)                ▷ see Algorithm 3.6
9:         PRUNE(T_R, Γ_V_)
10:    return Γ_V_        ▷ lower bound as approximation of the value function
```

function. Kurniawati et al. argue that the action and observation selection strategy together with the termination condition of the trial controls the sampling distribution. Hence, to focus more on sampling near $\mathfrak{R}^*(b_0)$ and minimize sampling from $\mathfrak{R}(b_0)\backslash\mathfrak{R}^*(b_0)$ they extend HSVI's termination condition by two new concepts: *Selective deep sampling* and the *gap termination criterion*.

Selective deep sampling uses a learning-enhanced prediction $\hat{V}$ of the possible optimal value $V^*(b)$ at node b and ensures that a trial is continued even if the HSVI termination criterion ($\overline{V}(b) - \underline{V}(b) \leq \epsilon\gamma^{-t}$) at b is met, if $\hat{V}$ is likely to improve the lower bound at the root b_0.

In contrast to selective deep sampling, the gap termination criterion focuses on the early termination of trials to avoid sampling in regions that are unlikely to be in $\mathfrak{R}^*(b_0)$ by leveraging the tree data structure $T_\mathfrak{R}$ in SARSOP. While HSVI runs trials in EXPLORE recursively until its termination condition is met (this corresponds in SARSOP to following a path in the tree from the root to a leaf), SARSOP can check other paths already in the tree sharing the same action-observation-history wether the termination condition is satisfied and terminate the trial early, even if the current node b does not have a target gap size less than $\epsilon\gamma^{-t}$.

For a detailed pseudocode of SAMPLE the reader is referred to [HK08].

Pruning

SARSOP also introduces a more aggressive pruning strategy keeping the set $\Gamma_{\underline{V}}$ of α-vectors considerably smaller than that of other point-based solvers. Existing pruning strategies usually remove an α-vector from $\Gamma_{\underline{V}}$, if it is dominated by others over the entire belief space $\mathcal{B}$, whereas SARSOP already tries to prune an α-vector if it is dominated over $\mathfrak{R}^*(b_0)$ only.
As $\mathfrak{R}^*(b_0)$ is not known in advance, the set B of belief points contained in $T_\mathfrak{R}$ is used as an approximation. To keep B as small as possible, any subtree of taking action a at node b in $T_\mathfrak{R}$ can be pruned, if $\overline{Q}(b,a) < \underline{Q}(b,a')$, since an optimal policy never takes action a in b.

Then with the pruned set B, an α-vector can be pruned if it is dominated by others in a δ-neighbourhood of every point in B. This *δ-dominance* is defined by Kurniawati et al.: α_1 dominates α_2 at a belief point b if $\alpha_1 \cdot b' \geq \alpha_2 \cdot b'$ at every point b' whose distance to b is less than δ, for some fixed constant δ [HK08].
δ-dominance can be evaluated quickly by computing the distance d from b to the intersection of the hyperplanes represented by α_1 and α_2 and making sure that $d \geq \delta$.

All in all, the improved sampling and pruning strategies increase the computational efficiency of SARSOP and explain the performance improvement compared to HSVI.

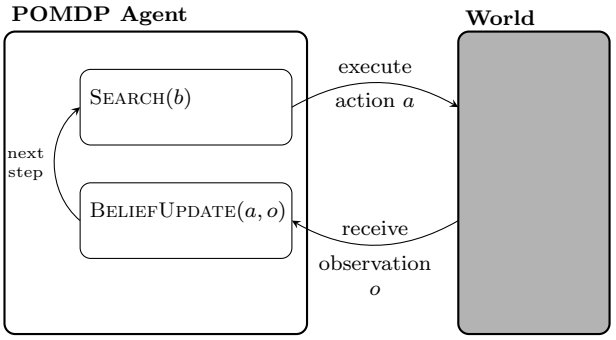


Figure 3.5.: Schematic of an online POMDP solving agent acting in a world.

3.2. State of the Art Online Solvers

In the previous section, some of the most important state of the art offline POMDP solvers were presented. They all compute the optimal value function and hence the optimal policy offline before policy execution. Because of this fact, they were all limited to discrete state, action and observation spaces of reasonable size.
Online solvers on the contrary determine the optimal policy solely by planning from the current belief state. That means the policy obtained by one planning step is only valid for a single belief state. After execution of one action of the computed policy the world state and hence also the belief state of the agent will change, such that the previous policy is not valid anymore. Therefore, replanning based on the updated belief is necessary.
In general the belief update is performed using filtering methods from Section 2.3. As a result of this planning and belief update cycle as depicted in Figure 3.5, the solver never stores a representation of the full optimal policy or value function in memory and plans only in the reachable belief space, which is often much smaller than the full belief space. This fact makes online solvers even applicable to problems with continuous spaces like for example automated driving problems.

The most successful online algorithms use tree search methods for calculating the current optimal policy. For that reason in the following a short introduction to Monte Carlo tree search methods is given, before it will be explained how these methods can be applied to POMDP problems.

3.2.1. Monte-Carlo Tree Search Methods

Monte Carlo tree search (MCTS) received great attention due to its success in the challenging problem of computer Go and has been applied to sequential decision-making and planning problems, which are modelled as MDPs or POMDPs, too [BPW+12].

MCTS incrementally constructs a search tree branching in different actions and states until a given time limit or tree size is reached. This fact makes MCTS an anytime algorithm, which means, that more computing resources increase the performance of the algorithm.
The basic steps of MCTS, which can be grouped in a tree and a default or rollout policy, are depicted in Figure 3.6. During execution of the tree policy the existing search tree is traversed from the root to its leafs by selecting existing nodes, while making a trade-off between exploring new nodes, which have not been visited often and exploiting nodes, which are already known to be promising. Upon arrival at a leaf node, a new node is added and the default/rollout policy is executed to estimate its value. A simple rollout policy could be based on random action selection for example. After that, the values of all previously visited nodes are updated in the back propagation step.

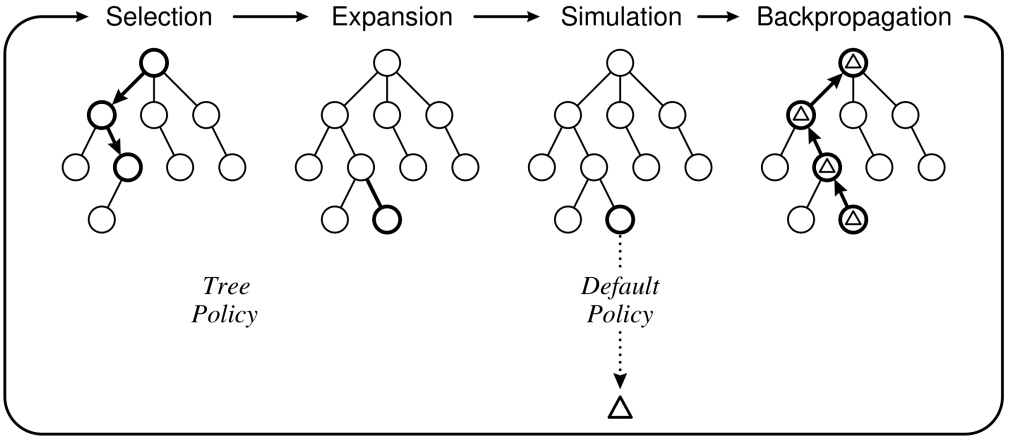


Figure 3.6.: One step of the general MCTS approach (from [BPW+12]).

Upper Confidence Bounds for Trees Algorithm

The most popular algorithm following the concept from Figure 3.6 is the Upper Confidence Bound for Trees (UCT) algorithm [KS06]. It can be used to solve MDP problems with large (discrete) state spaces, where there is one node for each state $s \in \mathcal{S}$ in the tree and an action $a \in \mathcal{A}$ is assigned to each edge between the nodes. Every node contains a value $Q(s,a)$ and a visitation count $N(s,a)$ for each action a and an overall count $N(s) = \sum_a N(s,a)$. The value $Q(s,a)$ is estimated by the mean return of all simulations in which action a was selected from s. At last, the transitions from one node to another (as for e.g. in the Expansion step) are defined by the transition model $\mathcal{T}$.
The special thing about UCT is, that during execution of the tree policy the nodes are selected according to the UCB1 algorithm introduced by Auer et al. [ACBF02]. That means the action selection is treated as multi-armed bandit problem, in which one needs to choose from the available actions (corresponding to the arms) at the current node in order to maximize the cumulative reward by taking the optimal action.

Auer et al. define a K-armed bandit problem as random variables $X_{i,n}$ for $1 \leq i \leq K$ and $n \geq 1$, where each i is the index of a gambling machine (i.e., an "arm" of a bandit). Successive plays of a machine i yield rewards $X_{i,1}, X_{i,2}, ...$ which are independent and identically distributed according to an unknown law with unknown expectation μ_i. Independence also holds for rewards across arms (actions); i.e., $X_{i,s}$ and $X_{i,t}$ are independent (and usually not identically distributed) for each $1 \leq i < j \leq K$ and each $s, t \geq 1$.

The UCB1 algorithm is a solution to this problem and can be seen as policy that determines which action to choose at the current node. This policy dictates to choose the "arm" or machine j that maximizes the upper confidence bound:

$$\text{UCB1} = \overline{X}_j + 2C_p \sqrt{\frac{2 \ln n}{n_j}}, \tag{3.9}$$

where $\overline{X}_j$ is the average reward obtained from machine j, n_j is the number of times machine j has been played so far, n is the overall number of plays done so far and $C_p > 0$ is an exploration constant to prefer exploitative (C_p small) or explorative (C_p large) behavior.

Auer et al. showed that, if UCB1 is used to select the "arms", the regret (i.e., the expected loss due to not playing the best machine) grows only logarithmically[2] with n for arbitrary reward distributions with bounded support[3].

[2]Logarithmic growth is very good, since it is a very slow growth!
[3]Rewards need to be finite.

In the MDP setting of UCT the machines or "arms" correspond to the available actions at the current node, the random variables $X_{i,n}$ correspond to the estimated Q-value of action i in the n-th visit of the current node, n_j is the number of times action j has been selected and hence $\overline{X}_j$ corresponds to the average (or estimated) Q-value of action j after n visits of the current node.

Since $n_j = 0$ leads to UCB1 $= \infty$, it must be ensured that all machines are played once at first. This implies that MCTS algorithms using UCB1 without any further adaptions such as for example UCT are only applicable to problems with finite discrete action spaces.

3.2.2. Partially Observable Monte-Carlo Planning

The UCT algorithm explained above can solve problems with fully observable states, i.e. MDPs. Silver and Veness extended this algorithm to partially observable domains by using a tree search of histories (see Eq. (2.8)) instead of states [SV10]. The new algorithm Partially Observable Monte Carlo Planning (POMCP) consists of a UCT tree search that selects actions at each time-step and a particle filter with rejection that updates the agent's belief state similar to Figure 3.5. It hereby uses the same set of Monte-Carlo simulations for both tree search and belief state updates.
Silver and Veness demonstrated that POMCP successfully solves POMDP problems with approximately 10^{18} and 10^{56} states, which is by many orders of magnitude more than any other online or offline solver (compare for e.g. HSVI2 $\sim 10^5$) was capable of at that time. POMCP achieved this massive improvement, because it successfully breaks the curse of dimensionality and history by sampling (see Section 2.2). Specifically, it breaks the curse of dimensionality, by sampling state transitions and focuses quickly on the promising ones instead of considering all possible state transitions and the curse of history, by sampling histories using a black box simulator.

Belief State Update

This simulator $\mathcal{G}$ is used as a generative model of the POMDP to sample successor states, observations and rewards, given a state and action:

$$(s_{t+1}, o_{t+1}, r_{t+1}) \sim \mathcal{G}(s_t, a_t) \tag{3.10}$$

$\mathcal{G}$ is called black box simulator, since it is used to update the value function without looking explicitly at the internal models of the dynamics of the POMDP problem (i.e. transition model $\mathcal{T}$, observation model $\mathcal{Z}$ and reward model $\mathcal{R}$). Therefore, the underlying distributions of $\mathcal{T}$ and $\mathcal{Z}$ do not need to be known explicitly; it just must be possible to generate samples from them and this is what the generative model $\mathcal{G}$ does.

As a result the standard particle filter from Algorithm 2.5 cannot be used, since it requires evaluating transition and observation model at specific points for weight calculation. Instead, POMCP uses an unweighted particle filter with rejection (Algorithm 3.10) for belief updating.

This particle filter makes use of the generative model and adds a sample state s' to the new belief b_t, if the sample observation o' matches the real observation o_t. The problem with this method is that it might be necessary to draw many samples until the observations match. Moreover, due to the observation matching this particle filter cannot be applied to problems with continuous observation spaces, since the probability of sampling the same observation twice is zero.

Algorithm 3.10 Particle filter with rejection

```
1: function UpdateBelief(b_{t-1}, a_{t-1}, o_t)
2:     b_t ← ∅
3:     for i = 1, ..., |b_{t-1}| do
4:         s ← random state in b_{t-1}
5:         repeat
6:             (s', o', r) ~ G(s, a_{t-1})
7:         until o' = o_t
8:         add s' to b_t
9:     return b_t
```

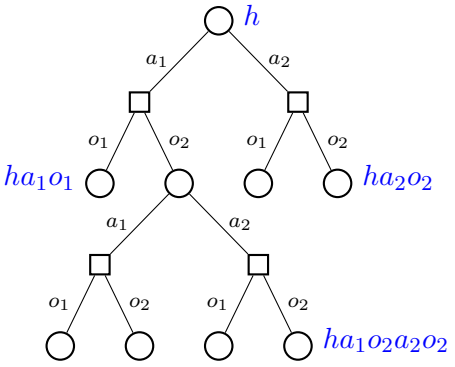


Figure 3.7.: A possible search tree of POMCP in an environment with 2 actions and 2 observations. Q-nodes are depicted as squares and V-nodes as circles.

POMCP Tree Search

During tree search, POMCP uses the same generative model $\mathcal{G}$ as for belief updates. Since POMCP plans with histories instead of states, the tree branches not only in different actions that can be taken from a node, but also in different observations that could possibly be received after taking an action. This results in a tree with two different types of nodes: Q-nodes $T(ha)$ representing histories ending with an action ha and V-nodes $T(hao)$ representing histories ending with an observation hao (see Figure 3.7). Each node contains its visitation count $N(h)$ and value $V(hao)$ or Q-value $Q(ha)$ in case of Q-nodes. V-nodes additionally store the belief $B(h)$ as an unweighted particle set corresponding to their history.

The partially observable tree search algorithm can then be divided into the functions Search(), Simulate() and Rollout() as outlined in Algorithm 3.11. The Search-function contains the main loop of the algorithm, which iteratively samples a state from the belief $B(h_t)$ representing the current history h_t and calls Simulate() until a time limit is reached.
Simulate() basically combines the steps *Selection, Expansion* and *Backpropagation* of the general MCTS approach. It recursively traverses the search tree from the root to a leaf by selecting actions (i.e. Q-nodes) according to UCB1 analog to the UCT algorithm and observations (i.e. V-nodes) according to the generated observation such that the generated observation matches the observation edge leading to a V-node at the next deeper tree level. The recursion stops as soon as a history hao is encountered for which no observation o exists in the search tree, yet. Then, a new V-node is initialized with N_{init} and Q_{init} (default values are 0) and its value V is estimated by the Rollout-function, which uses a history based rollout policy $\pi(h, \cdot)$ to select actions. If the tree depth d reaches its maximum d_{max}, no further node is added and the values of previously visited nodes are updated based on

existing nodes.
As a last step, this value is back propagated through all visited nodes in reverse order to compute the cumulative discounted rewards. Additionally, the visitation counts are updated and at each V-node the generated successor state s' is added to the belief $B(h)$.

When the time limit is reached, the agent selects the action a_t with the greatest value, receives an observation o_t from the world and updates its belief $B(h_{t+1} = h_t a_t o_t)$ accordingly. At this point, the node $T(h_t a_t o_t)$ becomes the new root of the tree and the remainder of the tree is pruned, as all other histories are now impossible.

Algorithm 3.11 Partially Observable Monte-Carlo Planning (from [SV10])

```
1:  function SEARCH(h)
2:      repeat
3:          s ~ B(h)
4:          SIMULATE(s, h, d_max)
5:      until timeout
6:      return arg max_a Q(ha)

7:  function ROLLOUT(s, h, d)
8:      if d = 0 then
9:          return 0
10:     a ~ π_rollout(h, ·)
11:     (s', o, r) ~ G(s, a)
12:     return r + γ ROLLOUT(s', hao, d−1)

13: function SIMULATE(s, h, d)
14:     if d = 0 then
15:         return 0
16:     if h ∉ T then
17:         for each a ∈ A do
18:             N(ha) ← N_init
19:             Q(ha) ← Q_init
20:         return ROLLOUT(s, h, d)
21:     a ← arg max_a' Q(ha') + c sqrt(log N(h) / N(ha'))
22:     (s', o, r) ~ G(s, a)
23:     R ← r + γ SIMULATE(s', hao, d − 1)
24:     B(h) ← B(h) ∪ {s}
25:     N(h) ← N(h) + 1
26:     N(ha) ← N(ha) + 1
27:     Q(ha) ← Q(ha) + (R − Q(ha)) / N(ha)
28:     return R
```

3.2.3. Partially Observable Monte-Carlo Planning in Continuous Spaces

The POMCP algorithm presented in the previous section builds a search tree from many simulated state trajectories and estimates the Q-value of an action node by averaging over the rewards from all state trajectories passing through that node. This procedure works very well for large discrete state, action and observation spaces, where large means many possible states, actions and observations.

Even though the restriction to discrete action spaces is often reasonable even in real world problems, since the number of possible actions of an agent are often limited naturally or can be easily discretized, discrete states or observations pose stronger restrictions to formulating real world problems, because the real world is continuous by nature. Consider for example the problem of automated driving. It is easy to discretize the acceleration values or steering angles of the ego-vehicle, but it is difficult to discretize the observation obtained from the environment, since other vehicles or pedestrians move continuously. Hence, it is desired to solve POMDP problems with continuous spaces.

Unfortunately, applying the standard POMCP algorithm to continuous problems is not possible. The reason for this is that the probability of sampling the same real number twice from a continuous distribution is zero. In case of generating observation nodes in the search tree, this results in a new observation being generated in every simulation. The consequence is a shallow search tree that does not extend below the first layer of observations (see Figure 3.8).

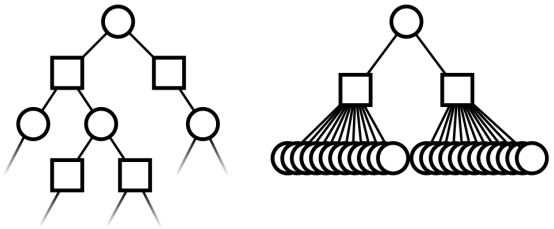


Figure 3.8.: POMCP tree for discrete POMDPs (left), and for POMDPs with continuous observation space (right). Due to continuous observation space, each simulation creates a new observation and the tree cannot extend deeper [SK18].

To resolve this issue Sunberg and Kochenderfer propose to apply *double progressive widening* to POMDP search trees to accommodate for continuous action and observation spaces and to use the observation model $\mathcal{Z}$ in combination with weighted particle filters (see section 2.3.3) for continuous state spaces [SK18]. One of their new algorithms making these adaptions to POMCP is called partially observable Monte Carlo planning with observation widening (POMCPOW). The requirement for the application of this algorithm is that in addition to the generative model as in POMCP, one needs to be able to evaluate the observation model $\mathcal{Z}$ pointwise. Hence, it must be known explicitly.

Double Progressive Widening in MCTS

Double progressive widening was first used to extend the UCT algorithm to continuous action and state spaces [CHS+11]. For continuous problems — similar to POMCP — UCT also produced only shallow search trees, since the probability of sampling the same action or state twice in continuous spaces is zero. For action selection with UCB1 this results in a selection strategy that will always explore newly generated actions, because for UCB1 to work every possible action must be chosen once first. In continuous action spaces there are infinitely many actions, and hence UCB1 will never stop exploring.

Progressive widening prevents the tree from becoming too wide and shallow, since it artificially limits the number of children of each node to kN^α, where N is the number of times a node has been visited and $k > 0$ and $\alpha \in (0, 1)$ are hyper-parameters controlling the tree growth. If the number of children is greater than kN^α, then, instead of adding a new node to the tree, a previously generated child node is selected with probability proportional to the number of times it has been previously visited, or if the child nodes are action nodes according to UCB1.

The term *double* progressive widening was used to indicate, that progressive widening is applied to both the state and action space, in case of UCT.

POMCPOW Algorithm

The newly proposed algorithm POMCPOW shares the same structure as POMCP and uses the same ROLLOUT function. The difference however lies in the SIMULATE function outlined in Algorithm 3.12. The notation closely follows the paper from Sunberg and Kochenderfer and is consistent with POMCP (Algorithm 3.11) to ease comparison [SK18]. To explain the extensions of POMCPOW, some extra notations are used: $C(h)$ is a list of children of a node in the tree indicated by its history h. The belief of a node is denoted as list of states $B(h)$ (in contrast to POMCP, where it was a set) with $W(h)$ a list of weights corresponding to those states. Finally, $M(h)$ is a counter for how often a history has been generated by the model.

To generate or select the next action in SIMULATE(), POMCPOW uses the function ACTIONPROGWIDEN, which controls progressive widening of the action space and selects

actions according to UCB1. Then, a successor state s', an observation o and the corresponding reward r is generated by the generative model $\mathcal{G}$ analog to POMCP. In the next step, progressive widening of the observation space determines whether the new observation o can be added to the tree or an existing observation will be selected. POMCPOW now appends the particle s' weighted by the observation model $\mathcal{Z}$ to the lists $B(hao)$ and $W(hao)$.
If progressive widening allowed a new observation node to be added as child to the tree, the recursion stops and the value of this node is estimated with ROLLOUT(), otherwise the state s' is replaced by a new one sampled from $B(hao)$, the reward is adapted to the new sampled state s' and SIMULATE() continues one level deeper in the tree. At last, the backpropagation step is analog to POMCP.

The result of this algorithm is a belief tree with weighted particles as belief representation in the V-nodes. Over time, the number of particles in the V-nodes increases the more simulations are added and beliefs which are more likely to be reached by the optimal policy will have more particles in the end [SK18].

Algorithm 3.12 POMCPOW (from [SK18])

function SEARCH(b)
  repeat
    $s \leftarrow$ sample from b
    SIMULATE(s, b, d_{max})
  until timeout
  return $\arg\max_a Q(ba)$

function ACTIONPROGWIDEN(h)
  if $|C(h)| \leq k_a N(h)^{\alpha_a}$ **then**
    $a \leftarrow$ NEXTACTION(h)
    $C(h) \leftarrow C(h) \cup \{a\}$
  return $\arg\max_{a \in C(h)} Q(ha) + c\sqrt{\frac{\log N(h)}{N(ha)}}$

function SIMULATE(s, h, d)
  if $d = 0$ **then**
    return 0
  $a \leftarrow$ ACTIONPROGWIDEN(h)
  $(s', o, r) \sim \mathcal{G}(s, a)$
  if $|C(ha)| \leq k_o N(ha)^{\alpha_o}$ **then**
    $M(hao) \leftarrow M(hao) + 1$
  else
    $o \leftarrow$ select $o \in C(ha)$ w.p. $\frac{M(hao)}{\sum_o M(hao)}$
  append s' to $B(hao)$
  append $\mathcal{Z}(o|s, a, s')$ to $W(hao)$
  if $o \notin C(ha)$ **then** ▷ new node
    $C(ha) \leftarrow C(ha) \cup \{o\}$
    $R \leftarrow r + \gamma \cdot$ROLLOUT($s', hao, d-1$)
  else
    $s' \leftarrow$ select $B(hao)[i]$ w.p. $\frac{W(hao)[i]}{\sum_{j=1}^{m} W(hao)[j]}$
    $r \leftarrow \mathcal{R}(s, a, s')$
    $R \leftarrow r + \gamma \cdot$SIMULATE($s', hao, d-1$)
  $N(h) \leftarrow N(h) + 1$
  $N(ha) \leftarrow N(ha) + 1$
  $Q(ha) \leftarrow Q(ha) + \frac{R - Q(ha)}{N(ha)}$
  return R

4. Information Particle Filter Tree Algorithm

The Information Particle Filter Tree (IPFT) algorithm will be used in this thesis to solve the driving scenario POMDP formulations in the next section. It is similar to POMCPOW in the sense that it is an online, Monte-Carlo tree search based algorithm for solving POMDPs with continuous state, action and observation spaces, but it differs in the way of how the belief particles are propagated through the search tree [FT20, Fis19].
POMCPOW simulates single particles by using a generative model $(s', o, r) \sim \mathcal{G}(s, a)$, which is based on state trajectories. Hence, it internally uses the transition model $\mathcal{T}$, observation model $\mathcal{Z}$ and reward model $\mathcal{R}$ directly to generate the tuple (s', o, r).
In contrast, IPFT uses a generative model $\hat{b}'_m \sim \mathcal{G}_{\mathrm{PF}(m)}(\hat{b}_m, a, o')$ which is based on belief trajectories. $\mathcal{G}_{\mathrm{PF}(m)}$ can be considered as approximation of the belief update function $\tau(b_{t-1}, a_{t-1}, o_t)$ as introduced for finite discrete state spaces in Section 2.2 (cf. Eq. (2.12)). For continuous spaces, however, the belief update τ is only tractable, if strong assumptions about the belief distributions are made (Sec. 2.3.1). Therefore, $\mathcal{G}_{\mathrm{PF}(m)}$ uses a particle filter (see Sec. 2.3.3) in combination with a sampled observation o' to generate the posterior belief $\hat{b}'_m$.

This method of solving the belief MDP by simulating approximate belief trajectories with particle filters was introduced by Sunberg and Kochenderfer as particle filter trees with double progressive widening (PFT-DPW) algorithm [SK18]. IPFT extends this algorithm by integrating information measures in the reward computation in order to guide the search in the direction of more informative beliefs.

How IPFT integrates information measures in PFT-DPW is discussed in the beginning of this chapter. After that the algorithm will be explained and some implementation details are given.

4.1. Belief Dependent Reward Calculation

As IPFT simulations are based on beliefs, rather than states, each observation node in the tree contains a small particle set of fixed size m as approximation of the true belief: $\hat{b}_m \approx b$. Therefore, to solve the belief MDP, it is required to plan with belief-based rewards $\rho(b, a)$ instead of state-based rewards $\mathcal{R}(s, a)$.

This reward function of the belief MDP is derived as expectation of $\mathcal{R}(s,a)$ under the belief $b(s)$ for a continuous state space $\mathbb{S}$:

$$\rho(b,a) = \int_{\mathbb{S}} \mathcal{R}(s,a)b(s)\,\mathrm{d}s \tag{4.1}$$

The discrete version of this equation was stated in Equation (2.14), where the integral becomes a sum.

The PFT-DPW algorithm, as well as all offline algorithms from Section 3.1 rely on this specific definition of the rewards to compute the optimal value function $V^*(b)$ for the belief MDP. Araya-López et al. propose ρPOMDPs as an extension to POMDPs, where $\rho(b,a)$ must not follow the definition in (4.1), but can be an arbitrary function over the belief [ABTC10]. Their offline solver for ρPOMDPs requires discrete state, action and observation spaces and a piecewise linear and convex reward function ρ.

Based on these ideas, IPFT augments the reward function by an additional term specifying the information gain $\Delta\mathcal{I}_\gamma(b,b')$, such that the belief-dependent reward becomes a weighted sum of the expected state-rewards and the information gain:

$$\rho(b,a,b') = \int_{\mathbb{S}} \mathcal{R}(s,a)b(s)\,\mathrm{d}s + \lambda\Delta\mathcal{I}_\gamma(b,b'), \tag{4.2}$$

where the parameter λ serves as factor to trade-off reward maximization over information gathering.

The form of the information gain $\Delta\mathcal{I}_\gamma(b,b')$ is chosen following the suggestion from Eck et al. to use potential-based reward shaping (PBRS) to implicitly guide the agents search [ESDK16]. It can be either discounted, where $\gamma \in (0,1)$ serves as discount factor, or undiscounted, where γ is set to one:

$$\Delta\mathcal{I}_\gamma(b,b') = \gamma\mathcal{I}(b') - \mathcal{I}(b), \tag{4.3}$$

with b' being the belief state simulated one time step further than b and the function $\mathcal{I}(b)$ being the potential function for a belief b.

Fischer and Tas make the distinction between those two cases, because in the discounted case the optimal policy is guaranteed to be invariant under PBRS [ESDK16], whereas the undiscounted case resembles more intuitively the idea of information gathering. However, in their experiments the performance of both variants were almost equivalent [FT20].

Information Measures as Potential Function of the Belief State

In IPFT, the potential function $\mathcal{I}(b)$ is chosen based on information measures, where Fischer and Tas define an information measure $\mathcal{I}$ in general as convex functions over the belief space. Eck et al. suggest several other types of potential functions, but IPFT uses information measures for multiple reasons:

- Information measures are domain independent, i.e. gathering information is rewarded on any domain and no prior knowledge must be included.
- In case of uncertainty about the initial state of the agent, reducing this uncertainty is often part of the optimal policy. Since at first the agent should determine its state with reasonable certainty, before it maximizes the reward.
- The optimal value function V^* of a POMDP is also convex. Thus, potentials based on the convex information measures serve as heuristic for V^*.

There exist various information measures, that meet the convexity requirement. In IPFT, however, the negative entropy

$$-\mathcal{H}(b) = \sum_{s \in \mathcal{S}} b(s)\ \log b(s) \tag{4.4}$$

is used as information measure for discrete states. In continuous state spaces, the negative differential entropy

$$-\mathcal{H}(b) = \int_{\mathbb{S}} b(s)\ \log b(s)\, \mathrm{d}s \tag{4.5}$$

serves as information measure for the belief distribution $\mathrm{bel}(s)$ over all states.

As IPFT approximates the belief $b(s)$ as weighted particle set $\hat{b}_m = \{s^i, w^i\}_{i=1}^m$ with normalized weights, $b(s)$ is only known at discrete points s^i. Hence, the integral is approximated by the weighted sum and the continuous belief distribution $b(s) \approx b_{\mathrm{KDE}}(s)$ is estimated using Kernel Density Estimation. This results in the estimate of the negative differential entropy:

$$-\mathcal{H}(b) \approx -\hat{\mathcal{H}}(\hat{b}_{\mathrm{KDE}}) = \sum_{i=1}^{m} w^i \log \hat{b}_{\mathrm{KDE}}(s^i) \tag{4.6}$$

For estimation of $b_{\mathrm{KDE}}(s)$ a Gaussian kernel is used [Gra18, p.29,35] and the bandwidth is selected according to Silverman's rule of thumb [Gra18, p.65,74]. The details of computing $\hat{b}_{\mathrm{KDE}}(s)$ are given in the supplemental material of [FT20] and are therefore not restated here.

However, using Silvermans rule of thumb for bandwidth selection can yield very inaccurate estimates, if the true density is not close to being Gaussian. In the experiments, especially in the Continuous Light Dark setting, where IPFT outperforms existing solvers, this is not a problem, since all probability distributions are Gaussians and the transition and observation models are linear. Therefore, all distributions remain Gaussian and are sufficiently well approximated by Gaussian kernels with a bandwidth computed by Silvermans rule of thumb.

Another property of the Continuous Light Dark problem, that made KDE methods reasonable, is the one dimensional state space, since the agent moves only on the real axis back and forth. In such low dimensional problems the probability density function could be approximated well enough by a small particle set used during tree search, such that the entropy estimate $\hat{\mathcal{H}}$ is still practicable to guide the agent in the direction of informative belief states.

4.2. Belief Tree Search

IPFT belongs to the class of Monte-Carlo tree search algorithms and therefore also consists of the four basic steps: node selection, node expansion, rollout and backpropagation (see Sec. 3.2.1). It uses the UCB1 algorithm for action selection similar to POMCP and applies progressive widening to both action and observation space analog to POMCPOW, if necessary.

The overall structure resembles the one from POMCPOW and is given in Algorithm 4.13. As IPFT simulates beliefs through the tree, each simulation starts by sampling a small particle set $\hat{b}_m$ with m particles from the initial belief b, which is represented by a bigger particle set ($N \gg m$). Then, as in the other algorithms, SIMULATE() is called in order to incrementally build the search tree (see Fig. 4.1).

In SIMULATE, at first an action is selected according to the function ACTIONPROGWIDEN (line 15). Then, a new observation is generated (line 16) from the belief $\hat{b}_m$. This is done by at first drawing a sample particle from the belief $s \sim \hat{b}_m$ with probability according to its weight. With this state s and the selected action a successor state s' is sampled from the transition model $s' \sim \mathcal{T}(\cdot|s,a)$ to generate the observation $o' \sim \mathcal{Z}(\cdot|s',a)$. In line 17 the posterior belief $\hat{b}'_m$ is computed with a particle filter update $\mathcal{G}_{\mathrm{PF}(m)}$ (see Algorithm 2.5).

In the implementation of IPFT sequential importance resampling (Algorithm 2.4), where resampling takes place in each step, is used for the particle filter update. This means that all particle sets $\hat{b}_m$ in the observation nodes are unweighted, as after resampling all particle weights are equal (see Sec. 2.3.3). Note, that even though possible future beliefs are simulated in this step, this is different from prediction in the context of state estimation (see Sec. 2.3), since the belief updates are computed with simulated observations sampled from the observation model. In prediction, no new observations and actions are available.

The algorithm proceeds in line 18 with the calculation of the shaped rewards $\rho(\hat{b}_m, a, \hat{b}'_m)$ with Equation (4.2). If the number of observation child nodes $|C(ha)|$ does not exceed the progressive widening limit, the tree is expanded by another observation node and the value is estimated with ROLLOUT. Otherwise an existing observation node is selected uniformly from the set of child nodes $C(ha)$, its observation o is replaced by the newly generated one o' and SIMULATE is called recursively from this node. Finally, the value and counts of the nodes are updated during backpropagation.

Observation generation

The consequence of this observation replacement step is that an observation is never used twice in the tree (see Figure 4.1, where observation edges are unlabeled). Hence, the particle filter update always relies on the currently generated observation, and always discards previously sampled beliefs at a node upon new visits to those already existing nodes. Experiments, in which observations where reused for belief updates during tree search did not improve or speed up the convergence of the algorithm. In fact, reusing observations has lead to edge cases, where the random generation of very unlikely observations result in degenerated particle sets with all particles being equal. This in turn resulted in very high kernel density estimates and hence high information rewards, as the variance of a particle set with only equal particles is zero. The high information reward misguided the tree search and the consequence was the selection of a suboptimal action.

From a theoretical perspective this means that during execution of SEARCH multiple independent episodes (i.e. calls to SIMULATE) are sampled from the initial belief. Here independent means that neither previously sampled observations, nor beliefs or particles are reused in different episodes. The quantities which are stored in the tree and hence reused over multiple different episodes are only the visitation counts and the (Q-)value function estimates at each node for action selection.

PFT without Progressive Widening

The two main reasons to use progressive widening in MCTS is to apply the UCB action selection algorithm in continuous action spaces and to ensure a deep tree in continuous state and observation spaces (see 3.8) [CHS+11, SK18]. However, in case of action spaces, it is a common practice in standard benchmark problems (as for e.g. Continuous Light Dark or Laser Tag [SK18, SYHL13, SV10]) to adapt the POMDP model, such that the actions remain discrete. Later, when the driving scenario will be formulated as POMDP (see Chapter 5), the acceleration will be discretized analogously such that the algorithm

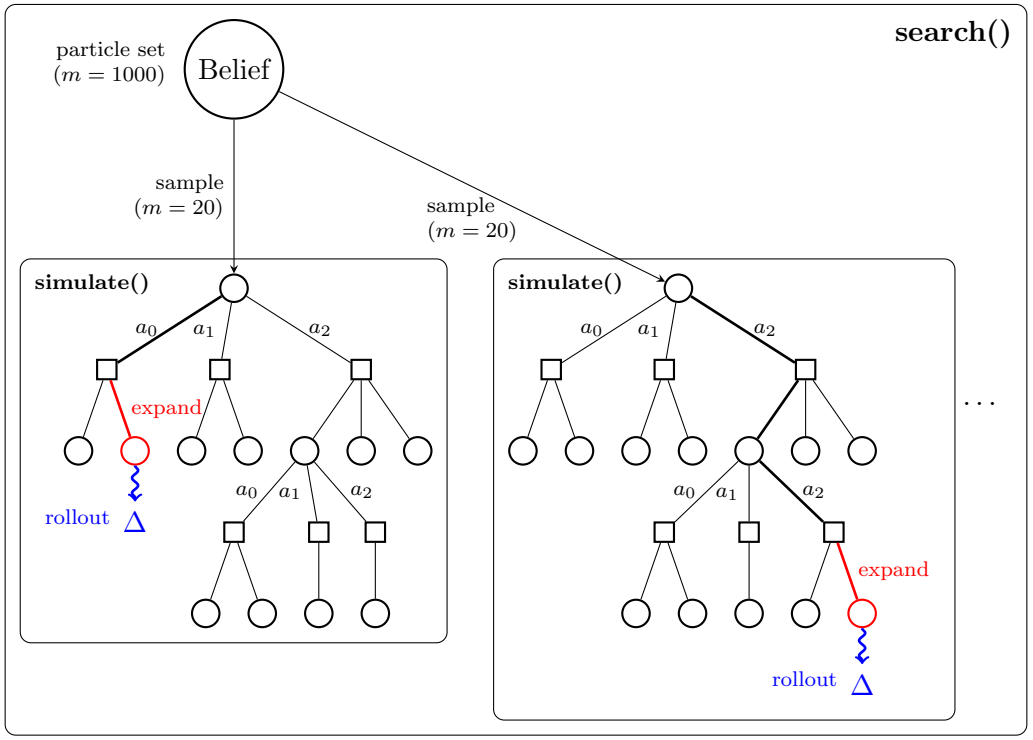


Figure 4.1.: Two iterations of belief tree search of the IPFT algorithm.

only needs to choose between different discrete actions. Therefore, in this thesis progressive widening will not be necessary for the action space.

In continuous observation spaces, progressive widening artificially limits the number of observation nodes and hence forces the algorithm to reuse observations as for e.g. in POMCPOW [SK18]. However, as explained in the previous paragraph, the experiments conducted in this thesis suggest that reusing observations for particle filter updates during tree search is not practicable. Hence, progressive widening might not be beneficial in algorithms, where observations are not reused, as in IPFT. Therefore, the impact of progressive widening on the observation space will be investigated further in Chapter 6.

Algorithm 4.13 Information Particle Filter Tree Algorithm

function SEARCH(b)
  repeat
    $\hat{b}_m \leftarrow$ sample from b
    ▷ $\hat{b}_m$ is a particle set with m particles
    SIMULATE($\hat{b}_m, h, d_{max}$)
  until timeout
  return $\arg\max_a Q(ba)$

function ACTIONPROGWIDEN(h)
  if $|C(h)| \leq k_a N(h)^{\alpha_a}$ **then**
    $a \leftarrow$ NEXTACTION(h)
    $C(h) \leftarrow C(h) \cup \{a\}$
  return $\arg\max_{a \in C(h)} Q(ha) + c\sqrt{\frac{\log N(h)}{N(ha)}}$

function SIMULATE($\hat{b}_m, h, d$)
  if $d = 0$ **then**
    return 0
  $a \leftarrow$ ACTIONPROGWIDEN(h)
  $o' \leftarrow$ sample s from $\hat{b}_m$,
    generate o' from (s, a)
  $\hat{b}'_m \leftarrow \mathcal{G}_{\mathrm{PF}(m)}(\hat{b}_m, a, o')$
  $r \leftarrow \rho(\hat{b}_m, a, \hat{b}'_m)$
  if $|C(ha)| \leq k_o N(ha)^{\alpha_o}$ **then**
    $C(ha) \leftarrow C(ha) \cup \{o'\}$
    $R \leftarrow r + \gamma \cdot$ ROLLOUT$(\hat{b}'_m, hao', d-1)$
  else
    $o \leftarrow$ select $o \in C(ha)$ uniformly
    $o \leftarrow o'$
    $R \leftarrow r + \gamma \cdot$ SIMULATE$(\hat{b}'_m, hao, d-1)$
  $N(h) \leftarrow N(h) + 1$
  $N(ha) \leftarrow N(ha) + 1$
  $Q(ha) \leftarrow Q(ha) + \frac{R - Q(ha)}{N(ha)}$
  return R

5. Formulating the Motion Planning Problem as POMDP

In the previous chapters, important fundamentals of decision-making and state estimation have been presented. Moreover, state of the art offline and online solvers that are capable of solving decision-making problems in continuous spaces were introduced. In this chapter, this theory will be applied to the motion planning problem for automated driving at urban intersection scenarios.

In general, motion planners for automated driving need to operate in highly uncertain environments, such as intersections in urban areas. The uncertainties in those traffic situations arise from the vehicle's imperfect perception of its own state (e.g. the position) and other traffic participants and the imperfect prediction of their future behavior. However, having a good prediction of the future behavior of surrounding road users is crucial for generating the future trajectory of the ego vehicle, as the behavior of the ego vehicle strongly depends on the behavior of others. Since it is impossible to predict the future with certainty, the automated vehicle needs to generate its trajectory based on uncertain predictions. To consider these uncertainties in prediction and perception directly in the planning process, the motion planning problem is formulated as POMDP (see Ch. 2).

The problem formulation used in this work is an extension to the one used in [HSB+18]. The new main contributions are

- the use of the interactive intelligent driver model to simulate other agents' behavior forward in time,
- the additional use of other vehicles pose (yaw-angle) as a feature to estimate the drivers route intention,
- the extension of the planning algorithm to dynamic map updates in every time step,
- the adaption of the model to be used with weighted particle filters for belief state updating,
- the application of a Particle Filter Tree online POMDP solver to the decision-making problem.

5.1. Problem Statement

Similar to [HSB+18], the output of the motion planner presented in this work is a sequence of desired acceleration values $(a_0^{t_0}, a_0^{t_1}, a_0^{t_2}, \dots)$ for driving in an urban scenario with an

arbitrary road layout and variable number of other traffic participants with unknown intentions. The path of the ego vehicle r_0 and the path options $r_k^{(i)}$ of all other vehicles are extracted online from a given map and are assumed to be collision free with respect to static obstacles. The sequence of acceleration values then defines the longitudinal velocity along the path of the ego vehicle, which is referred to as path-velocity decomposition in the literature and reduces the trajectory planning problem to a one dimensional problem [KZ86].

In the problem setting, the environment consists of a set of agents (in this work only vehicles are considered) $\mathcal{V} = \{V_0, V_1, \ldots, V_{N_\mathcal{V}}\}$ with variable size, with $N_\mathcal{V} \in \mathbb{N}_0$ and the ego vehicle V_0 and its given route r_0. In every time step $t \in \mathbb{N}$, every other vehicle V_k, with $k \in \{1, 2, \ldots, N_\mathcal{V}\}$ has a set of route options $\mathcal{R}_{k,t} = \{r_{k,t}^{(1)}, r_{k,t}^{(2)}, \ldots, r_{k,t}^{(N_{\mathcal{R}_{k,t}})}\}$, with $N_{\mathcal{R}_{k,t}} \in \mathbb{N}$. Upon change, the route options are updated online for every other agent based on their current position in the map. A route option is given by its arc-centerline which consists of a list of waypoints $r_{k,t}^{(i)} = (q_0, q_1, \ldots, q_M)$, with $M \in \mathbb{N}$, and the vectors $q_j \in \mathbb{R}^2$ defined in the global Cartesian coordinate system $\mathcal{O}$, such that every route defines its own Frenet coordinate system, denoted as $\mathcal{F}_{r_{k,t}}$. Any other vehicle V_k may traverse from its current route to another route with the unknown probability $P(r_{k,t} | \, r_{k,t-1})$. The route of the ego vehicle does not change over time.

5.2. POMDP Formulation

To generate the velocity profile along the ego vehicle's path as stated in the previous section, the problem is now formulated as POMDP, which will then be solved by the online Monte-Carlo tree search based solver introduced in Chapter 4.

In Section 2.2, the POMDP was defined as 7-tuple $(\mathbb{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathbb{O}, \mathcal{Z}, \gamma)$. Therefore, all elements of this tuple will be defined in this section.

5.2.1. Action-, State- and Observation Space

Motions and positions of vehicles in traffic are continuous and as the Particle Filter Tree (PFT) algorithm (Ch. 4) can deal with continuous state spaces $\mathbb{S}$ and observation spaces $\mathbb{O}$ they do not need to be discretized.

For action selection, PFT uses the UCB algorithm, which means that discrete actions are required (see Sec. 3.2.1) since in this work no progressive widening is used for the action space (see Sec. 4.2). Hence, the action space, which in reality is a continuous range of accelerations, will be discretized to an equidistant spaced finite set of acceleration values, even though the actuators of automated vehicles, which are in this case break and throttle, can select continuous acceleration and deceleration values. This means that the behavior generated by this POMDP planner can serve as reference input to lower planning or control layers in the vehicle architecture, where the generated velocity profile is then smoothed out, for example.

State space

Analog to [HSB+18], all vehicles in the scene are included in the state space, in order to allow the modeling of interactive behavior between the ego vehicle and others. Hence, a certain state $s_t \in \mathbb{S}$ at time t is defined as:

$$s_t = (\mathbf{s}_{V_0,t}, \mathbf{s}_{V_1,t}, \mathbf{s}_{V_2,t}, \ldots, \mathbf{s}_{V_{N_\mathcal{V}},t})^\top, \tag{5.1}$$

where $\mathbf{s}_{V_0,t}$ represents the state of the automated vehicle and $\mathbf{s}_{V_k,t}, k \in \{1, 2, \ldots, N_\mathcal{V}\}$ the states of the other vehicles in the scene. To ease notation, the subscript t will be dropped in the following.

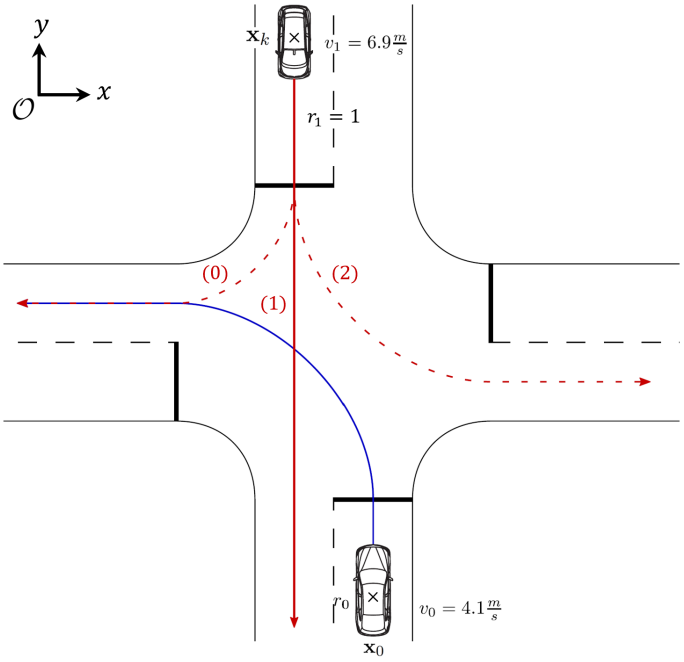


Figure 5.1.: Visualization of the state space (adapted from [HSB+18]).

Since in this work, the planner should be able to deal with dynamic map updates as defined in Section 5.1, it is not reasonable to store the longitudinal position of the vehicles on their respective route, since the route may change from time step to time step. Such map updates – in the following called environment updates – occur for example when the car has traversed one intersection and arrives at the next one. In order to avoid repeated reinitialization of the planner or planning with huge maps, the POMDP model is adapted to support those environment updates.

Therefore, in this work instead of the longitudinal Frenet coordinate, the position of each vehicle is included in global Cartesian coordinates in the state space. The ego vehicle state is thus defined as

$$\mathbf{s}_{V_0} = \begin{pmatrix} \mathbf{x}_0 \\ v_0 \end{pmatrix}, \tag{5.2}$$

where $\mathbf{x}_0 = (x, y)^\top \in \mathbb{R}^2$ is the position vector of the ego vehicle in the global Cartesian coordinate system $\mathcal{O}$ and $v_0 \in \mathbb{R}$ is its velocity. The state of the other vehicles is

$$\mathbf{s}_{V_k} = \begin{pmatrix} \mathbf{x}_k \\ v_k \\ r_k \end{pmatrix}, \tag{5.3}$$

with $\mathbf{x}_k = (x, y)^\top \in \mathbb{R}^2$ is the position vector of the other vehicle in the global Cartesian coordinate system $\mathcal{O}$ and $v_k \in \mathbb{R}$ is its velocity. The route $r_k \in \mathcal{R}_k$ is the hidden variable in the state space, which cannot be observed directly.

Observation space

Similar to the states, the observations $o_t \in \mathbb{O}$ at time t are defined as

$$o_t = (\mathbf{o}_{V_0,t}, \mathbf{o}_{V_1,t}, \mathbf{o}_{V_2,t}, \ldots, \mathbf{o}_{V_{N_\mathcal{V}},t})^\top, \tag{5.4}$$

with $\mathbf{o}_{V_0,t}$ the observation of the ego vehicle and $\mathbf{o}_{V_k,t},\ k \in \{1, 2, \ldots, N_\mathcal{V}\}$ the observations of the other vehicles. Again to simplify notation, the time t is left out in the following.

As the route of the ego vehicle is known, its state is fully observable, such that the ego observation resembles the ego state:

$$\mathbf{o}_{V_0} = \begin{pmatrix} \mathbf{x}_0 \\ v_0 \end{pmatrix}, \tag{5.5}$$

where $\mathbf{x}_0 = (x, y)^\top \in \mathbb{R}^2$ is the observed position vector of the ego vehicle in the global Cartesian coordinate system $\mathcal{O}$ and $v_0 \in \mathbb{R}$ is its observed velocity. The route of the other vehicles cannot be observed directly, as the route intention cannot be measured directly by sensors. Hence, it is not part of the other vehicles' observation. However, perception systems of automated vehicles are capable of estimating other vehicles pose. Therefore, the pose of the other vehicles is included in the observations, such that they are defined as

$$\mathbf{o}_{V_k} = \begin{pmatrix} \mathbf{x}_k \\ \theta_k \\ v_k \end{pmatrix}, \tag{5.6}$$

with $\mathbf{x}_k = (x, y)^\top \in \mathbb{R}^2$ is the position vector of the other vehicle in the global Cartesian coordinate system $\mathcal{O}$, $\theta_k \in \mathbb{R}$, the other vehicles' pose, given by its yaw angle and $v_k \in \mathbb{R}$ is its velocity.

Action set

As explained in the beginning of this section, the actions for the ego vehicle need to be discretized in the POMDP problem formulation. Therefore, to generate an optimal sequence of desired acceleration values (see Sec. 5.1) the actions $a_t \in \mathcal{A}$ at time t are taken from the equidistant spaced set

$$\mathcal{A} = \{-4.5, -3.0, -1.5, 0.0, 1.5\} \, \frac{m}{s^2}. \tag{5.7}$$

The asymmetry between maximum acceleration and deceleration values reflects the asymmetric acceleration capabilities of vehicles. Moreover, the rather coarse discretization is chosen to resemble the intuitive decisions human drivers need to make at intersection in urban scenarios: "accelerate", "keep driving at the current speed" or "brake in different strengths depending on the situation". Another reason not to use a finer discretization is to reduce computational cost and to enable planning for longer horizons.

5.2.2. Transition Model

The transition model $\mathcal{T}(s_{t+1} | \, s_t, a_t)$ is used by the planner to predict multiple possible future outcomes based on its current belief state, which is approximated by a particle set consisting of many weighted state instances according to Section 5.2.1. As it is impossible to predict the behavior of other road users with certainty, the model is chosen to be probabilistic. Moreover, the model is also interactive since the interaction of other vehicles with the ego vehicle is considered.

As the planner holds at least one route for each agent in the scene, and it is assumed that every agent moves only along one of its routes (see Sec. 5.1), the motion of all agents can be described by the one dimensional discrete motion model $\mathbf{l}_{t+1} = \mathcal{M}(\mathbf{l}_t, a_t)$ with step size Δt:

$$\mathbf{l}_{t+1} = \begin{pmatrix} l_{t+1} \\ v_{t+1} \end{pmatrix} = \mathcal{M}(\mathbf{l}_t, a_t) = \mathcal{M}(l_t, v_t, a_t) = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} l_t \\ v_t \end{pmatrix} + \begin{pmatrix} \frac{1}{2}(\Delta t)^2 \\ \Delta t \end{pmatrix} a_t, \tag{5.8}$$

where $\mathbf{l}_t = (l, \dot{l})^\top = (l, v)^\top \in \mathbb{R}^2$ is a vector describing the position l and velocity v of a vehicle along a route and a_t is the applied acceleration at time t.

Each state s_t holds the position $\mathbf{x}_k$ of each vehicle in the global Cartesian coordinate system $\mathcal{O}$. Thus, on each call to the transition model the Cartesian position $\mathbf{x}_k = (x, y)^\top$ must be matched to the respective Frenet coordinates $\mathbf{f}_k = (l, d)^\top$ in the coordinate system $\mathcal{F}_{r_k}$ of route r_k and transformed back to Cartesian coordinates afterwards. Hence, with a slight abuse of notation, the motion model $\mathcal{M}_{r_k}$ of a vehicle along a route can be defined as

$$\begin{pmatrix}\mathbf{x}_{t+1}\\ v_{t+1}\end{pmatrix} = \mathcal{M}_{r_k}(\mathbf{x}_t, v_t, a_t) \mapsto {}^{\mathcal{O}}\mathrm{T}^{\mathcal{F}_{r_k}} \mathcal{M}\left({}^{\mathcal{F}_{r_k}}\mathrm{T}^{\mathcal{O}}\mathbf{x}_t, v_t, a_t\right), \tag{5.9}$$

where the operator ${}^{B}\mathrm{T}^{A}$ represents a coordinate transform from coordinate system A to B.

Transition Model of the Ego Vehicle

With the above definition, the transition model for the ego vehicle with added Gaussian noise can be written as

$$\mathbf{s}_{V_0,t+1} = \mathcal{T}(\mathbf{s}_{V_0,t+1} | \, \mathbf{s}_{V_0,t}, a_t) = {}^{\mathcal{O}}\mathrm{T}^{\mathcal{F}_{r_0}} \left(\mathcal{M}\left({}^{\mathcal{F}_{r_0}}\mathrm{T}^{\mathcal{O}}\mathbf{x}_t, v_t, a_t\right) + \begin{pmatrix} n_{\mathbf{x},V_0} \\ n_{v,V_0} \end{pmatrix} \right), \tag{5.10}$$

where $n_{v,V_0} \in \mathbb{R} \sim \mathcal{N}(0, \sigma_{v,V_0})$ and $n_{\mathbf{x},V_0} \in \mathbb{R} \sim \mathcal{N}(0, \sigma_{\mathbf{x},V_0})$. $n_{\mathbf{x},V_0}$ is added to the posterior longitudinal position l_{t+1} before back transformation to Cartesian coordinates. The acceleration value a_t is the action selected by the planner for the ego vehicle, based on the computed policy and the current belief state, s.t. $a_t = \pi(b_t)$.

Transition Model of the Other Vehicles

In order to predict the behavior of the surrounding vehicles in the scene with respect to the ego vehicle, an appropriate model that describes human driver behavior is necessary. Moreover, the model should be easy to evaluate in terms of computational cost, since it will often be queried during tree search. Hence, finding a suitable model is not a trivial task, since building those models is a field with a lot of ongoing research [BDCK20].

In this work, the Intelligent Driver Model (IDM) is chosen as model for the other vehicles [THH00]. Originally, the IDM was designed as car-following model for one-lane situations and is used for example in Adaptive Cruise Control (ACC) driver assistance systems for highway driving [KTH10]. It is a parametric rule-based model, that balances the desire to achieve free speed in case no vehicle is at front and the need to maintain a safe distance to a leading vehicle if there is one.

Since the future trajectory of other vehicles is predicted along given route hypotheses, which can be considered as different single lanes, it is considered as reasonable choice in the transition model for the POMDP formulation in this work.

The IDM acceleration function is given by

$$a_{\mathrm{IDM}}(d_{\mathrm{bump}}, v, \Delta v) = a_{max} \left[1 - \left(\frac{v}{v_{ref,k}} \right)^{\delta} - \left(\frac{d^*(v, \Delta v)}{d_{\mathrm{bump}}} \right)^2 \right], \tag{5.11}$$

where the desired (safe) distance is

$$d^*(v, \Delta v) = d_{min} + vT + \frac{v \Delta v}{2\sqrt{a_{max} b}}, \tag{5.12}$$

and d_{bump} is the (bumper-to-bumper) distance to the leading vehicle, v_k the current velocity of the vehicle and $\Delta v = v_k - v_{lead}$ the velocity difference or approaching rate to the leading vehicle. The other parameters are given in Table 5.1.

Parameter	Description
$v_{ref,k}$	desired/reference velocity of vehicle V_k
δ	free acceleration exponent
T	desired time gap
d_{min}	minimum jam distance
a_{max}	maximum acceleration
b	desired deceleration

Table 5.1.: Parameters of the Intelligent Driver Model.

This expression combines the free-road acceleration strategy

$$a_{ref,k} = a_{max}\left[1 - \left(\frac{v}{v_{ref,k}}\right)^{\delta}\right], \tag{5.13}$$

which accelerates the vehicle V_k to its reference velocity $v_{ref,k}$ with an interactive deceleration strategy

$$a_{int,k} = -a_{max}\left(\frac{d^*}{d_{\text{bump}}}\right)^2, \tag{5.14}$$

that becomes relevant when the gap to the leading vehicle is not significantly larger than the effective desired (safe) distance d^*.

In this work, only the interaction of other vehicles with the ego vehicle (and no interaction among other vehicles) is considered. Therefore, in case the ego vehicle is not leading to vehicle V_k, the acceleration a_{V_k} for vehicle V_k is given by

$$a_{V_k} = a_{ref,k} + n_{\text{IDM}}, \tag{5.15}$$

where $n_{\text{IDM}} \in \mathbb{R} \sim \mathcal{N}(0, \sigma_{\text{IDM}})$ is an additional Gaussian noise to accommodate for model and parameter mismatch, as well as different driving styles of other drivers. If the ego vehicle is on route r_k of the other vehicle V_k, then the acceleration a_{V_k} for vehicle V_k is calculated by

$$a_{V_k} = a_{ref,k} + a_{int,k} + n_{\text{IDM}}, \tag{5.16}$$

with n_{IDM} defined as above. Whether the ego vehicle is on vehicle V_k's route, can be determined by transforming the ego vehicles position in the routes coordinate system $\mathcal{F}_{r_k}$ and checking if the lateral distance coordinate d is less than the lane width. The lane width can either be passed to the planner for every route or — as done here — be set to a constant lane width parameter w_{lane} for all routes.

Having determined the behavior model of the other vehicles in the scene, the transition model can then be formulated similar to the ego transition model as

$$\mathbf{s}_{V_k,t+1} = \begin{pmatrix} \mathbf{x}_{k,t+1} \\ v_{k,t+1} \\ r_{k,t+1} \end{pmatrix} = \mathcal{T}(\mathbf{s}_{V_k,t+1} | \, \mathbf{s}_{V_k,t}, a_{V_k,t}) = \begin{pmatrix} \mathcal{M}_{r_k}(\mathbf{x}_{k,t}, v_{k,t}, a_{V_k}) \\ r_{k,t} \end{pmatrix}, \tag{5.17}$$

where the route r_k of a vehicle V_k in each state does not change over time.

In real traffic, there are often many other surrounding vehicles, which need to be considered for trajectory planning of the ego vehicle, even if they do not interact with the ego vehicle at the moment, but may in the future. Hence, most of the time the behavior of the other vehicles will be determined by the free-road driving strategy (see Eqs. (5.13) and (5.15)) and the accelerations generated by this strategy in turn strongly depend on the desired or reference velocity $v_{ref,k}$ of vehicle V_k. Hence, including this parameter in the

state space and thus in the planning process might be a promising extension to this work to improve the prediction of other vehicles' behavior during tree search. Similar to the route intention, this desired goal velocity is another hidden internal state of other drivers, which cannot be observed but would have to be inferred or estimated from observations as well. Bhattacharyya et al. tackle exactly this problem in their work and estimate the desired velocity of human drivers online using a particle filter in order to improve prediction [BSBK20]. Another work using a particle filter for intention estimation is [SHLB18].

So far, the parameter is set manually to a reasonable constant value depending on the scenario and the speed limit, and it is argued that the additional noise (in Eqs. (5.15) and (5.16)) takes deviations from the true value into account. However, the variation of this parameter still has an influence on the Q-function and hence the planned trajectory. This relation will be evaluated later in Chapter 6.

5.2.3. Observation Model

The Particle Filter Tree algorithm uses particle filter updates with simulated observations during tree search in order to simulate possible beliefs forward in time (see Sec. 4.2). In this way, it solves the corresponding belief MDP of the POMDP (see Sec. 2.2). Therefore, on one hand the observation model serves as generative model, such that observation samples $o_{t+1} \sim \mathcal{Z}(\cdot|\, s_{t+1})$ are generated from $\mathcal{Z}$ given a posterior state sample s_{t+1}, and on the other hand $\mathcal{Z}(o_{t+1}|\, s_{t+1})$ must be evaluated point-wise, such that it can be used during weighted particle filter updates in order to determine the weight of particles s_{t+1} with either simulated or real observations o_{t+1} (see Sec. 3.2.3).

Generative Observation Model

The generative observation model, denoted as $\mathcal{Z}(\cdot|\, s_{t+1})$ generates observations $\mathbf{o}_{V_0,t+1}$ for the ego vehicle and for all other vehicles $\mathbf{o}_{V_k,t},\ k \in \{1, 2, \dots, N_\mathcal{V}\}$. Since the action a_t is already incorporated in the posterior state of the ego vehicle $\mathbf{s}_{V_0,t+1}$, the generated observation o_{t+1} does not depend on the previous action and is therefore no argument of $\mathcal{Z}$.

As the state of the ego vehicle is fully observable, $\mathbf{o}_{V_0,t+1}$ can be simply generated from $\mathbf{s}_{V_0,t+1}$ by adding Gaussian noise (the subscript $t+1$ is dropped to simplify notation):

$$\mathbf{o}_{V_0} = \begin{pmatrix} \mathbf{x}_0 \\ v_0 \end{pmatrix} \sim \mathcal{Z}(\cdot|\, \mathbf{s}_{V_0}) = \begin{pmatrix} \mathcal{N}(\mathbf{s}_{V_0,\mathbf{x}}, \Sigma_{\mathbf{x},V_0}) \\ \mathcal{N}(\mathbf{s}_{V_0,v}, \sigma_{v,V_0}) \end{pmatrix}, \tag{5.18}$$

where the position vector $\mathbf{x}_0$ of the observation is sampled from a two-dimensional Normal distribution with covariance matrix $\Sigma_{\mathbf{x},V_0} = \begin{pmatrix} \sigma_{\mathbf{x},V_0} & 0 \\ 0 & \sigma_{\mathbf{x},V_0} \end{pmatrix} \in \mathbb{R}^{2\times 2}$. The velocity v_0 is sampled from a normal distribution with standard deviation σ_{v,V_0}.

To generate observations for the other vehicles $\mathbf{o}_{V_k,t+1}$ from their states $\mathbf{s}_{V_k,t+1},\ k \in \{1, 2, \dots, N_\mathcal{V}\}$ their pose (yaw-angle) must be sampled. For this, it is assumed that a vehicles pose is approximately aligned with the direction of route r_k it is following. Hence, the yaw angle in $\mathbf{o}_{V_k,t+1}$ is sampled from a normal distribution with the mean being the angle of the pseudo tangent vector $\theta_{\tan,r_k}(\mathbf{x})$ of route r_k at point $\mathbf{x} \in \mathbb{R}^2$ in the Cartesian coordinate system $\mathcal{O}$. The calculation of $\theta_{\tan,r_k}(\mathbf{x})$ is described in [ZBDS14].

Since position $\mathbf{x}_k$ and velocity v_k of the other vehicles are sampled analog to the ego vehicle (5.18), the observations $\mathbf{o}_{V_k,t+1}$ for every vehicle $V_k,\ k \in \{1, 2, \dots, N_\mathcal{V}\}$ are generated according to:

$$\mathbf{o}_{V_k} = \begin{pmatrix} \mathbf{x}_k \\ \theta_k \\ v_k \end{pmatrix} \sim \mathcal{Z}(\cdot|\, \mathbf{s}_{V_k}) = \begin{pmatrix} \mathcal{N}(\mathbf{s}_{V_k,\mathbf{x}}, \Sigma_{\mathbf{x},V_k}) \\ \mathcal{N}(\theta_{\tan,r_k}(\mathbf{x}_k),\, \sigma_{\theta,V_k}) \\ \mathcal{N}(\mathbf{s}_{V_k,v},\, \sigma_{v,V_k}) \end{pmatrix}, \tag{5.19}$$

where the sampled position $\mathbf{x}_k$ is used to calculate $\theta_{\tan,r_k}$ and $\Sigma_{\mathbf{x},V_k} \in \mathbb{R}^{2\times 2}$ is defined analog to (5.18).

Together, Equations (5.18) and (5.19) build the generative observation model of the POMDP.

Observation Likelihood Model

In order to apply a (weighted) particle filter, the conditional probability density $\mathcal{Z}(o_{t+1}|\, s_{t+1})$ is required, that defines the likelihood of a posterior state s_{t+1}, given an observation o_{t+1}. This density is also called *measurement model* (see Sec. 2.3).

The probability $\mathcal{Z}(o_{t+1}|\, s_{t+1})$ is obtained as the product of the observation likelihood of all vehicles (including the ego vehicle) in the scene (the subscript $t+1$ is omitted to simplify notation):

$$\mathcal{Z}(o_{t+1}|\, s_{t+1}) = p(\mathbf{o}_{V_0}|\, \mathbf{s}_{V_0}) \prod_{k=1}^{N_\mathcal{V}} p(\mathbf{o}_{V_k}|\, \mathbf{s}_{V_k}). \tag{5.20}$$

Technically, this amounts to an independence assumption between the noise in each observation of any other vehicle, which is only true in the ideal case and might not hold in cases where two other vehicles are close to each other. However, this assumption is common practice and therefore also used in this work [TBF06, p.152].

The likelihood $p(\mathbf{o}_{V_0}|\, \mathbf{s}_{V_0})$ for the ego vehicle is defined as

$$\begin{aligned} p(\mathbf{o}_{V_0}|\, \mathbf{s}_{V_0}) &= p(\mathbf{o}_{\mathbf{f}_l,V_0}|\, \mathbf{s}_{\mathbf{f}_l,V_0}) \cdot p(\mathbf{o}_{v,V_0}|\, \mathbf{s}_{v,V_0}) \\ &= \mathcal{N}(\mathbf{o}_{\mathbf{f}_l,V_0}|\, \mathbf{s}_{\mathbf{f}_l,V_0}, \sigma_{\mathbf{f}_l,V_0,\mathrm{PF}}) \cdot \mathcal{N}(\mathbf{o}_{v,V_0}|\, \mathbf{s}_{v,V_0}, \sigma_{v,V_0,\mathrm{PF}}), \end{aligned} \tag{5.21}$$

where $(\cdot)_{\mathbf{f}_l}$ denotes the l-coordinate of the position in Frenet coordinates of the respective route: $\mathbf{f} = {}^{\mathcal{F}_{r_k}}\mathrm{T}^{\mathcal{O}}\mathbf{x}$.

The likelihood $p(\mathbf{o}_{V_k}|\, \mathbf{s}_{V_k})$ for the other vehicles V_k is defined analog to (5.21), but with an additional likelihood $p(\mathbf{o}_{r_k,V_k}|\, \mathbf{s}_{r_k,V_k})$ for the route r_k:

$$p(\mathbf{o}_{V_k}|\, \mathbf{s}_{V_k}) = p(\mathbf{o}_{\mathbf{f}_l,V_k}|\, \mathbf{s}_{\mathbf{f}_l,V_k}) \cdot p(\mathbf{o}_{v,V_k}|\, \mathbf{s}_{v,V_k}) \cdot p(\mathbf{o}_{V_k}|\, \mathbf{s}_{r_k,V_k}), \tag{5.22}$$

where the first two terms are computed analog to (5.21) by Normal distributions $\mathcal{N}(\mathbf{o}_{\mathbf{f}_l,V_k}|\, \mathbf{s}_{\mathbf{f}_l,V_k}, \sigma_{\mathbf{f}_l,V_k,\mathrm{PF}})$ and $\mathcal{N}(\mathbf{o}_{v,V_k}|\, \mathbf{s}_{v,V_k}, \sigma_{v,V_k,\mathrm{PF}})$, respectively.

The third term $p(\mathbf{o}_{r_k,V_k}|\, \mathbf{s}_{r_k,V_k})$ is the route likelihood function that determines the likelihood of $\mathbf{s}_{V_k}$ being on route r_k given the observation $\mathbf{o}_{V_k}$. It is constructed of two features that classify the route likelihood using the arc-centerline of the route. These features are

- the lateral distance $d_{\mathbf{o},\mathbf{x},V_k}$ of the observed position $\mathbf{o}_{\mathbf{x},V_k}$ to the centerline of route r_k, and
- the yaw angle deviation $\Delta\theta = \theta_{\tan,r_k}(\mathbf{x}_k) - \theta_k$ between the observed yaw angle θ_k of $\mathbf{o}_{V_k}$ and the tangent $\theta_{\tan,r_k}(\mathbf{x}_k)$ of the arc-centerline of route r_k at position $\mathbf{x}_k$ of $\mathbf{s}_{V_k}$.

Hence, $p(\mathbf{o}_{V_k}|\, \mathbf{s}_{r_k,V_k})$ is defined as

$$\begin{aligned} p(\mathbf{o}_{V_k}|\, \mathbf{s}_{r_k,V_k}) &= p(d_{\mathbf{o},\mathbf{x},V_k}|\, \mathbf{s}_{r_k,V_k}) \cdot p(\Delta\theta|\, \mathbf{s}_{r_k,V_k}) \\ &= \mathcal{N}(d_{\mathbf{o},\mathbf{x},V_k}|\, 0, \sigma_{d,\mathrm{PF}}) \cdot \mathcal{N}(\Delta\theta|\, 0, \sigma_{\Delta\theta,\mathrm{PF}}), \end{aligned} \tag{5.23}$$

where the standard deviations $\sigma_{d,\mathrm{PF}}$ and $\sigma_{\Delta\theta,\mathrm{PF}}$ have been guessed, based on simulation results. The features are depicted in Fig. 5.2.

A common practice to mitigate the particle deprivation problem is to increase the measurement model noise. Therefore, the standard deviations $\sigma_{\mathbf{f}_l,V_0,\mathrm{PF}}$, $\sigma_{v,V_0,\mathrm{PF}}$, $\sigma_{\mathbf{f}_l,V_k,\mathrm{PF}}$ and

(a) lateral distance $d_{\mathbf{o},\mathbf{x},V_k}$ (b) yaw angle deviation $\Delta\theta$

Figure 5.2.: Visualization of the two features for the route likelihood.

$\sigma_{v,V_k,\mathrm{PF}}$ are chosen to be larger than the standard deviations $\sigma_{\mathbf{x}}$, $\sigma_{\mathbf{v}}$, σ_θ used in the generative observation model.

Moreover, it should be noted that $\mathcal{Z}(o_{t+1}|\, s_{t+1})$ as defined above is not a true probability density function. In the form of Equation (5.20), $\mathcal{Z}(o_{t+1}|\, s_{t+1})$ rather is a function proportional to the true probability, which is sufficient to compute the particle weights, since due to normalization of the particle weights, the particle set becomes an approximation of the true density.

It would be interesting to investigate, whether including other features into $p(\mathbf{o}_{V_k}|\, \mathbf{s}_{r_k,V_k})$ (e.g. the velocity profile) would further improve the route intention estimation, which has been done in [SHLB18].

5.2.4. Reward Model

The ego vehicle seeks to maximize its reward. Hence, the reward model is designed, such that the vehicle drives with its reference velocity and avoids collisions and unnecessary accelerations. A behavior deviating from this desired behavior is punished with high negative rewards. Therefore, all terms of $\mathcal{R}(s_t, a_t, s_{t+1})$ are modelled as negative rewards:

$$\mathcal{R}(s_t, a_t, s_{t+1}) = R_{\mathrm{vel}}(\mathbf{s}_{V_0,t}) + R_{\mathrm{acc}}(a_t) + R_{\mathrm{coll}}(s_t, a_t, s_{t+1}). \tag{5.24}$$

The velocity reward term $R_{\mathrm{vel}}(\mathbf{s}_{V_0,t})$ penalizes deviations from a given reference velocity profile. Depending on the sign of the deviation, the reward is calculated by

$$R_{\mathrm{vel}}(\mathbf{s}_{V_0,t}) = \begin{cases} C_{v+}\ (v_0 - v_{\mathrm{ref}})^2, & \text{if } v_0 > v_{\mathrm{ref}} \\ C_{v-}\ \log\left(1 + (v_0 - v_{\mathrm{ref}})^2\right) & \text{otherwise} \end{cases}, \tag{5.25}$$

where a quadratic cost is applied if v_{ref} is exceeded and *Cauchy loss* otherwise [Bar19]. C_{v+} and C_{v-} are reward scaling factors. An asymmetric cost function is used, since driving slow or standing still might be an ordinary behavior in some cases (e.g. an occupied lane), whereas driving with higher speed is rarely acceptable.

The reference velocity can be given as velocity profile along the route $v_{\mathrm{ref},r_k}(l)$ which may depend on speed limit, road curvature and vehicle dynamics. For simplification, the reference velocity is set to a constant value v_{ref,V_0} in this work.

The quadratic acceleration reward term

$$R_{\mathrm{acc}}(a_t) = C_{\mathrm{acc}}\ a_t^2 \tag{5.26}$$

is appended to obtain a smoother velocity profile as it encourages the ego vehicle to drive with constant velocity.

In order to generate a collision free trajectory for the ego vehicle, the collision reward term $R_{\text{coll}}(s_t, a_t, s_{t+1})$ returns a high negative reward in case a collision with a surrounding vehicle occurs. Thus, it is defined as

$$R_{\text{coll}}(s_t, a_t, s_{t+1}) = \begin{cases} 0 & \text{no collision} \\ C_{\text{coll}} & \text{ego vehicle collides || unsafe action} \end{cases}, \tag{5.27}$$

where the negative collision cost C_{coll} is chosen to be by magnitudes larger than the other scaling factors, i.e. $C_{\text{coll}} \ll C_{v+}$.

Collisions are detected, by modelling each vehicle as a single sphere and checking if the moving spheres do intersect during a state transition [Eri05, p.223]. The radius of each sphere is chosen to be half the width of the respective vehicle, with an additional safety offset $d_{\text{safe,offs}}$. To further improve the quality of the collision checks, multiple spheres in longitudinal direction could be used.

The IDM is reported to generate strong breaking maneuvers, if the input quantities such as for e.g. the bumper-to-bumper distance d_{bump} change in a non-continuous way, as it is the case for "cut-in" maneuvers on highways [KTH10]. In urban scenarios considered in this work, when the automated vehicle merges onto lane before another vehicle at a T-intersection or when it enters a roundabout the bumper-to-bumper distance of the other vehicle to the ego vehicle can change non-continuously as well. Therefore, in this case the other vehicle is likely to perform strong or even emergency braking, which is not desired. Hence, actions of the ego vehicle, where other vehicles must react to with strong braking are marked as unsafe similar to [SK20] and treated analog to collisions in terms of returned reward.

In particular, if the acceleration of the IDM $a_{\text{IDM}} = a_{ref,k} + a_{int,k}$ for another vehicle V_k is less than a given emergency braking threshold $a_{\text{th,emergency}}$ ($a_{\text{IDM}} < a_{\text{th,emergency}}$), the action a_t is marked unsafe and the negative collision cost C_{coll} is applied.

As soon as a collision has occurred or an unsafe action has been taken, the posterior state s_{t+1} is a terminal state, and will not be considered in future transitions or belief updates. When all particles of a particle set are collision particles, the belief is terminal.

In this work the rewards are discounted, which means the discount factor γ is set to be less than 1 in order for the rewards to remain finite even for infinite horizons [RN10, p.650].

So far, only rewards based on states and no information rewards are considered (see Ch. 4). Including belief dependent rewards based on information measures remains open for future work.

5.3. Implementation Details

In the scope of this thesis, the Particle Filter Tree algorithm presented in Chapter 4 and the POMDP model formulated in the previous section (Sec. 5.2) were implemented in C++ from scratch and are evaluated in the simulation environment "P3IV: Probabilistic Prediction and Planning Simulator for Intelligent Vehicles" [OcT20]. More details about the source code and a link is provided in the Appendix A.2. As this simulator is written in Python, the library pybind11 is used to create the necessary bindings for the C++ packages [JRM17]. Hence, the interface of the POMDP motion planner (which consists of the solver and the POMDP model implementation) is exposed to the Python side.

The planner takes as inputs all information necessary to feed the POMDP model as introduced in Sections 5.1 and 5.2. This contains the position, velocity, pose and the size of all relevant vehicles in the scene, as well as the routes attached with some regulatory information, such as speed limits for example.

The output of the motion planner is a sequence of optimal acceleration values, which is integrated to obtain a velocity profile over time. This velocity profile is then returned to the simulation environment, where the planned actions are executed for the ego vehicle and the other vehicles are steered by driver models or recorded real world data.

5.3.1. Map Data and Environment Update

In the simulation environment, the road map is internally represented by Lanelets [BZS14, PPJ+18]. Since the route of the other vehicles is encoded in only one dimension in the state space (see Eq. (5.3)), the map layout must be preprocessed, such that for every vehicle based on their current position on the map and the maps routing graph, a list of route options is returned. This is done in the simulators' 'Understanding' module, which takes care of all the map processing and returns the route options as centerlines. In the overall processing pipeline, the Understanding module is always executed before the belief state update of the POMDP planner in every time step.

For the POMDP planner to be used in automated vehicles, it must be able to deal with dynamic changes in the vehicles' environment (see Sec. 5.1). The changes or environment updates considered in this work are

- the detection of a new vehicle in the scene, that must be considered in planning,
- the exiting of a vehicle from the scene of the ego vehicle, and
- the change of the route options of a vehicle in the scene (for example if it has passed an intersection).

If the POMDP planner receives information from the understanding module about a newly detected vehicle, it initializes the belief about this vehicle and includes the sampled state representation in the other states in the current belief state. Similarly, if the observation for the belief update does not include a vehicle, which has been in the state representation before, the vehicle is simply removed from the belief and not considered any further. In case the possible route options of a vehicle in the scene have changed from time step to time step, the POMDP planner resamples only the route option variable r_k in the state representation (see Eq. (5.3)) is resampled, based on the observation and the remaining state information.

The initialization of a new vehicle and resampling of the route index r_k will be explained in more detail in the next section.

5.3.2. Belief Initialization

The generation of particles or state samples for the initial belief state is similar to observation generation described in Section 5.2.3. The difference is that instead of generating observations given a posterior state, the initial states $s_{t=0}$ need to be sampled given an initial observation $o_{t=0}$ as defined in Section 5.2.1.

Hence, the ego vehicle states $\mathbf{s}_{V_0}$ are sampled analog to Equation (5.18) from Normal distributions with equivalent standard deviations, but the observation as mean:

$$\mathbf{s}_{V_0,t=0} = \begin{pmatrix} \mathbf{x}_0 \\ v_0 \end{pmatrix} \sim \begin{pmatrix} \mathcal{N}(\mathbf{o}_{V_0,\mathbf{x}}, \Sigma_{\mathbf{x},V_0}) \\ \mathcal{N}(\mathbf{o}_{V_0,v}, \sigma_{v,V_0}) \end{pmatrix}. \tag{5.28}$$

As the position $\mathbf{x}_0$ sampled in this way is not on the route r_0 of the ego agent, $\mathbf{x}_0$ is the point on the route, which is closest to the sampled position.

Generating initial states for the other vehicles $\mathbf{s}_{V_k}$ is different from those for the ego vehicle, since their state is not fully observable. Therefore, only the observable variables position $\mathbf{x}_k$ and velocity v_k are sampled in the same way as for the ego vehicle:

$$\mathbf{s}_{V_k,t=0} = \begin{pmatrix} \mathbf{x}_k \\ v_k \end{pmatrix} \sim \begin{pmatrix} \mathcal{N}(\mathbf{o}_{V_k,\mathbf{x}}, \Sigma_{\mathbf{x},V_k}) \\ \mathcal{N}(\mathbf{o}_{V_k,v}, \sigma_{v,V_k}) \end{pmatrix}. \tag{5.29}$$

The hidden variable r_k is sampled from a discrete probability distribution $p(r_k = r_k^{(i)} |\, d_{\mathbf{o},\mathbf{x}},\, \Delta\theta)$, which depends on the lateral distance feature $d_{\mathbf{o},\mathbf{x}}$ and the angle difference feature $\Delta\theta$ as introduced in Section 5.2.3:

$$p(r_k = r_k^{(i)} |\, d_{\mathbf{o},\mathbf{x}},\, \Delta\theta) = \frac{p(d_{\mathbf{o},\mathbf{x}} |\, \mathbf{s}_{r_k^{(i)}}) \cdot p(\Delta\theta |\, \mathbf{s}_{r_k^{(i)}})}{\sum_{l=1}^{N_{\mathcal{R}_k}} p(d_{\mathbf{o},\mathbf{x}} |\, \mathbf{s}_{r_k^{(l)}}) \cdot p(\Delta\theta |\, \mathbf{s}_{r_k^{(l)}})}, \tag{5.30}$$

where $p(d_{\mathbf{o},\mathbf{x}} |\, \mathbf{s}_{r_k^{(i)}})$ and $p(\Delta\theta |\, \mathbf{s}_{r_k^{(i)}})$ are assumed to be normal distributed analog to Equation (5.23).

5.3.3. Rollout Policy

During tree search, the MCTS based online POMDP solver (see Ch. 4) selects actions according to UCB until a leaf node is reached, which has not been visited before. In order to guess the value of this node a default or rollout policy $\pi_{rollout}$ is used for action selection (see Sec. 3.2.1). In general, the closer this rollout policy is to optimal behavior, the faster the algorithm will converge to the optimal solution. Moreover, as the rollout is performed at the end of every episode it should also be quick and easy to execute, to obtain a deep search tree. Hence, finding a good rollout policy is crucial for the performance of the planning algorithm.

Even though it is possible to use rollouts based on other search algorithms, a constant-velocity rollout for the ego vehicle is applied in this work [HSB+18, HSX+18].

The Particle Filter Tree algorithm simulates belief trajectories instead of state trajectories in each episode by performing multiple Particle Filter updates with simulated observations (see Ch. 4). This means that every rollout starts from the belief (which is approximated by a particle set) at a leaf node. Therefore, rollouts also use particle filter updates in combination with the constant-velocity action selection strategy and simulated observations.

Another option would be to find a single state that represents the belief at the leaf node appropriately (either by sampling or averaging) and use this state to perform a state-based rollout.

The rollouts are performed up to a maximum time horizon T or until a terminal belief has been reached (see Sec. 5.2.4). In all experiments the maximum time horizon is set to $T = 5s$.

5.3.4. Particle Filter

Particle Filters use resampling to avoid the particle degeneracy problem, where all the weight is concentrated on a small portion of particles after a few iterations (see Sec. 2.3.3). In general, resampling can take place in every iteration or only if the particle set is degenerated, as denoted in Algorithm 2.5.

In this work, the particle filter is implemented, such that resampling is performed in every iteration. This means that at every time step and in every V-node the respective particle set consists of equally weighted particles.

In addition, in every time step, a small portion of particles is reinvigorated or replaced by newly sampled particles, where the number of particles to be replaced depends on the ratio between the maximum weight in the particle set and the maximum possible weight, given an observation. The new particles are generated in the same way as during belief initialization (see Sec. 5.3.2).

6. Evaluation

In this chapter, the particle filter and the POMDP planner developed within this work will be evaluated. It will be demonstrated that the particle filter is able to track the belief state and estimate the intention of other drivers successfully, and that the POMDP planner generates collision free and interactive driving behavior for urban scenarios.

The feasibility of POMDP planners for behavior generation of automated vehicles has been shown before, as for e.g. in the evaluation of the work used as main reference [HSB+18]. However, in their work Hubmann et al. did not analyze the sensitivity of the generated trajectories on variations in parameters such as for e.g. the runtime or the exploration constant of the planner.

Therefore, in this work the influence of parameter variations of the PFT POMDP solver and the POMDP model on planned trajectories, the Q-function estimate and the convergence will be investigated further.

The evaluation of the planning algorithm in this thesis is threefold: At first, the belief state tracking and the driver intent estimation with the particle filter will be evaluated in Section 6.2. Then, in Section 6.3 the PFT POMDP solver's sensitivity to parameter variations will be investigated and based on this analysis an optimal parameter selection is made. Finally in Section 6.4, the impact of different IDM parameters in the POMDP model on the generated trajectories are analyzed.

6.1. Driving Scenarios and Experimental Setting

For the evaluation of the POMDP planner the Probabilistic Prediction and Planning Simulator for Intelligent Vehicles P3IV is used [OcT20]. Since the simulator is still being developed at FZI Research Center in Karlsruhe, the source code has not been published yet.

The simulator uses scenarios and traffic data from the interaction dataset and therefore provides a realistic environment to evaluate behavior and motion planners [ZSW+19]. The interaction dataset is a record of real-world traffic and contains motions of various traffic participants in multiple highly interactive driving scenarios from different countries. Thus, it is used for behavior-related research in the domain of automated driving.

In this work only the roundabout scenario `DR_DEU_Roundabout_OF` is used for the evaluation and demonstration of the POMDP planner. In the dataset every vehicle is assigned a

unique ID, which will be used in this work to refer to the respective vehicles. The parameters of the POMDP model and the solver are given in tables A.1 and A.2 in the Appendix A, if not mentioned differently.

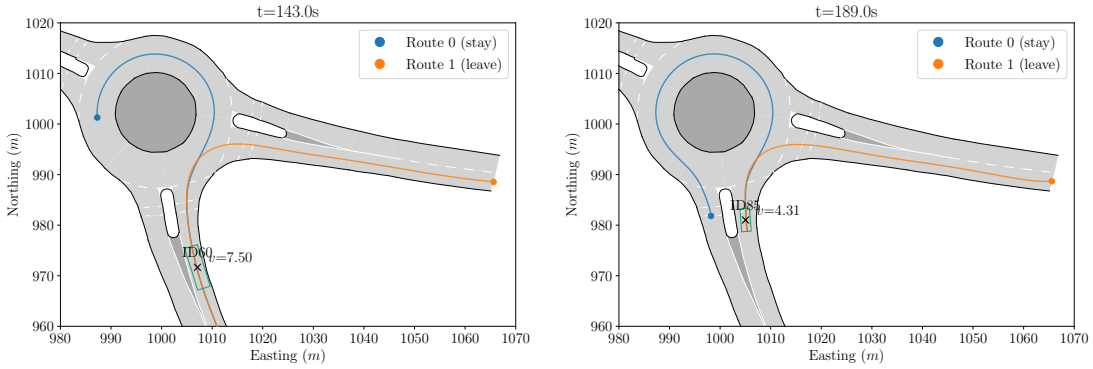
According to German traffic rules, vehicles in a roundabout have the right of way, which means that other vehicles trying to enter the roundabout must yield to the vehicles already in the roundabout. However, the POMDP planner is not given any prior knowledge about traffic rules, which means they are not considered in the experiments.

The simulation environment is running on the same system as the planner. It reads all the scene data provided in the data set and performs all the necessary preprocessing (as explained in Sections 5.1 and 5.3). All experiments were performed on an Intel Core i7-8850H CPU with a base frequency of 2.60GHz.

6.2. Driver Intent Estimation

In this section, the driver intent estimation of the particle filter is evaluated. The particle filter uses the definition of the state and observation space from Section 5.2.1 and the transition and observation model from Sections 5.2.2 and 5.2.3. All evaluations in this section were performed with a particle set size of 10000 particles.

The performance will be assessed in two different scenarios from `DR_DEU_Roundabout_OF` (see Fig. 6.1). In the first scenario (Fig. 6.1(a)) from timestamp `143000`[1] vehicle ID60 is on Route 0 and stays in the roundabout at the first exit. The second scenario (Fig. 6.1(b)) is taken from timestamp `189000`, where vehicle ID85 follows Route 1 and leaves the roundabout at the first exit.



(a) Vehicle ID60 follows Route 0

(b) Vehicle ID85 follows Route 1

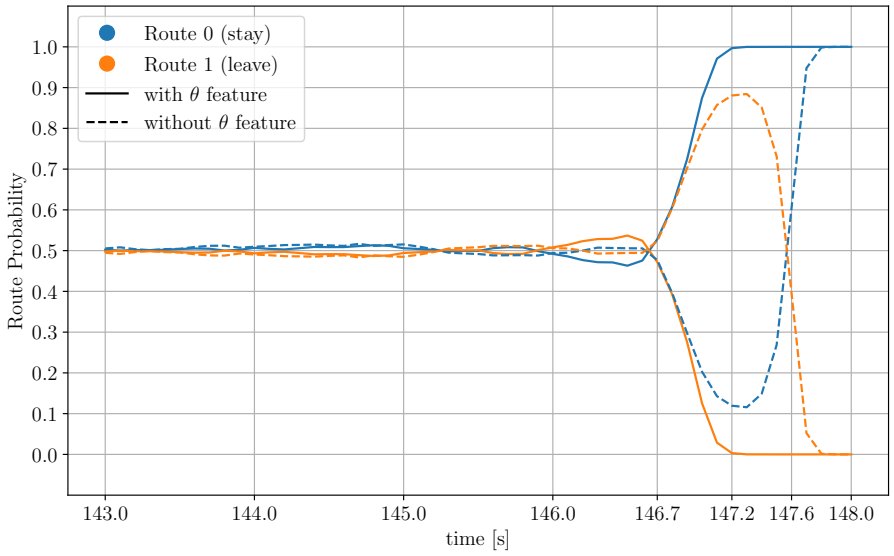
Figure 6.1.: The two evaluation scenarios for driver intent estimation from scenario `DR_DEU_Roundabout_OF` at timestamp `143000` and `189000`.

Results

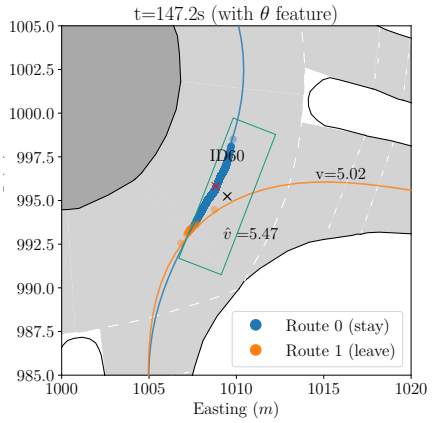
Figures 6.2 and 6.3 show the estimated route probability over time, as well as representative particle set approximations at times, where the route intention becomes obvious. For both scenarios the intention estimation is performed once with the yaw angle feature and once without it (see Sec. 5.2.3).

In the first scenario where vehicle ID60 stays in the roundabout including the yaw angle feature greatly improves the route intention estimation, as the particle filter is able to

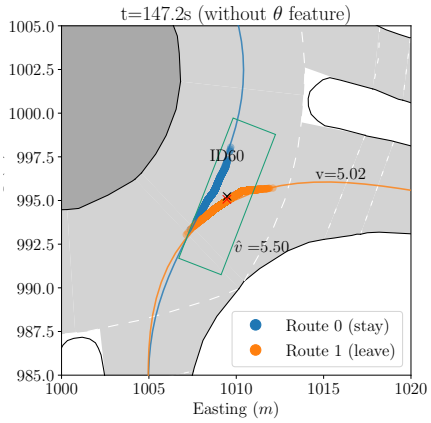
[1] Timestamps are given in milliseconds.



(a) The estimated route probability for Vehicle ID60.

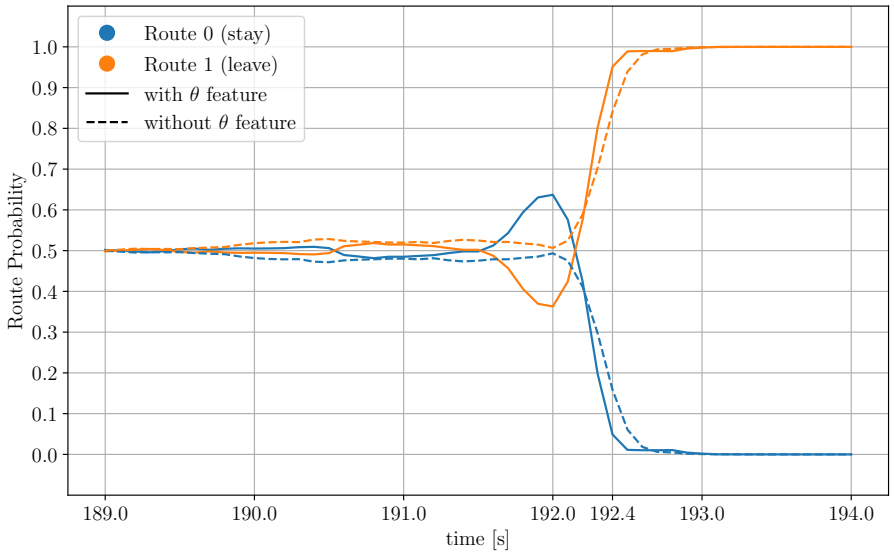


(b) Particle set with yaw angle feature.

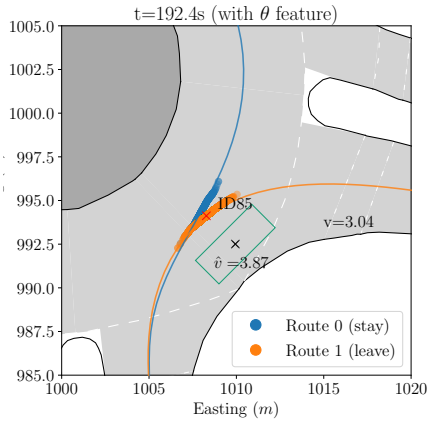


(c) Particle set without yaw angle feature.

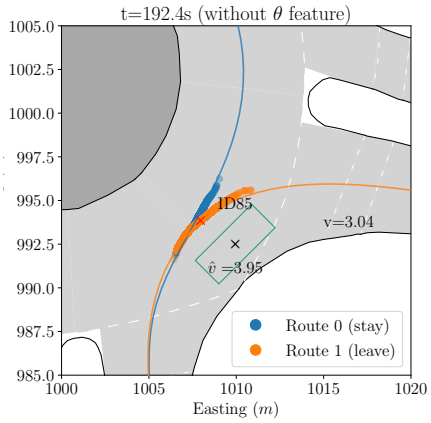
Figure 6.2.: The route intention estimation for a vehicle staying in the roundabout (making a left-turn).



(a) The estimated route probability for Vehicle ID85.



(b) Particle set with yaw angle feature.



(c) Particle set without yaw angle feature.

Figure 6.3.: The route intention estimation for a vehicle leaving the roundabout (making a right-turn).

estimate the right route intention approx. 0.8s earlier than without the yaw angle feature (see Fig. 6.2(a)). Moreover, without it, the estimation is not able to estimate the true intention, as it assigns a high probability to the wrong intention for a long time (see peak at 147.2s in Fig. 6.2(a)). The reason for the wrong estimation is that at 147.2s, the lateral distance to the wrong Route 1 is close to zero (see origin of ID60 in Fig. 6.2(c)) and hence the lateral distance classifier assigns high probability to particles with the wrong route variable.

For the second scenario where vehicle ID85 leaves the roundabout at the first exit, the performance improvement is not as large as before, as vehicle ID85 drives far from the route centerline and the orientation of the route centerlines up to 192.4s does not differ very much (see Fig. 6.3(a)). The reason for peak at 192.0s for the estimation with the yaw angle are unsmooth route centerlines returned by the Lanelet2 library [PPJ+18]. The peak is also visible but less noticeable in Fig. 6.2(a) between time 146.0s and 146.7s.

6.3. Solver Parameter Analysis

The POMDP motion planner uses the Particle Filter Tree algorithm introduced in Chapter 4 to solve the POMDP, which has been formulated in Section 5.2. In order to apply this online POMDP solver, various parameters need to be selected such that the solver finds a good approximation to the optimal policy for the POMDP in limited runtime.

Therefore, in this section, the sensitivity of the algorithm to changes in the parameters search time (runtime) T_{run}, exploration constant c, and the number of particles used for tree search m is analyzed. Moreover, at the end of this section by varying the parameter k_{obs} it will be examined whether Progressive Widening on the observation space is useful for the PFT algorithm (see Section 4.2).

Objective

Typically, planning algorithms must meet hard real-time constraints for the application in automated vehicles. This means, that runtime available for trajectory planning is very limited in a cars control loop. The presented algorithm is very computationally complex as it finds a policy for the automated vehicle by multiple random simulations and improves its solution with additional runtime. Therefore, one must trade-off between solution quality (e.g. long planning horizons) and replanning frequency.

In their work, Hubmann et. al limit the runtime available for planning to 1s and use a step size of $\Delta t = 1s$ in the transition model in order to obtain trajectories for long planning horizons of 6s to 8s [HSB+18]. However, urban scenarios are highly uncertain and predictions become very unreliable for long planning horizons. Thus, in this work the planning horizon is limited to 5s and a smaller step size of $\Delta t = 0.5s$ is used in the transition model, while keeping the runtime limit equal to $T_{run} = 1s$.

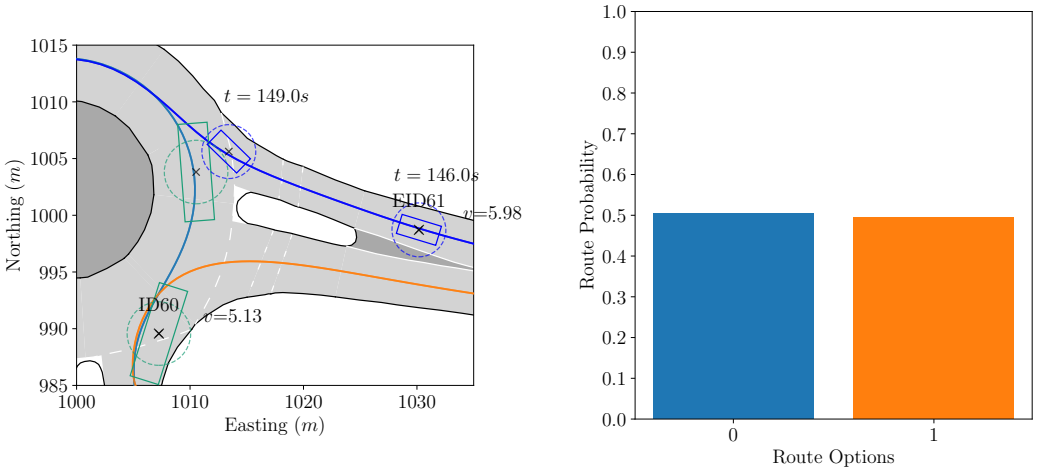
Within these constraints, the influence of the solver parameters on the planned trajectory will be investigated. The objective is to find a suitable parameter set, such that the algorithm finds a good approximation to the optimal policy within one second (1Hz replanning frequency).

Evaluation Scenario

The scenario used for the solver parameter analysis is shown in Fig. 6.4. It is taken from the roundabout setting `DR_DEU_Roundabout_OF` at timestamp `146000` in the interaction dataset. Initially, the ego vehicle (ID61) is driving at speed $v = 6\frac{m}{s}$ and the other vehicle (ID60) at approximately $v = 5\frac{m}{s}$. The approximate time-to-collision of the ego vehicle

is $3s$ if the other vehicle follows Route 0 (blue route). This can be seen in Fig. 6.4(a), where the position of the cars at $t = 149.0s$ is drawn for the ego vehicle moving with constant speed and the other vehicle according to the recorded data. The initial belief of the planner about the route intention of vehicle ID60 is depicted in Fig. 6.4(b).

For the solver, this scenario is challenging as the collision is relatively far in the future and not certain, such that a deep search tree of at least 6 levels is necessary to find a good policy, that contains knowledge about the potential collision.



(a) Evaluation scenario

(b) Initial route probabilities of vehicle ID60

Figure 6.4.: The evaluation scenario `DR_DEU_Roundabout_OF` at timestamp `146000` and at $t = 149.0s$ three seconds later, if the ego vehicle EID61 keeps driving at constant speed with $v = 6\frac{m}{s}$ and vehicle ID60 behaves according to the recorded data.

Approach

In contrast to standard POMDP benchmark problems such as Light Dark or Laser Tag, there is no benchmark available for the self-defined automated driving POMDP, to select the parameters according to. Hence, in this work the parameters are selected based on the shape of the search tree, the Q-function estimate and the convergence of the Q-value for each action on the first tree level.

Therefore, in the following subsections (Figs. 6.5, 6.6, 6.7, 6.8), three plots are evaluated for every selected parameter combination:

The left plot shows the convergence of the Q-function estimate $\widehat{Q}(b, a)$ for all actions on the first tree level. Thus, Q-values are plotted over runtime.

The center plot shows all sampled trajectories contained in the search tree. Each acceleration sequence is integrated in order to obtain a velocity profile, which is then plotted over time with high transparency. During tree search many action sequences are visited several times. Thus, the darker the line in the plot, the more often the corresponding action sequence has been visited. Trajectories which have led to collisions are plotted less transparent in red and collision-free trajectories are plotted in green. Additionally, the thick red dashed line is the reference velocity of the ego-vehicle and the thick blue line the optimal trajectory returned by the planner.

The right plot shows the Q-function estimate $\widehat{Q}$ as a contour plot in a time-velocity diagram. This plot is obtained by extracting the Q-value of every Q-node in the search tree. The x and y coordinate (which correspond to time and velocity values in this case) of every value is the end point of the integrated action sequence leading to the corresponding Q-node. Put differently, the $\widehat{Q}$ plot emerges from the sampled trajectories plot, by plotting the Q-value at the end of each action sequence. Due to the equidistant acceleration spacing, different action sequences have equal endpoints in time-velocity space. Therefore, at those points the mean of all Q-values is depicted in the plot. The color scale ranges from red for low or high negative Q-values (potentially arising from collisions) to green for high or low negative rewards. Regions in the time-velocity space, which could not be explored to limited action space or runtime limits are blanked out. Finally, the thick red dashed line and the thick blue line are added analog to the sampled trajectory plot in the middle.

Runtime T_{run}

At first, the influence of the runtime is examined in general. For that, with an initial guess for the parameters number of search particles m, exploration constant c and the observation widening factor k_{obs} the runtime T_{run} is varied. The values are given in Tab. 6.1.

Parameter	Value
m	5
c	5000
k_{obs}	0.5
T_{run}	$\{0.5, 1, 5, 10\}s$

Table 6.1.: Parameters for runtime evaluation.

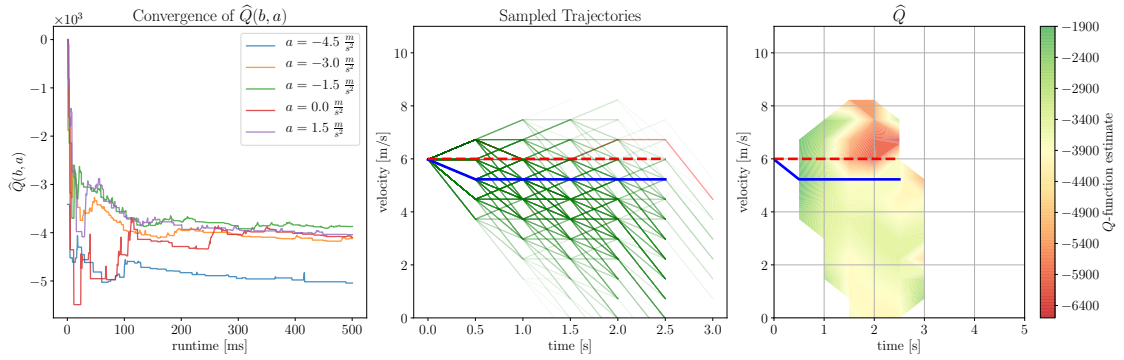
Results

The results for the different runtimes are depicted in Fig. 6.5. In general, it can be seen, that with longer runtime the search tree gets deeper and the Q-function estimate becomes smoother with fewer peaks. This manifests, that indeed with more runtime the algorithm improves its solution. The reason for this is, that over time more episodes, i.e. trajectories are sampled, which can be seen in the middle plots.

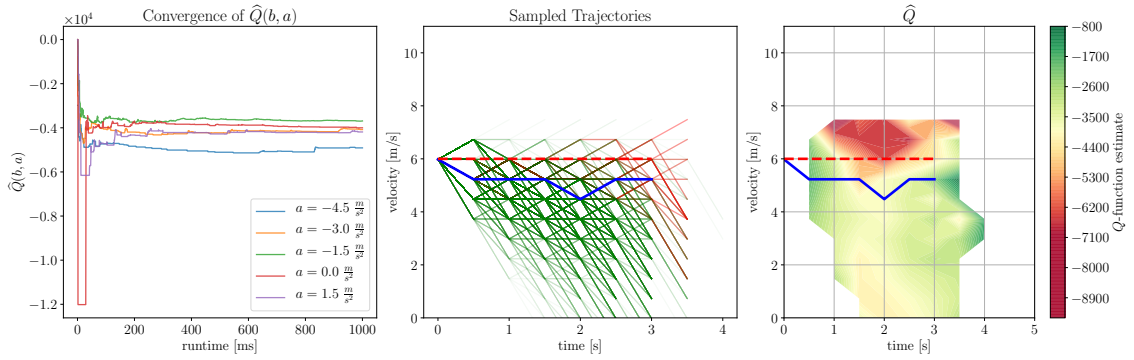
In this scenario, the algorithm already detects the potential collision with only 500ms runtime, even though the solution has a high variance and planning horizon is very limited. For long runtimes (5 and 10s), the algorithm is able to plan for the full 5s time horizon and finds the critical time interval where the collision is likely. However, as can be seen in the left plots, the Q-value of the actions on the first level do not vary strongly after a runtime of 1s. Hence, it can be concluded that even though the algorithm is not able to plan for the full 5s time-horizon, it has most likely converged to a feasible solution and that a runtime of 1s is a good trade-off between solution quality and computational resources.

Exploration Constant c

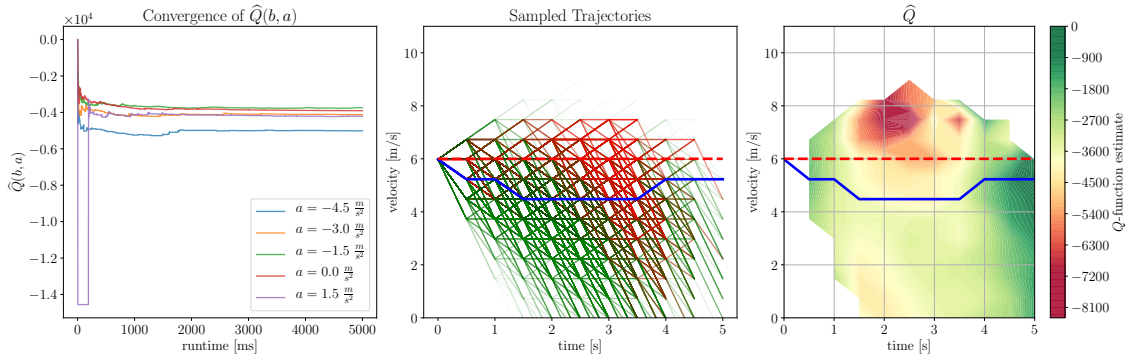
Next, the impact of the important parameter c, which is the exploration constant that serves as a method to trade-off exploration and exploitation is investigated further. In general, low values for c make the algorithm favor exploitation over exploration. This means, once it has found a policy (i.e. a trajectory) that yields reasonable rewards, it will stick to this policy and not explore other possible trajectories. For high values in the



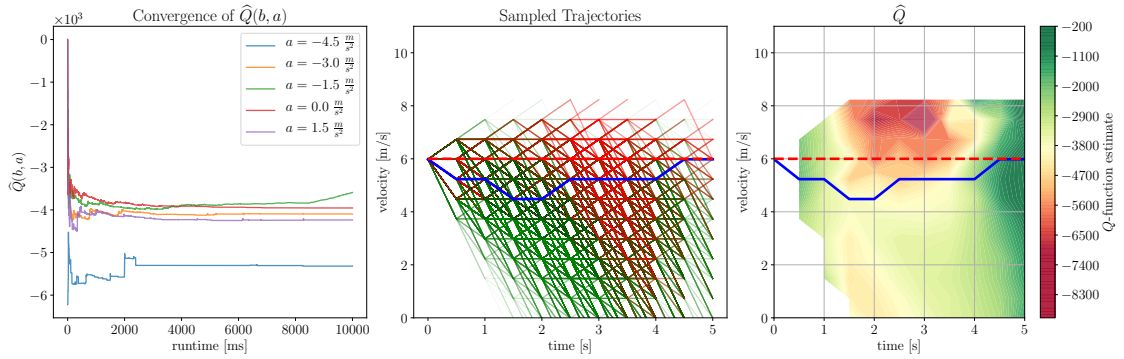
(a) Results for runtime T_{run}=500ms.



(b) Results for runtime T_{run}=1000ms.



(c) Results for runtime T_{run}=5000ms.



(d) Results for runtime T_{run}=10000ms.

Figure 6.5.: Planning results for the evaluation scenario with $m = 5$ tree search particles, an observation widening factor $k_{obs} = 0.5$ and a UCB constant $c = 5000$.

parameter c, the planner favors exploration over exploitation. Hence, it always explores untried actions and trajectories first.

With this in mind, the planning results are now evaluated for different c values, where the reward constants for velocity and collision served as guideline for initial guesses (see Sec. 5.2.4). In this experiment, the runtime is chosen to be T_{run}=5s in order to obtain deeper trees and lower variance in the optimal trajectory. The remaining parameters are given in Tab. 6.2.

Parameter	Value
m	5
c	$\{500, 5000, 10000, 50000\}$
k_{obs}	0.5
T_{run}	$5s$

Table 6.2.: Parameters for exploration constant evaluation.

Results

The results for different exploration constants c are shown in Fig. 6.6. It should be noted, that finding the optimal parameters in this setting is very difficult, as the planner is based on many random samples and therefore does not yield the same result for multiple runs. Hence, the plots in Fig. 6.6 for each c value can be regarded as "one out of many similar" solutions, that give an intuition about the influence of parameter c on the tree search.

For low values of c such as in Fig. 6.6(a), the resulting search tree is very narrow (but deep), as the algorithm does not explore many other trajectories once it has found a collision-free trajectory. Moreover, with this low c, the planner is not able to find an acceptable approximation of the Q-function (see right plot in Fig. 6.6(a)), as it does not explore the region with high collision probability well enough.

By increasing c sufficiently (see Fig. 6.6(b) and 6.6(c)), the planner is able to find a better approximation to the Q-function in high collision probability regions. However, this comes at the cost of a reduced tree depth.

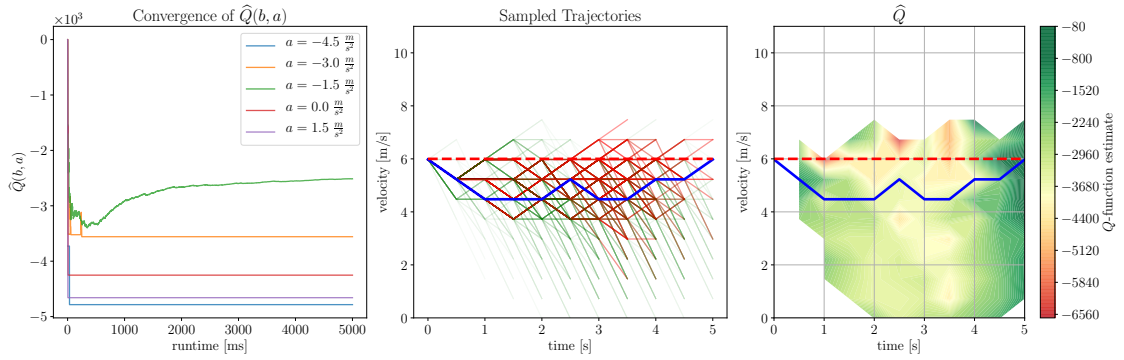
Even larger c (see Fig. 6.6(d)) do not improve the Q-function approximation as they lead to an "over-explorative" planner, which is not capable of planning for longer horizons in limited runtime. However, in case the planner had enough runtime or computational resources, the generated trajectories would be less conservative (see Fig. 6.6(c) and 6.6(d)).

Based on these observations, one can conclude that the "optimal" c lies in or around the interval from c=5000 to c=10000 (for the reward parameters chosen as in Tab. A.1) and that reducing c leads to deeper trees and hence longer planning horizons.

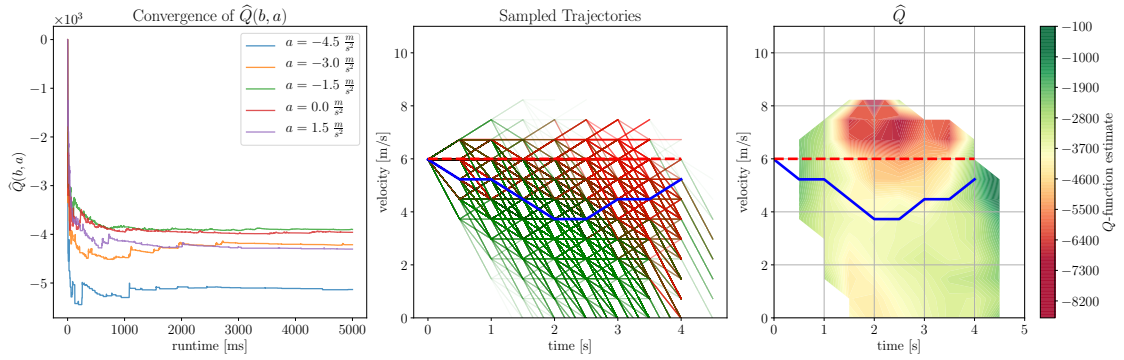
Number of Particles in Tree Search m

Another important and special parameter of the Particle Filter Tree algorithm (see Ch. 4.2) is the number of particles used in the particle sets during tree search m. This parameter has great influence on the number of sampled episodes (i.e. calls to SIMULATE()) during tree search, since every episode consists of many particle filter updates. However, it is known that probability distributions are approximated better with larger particle sets. Hence, the question that is addressed in this section is whether it is beneficial to use less, but more accurate episodes to generate an optimal trajectory.

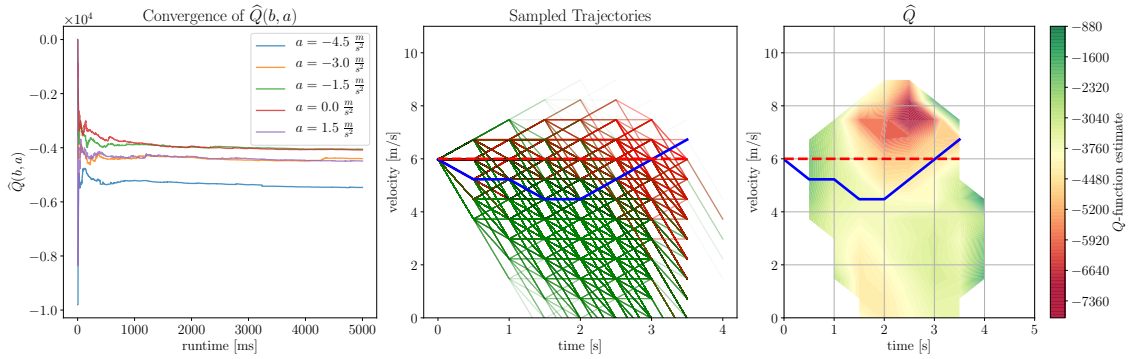
For that, the runtime T_{run} is chosen to be 1s, as then the impact on computing time for each episode becomes more obvious. The other parameters are selected according to Tab. 6.3.



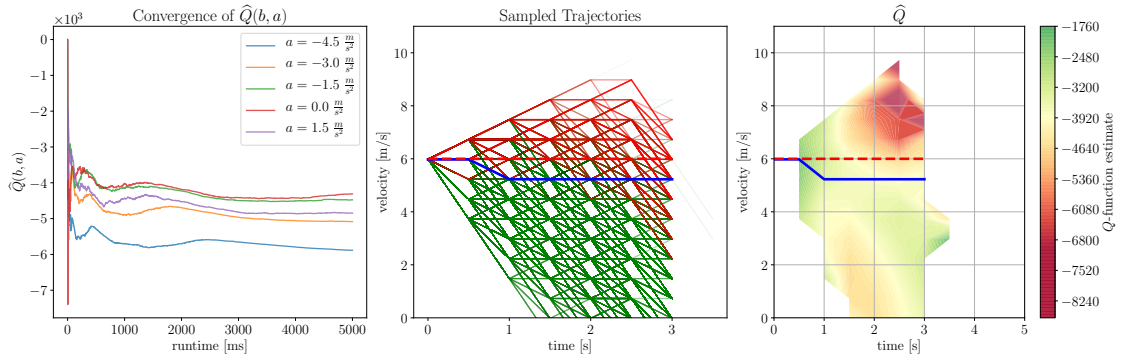
(a) Results for UCB constant c=500.



(b) Results for UCB constant c=5000.

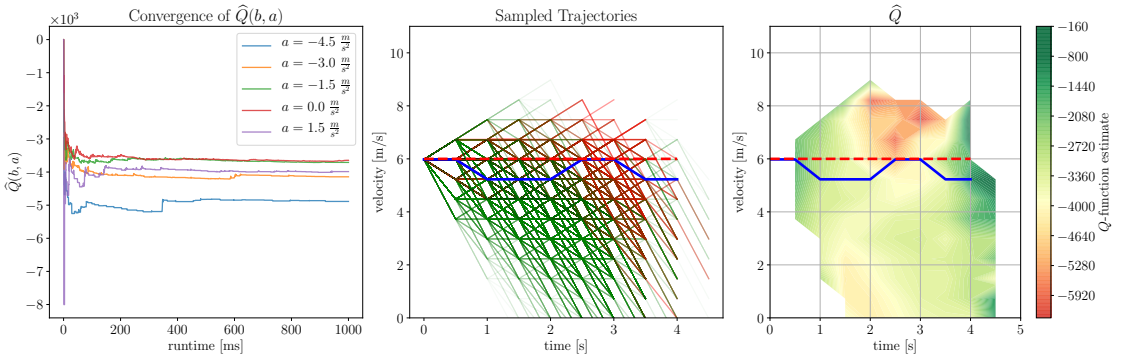


(c) Results for UCB constant c=10000.

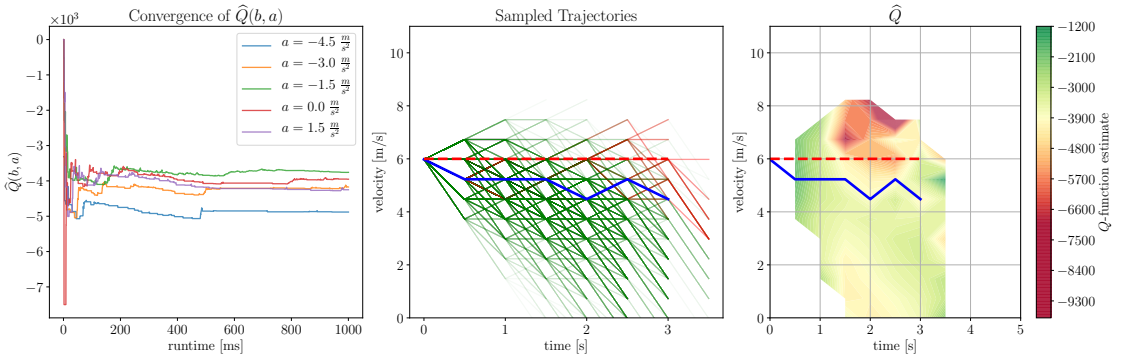


(d) Results for UCB constant c=50000.

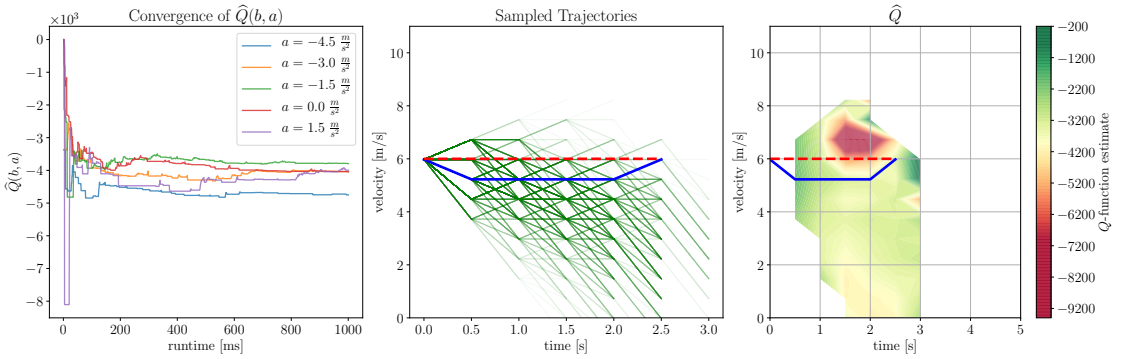
Figure 6.6.: Planning results for the evaluation scenario with $m = 5$ tree search particles, an observation widening factor $k_{obs} = 0.5$ and a runtime $T_{run} = 5000$ms.



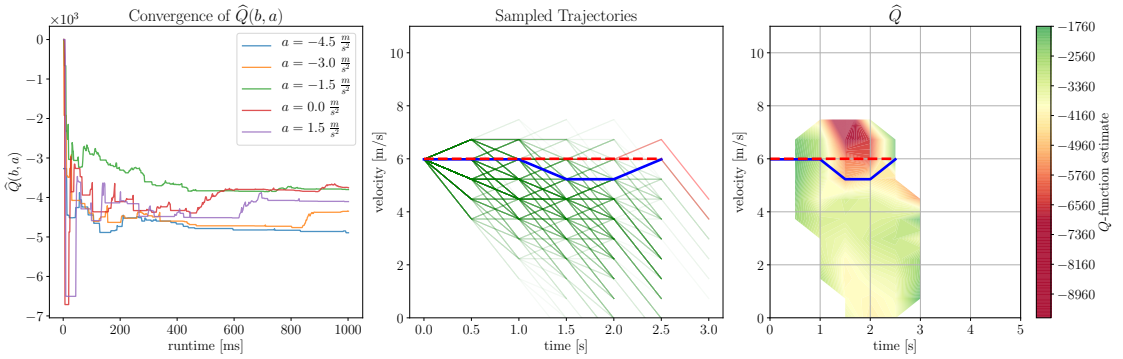
(a) Results for number of search particles m=1.



(b) Results for number of search particles m=5.



(c) Results for number of search particles m=10.



(d) Results for number of search particles m=20.

Figure 6.7.: Planning results for the evaluation scenario with a UCB constant $c = 5000$, an observation widening factor $k_{obs} = 0.5$ and a runtime $T_{run} = 1000$ms.

Parameter	Value
m	$\{1, 5, 10, 20\}$
c	5000
k_{obs}	0.5
T_{run}	$1s$

Table 6.3.: Parameters for number of search particles evaluation.

Results

The results are depicted in Fig. 6.7, where the number of particles m are in the range from 1 to 20.

In the extreme case where each particle set is represented by only one particle (Fig. 6.7(a)), the search tree contains many nodes and is very deep, as more episodes could be sampled during runtime. As expected, an increase in the particle set size results in fewer episodes being sampled and hence shorter planning horizons, but a more accurate definition of the collision region (see Figs. 6.7(c) and 6.7(d)). However, this is likely to be an over-confident result, as particle sets with sizes of 10 or 20 particles are usually not capable of approximating the complex probability distributions sufficiently well.

Therefore, it can be concluded that using to many particles is disadvantageous, since the computational effort for tree search is increasing much faster than the increase in approximation quality of larger particle sets and that in general it is better to have more episodes than larger particle sets. However, using more than one particle (but not too many) yielded better results and can be regarded as good trade-off approximation accuracy and the number of sampled episodes (see Fig. 6.7(b)). Hence, in this work, a particle set size around five particles will be used for subsequent experiments.

Progressive Widening of the Observation Space k_{obs}

At last, it should be investigated, whether progressive widening still improves the solution even though no particles or observations are reused. In Section 4.2 it was explained that for this algorithm progressive widening might not be beneficial. Hence, in this section, the influence of progressive widening on the search tree and Q-function estimate is examined.

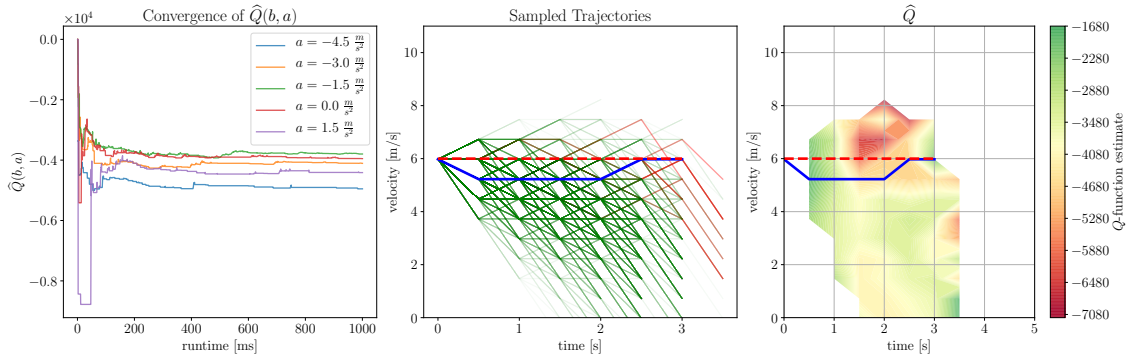
For this, the results with an observation widening factor $k_{obs} = 0.5$ are compared with $k_{obs} = 3.0$ and $k_{obs} = 5.0$, where setting $k_{obs} < 1$ is analog to giving up progressive widening as in this case ($\alpha_{obs} = 0.05$) the number of child nodes is always limited to 1 (see Sec. 3.2.3). The other parameters are chosen according to Tab. 6.4.

Parameter	Value
m	5
c	5000
k_{obs}	$\{0.5, 3.0, 5.0\}$
T_{run}	$1s$

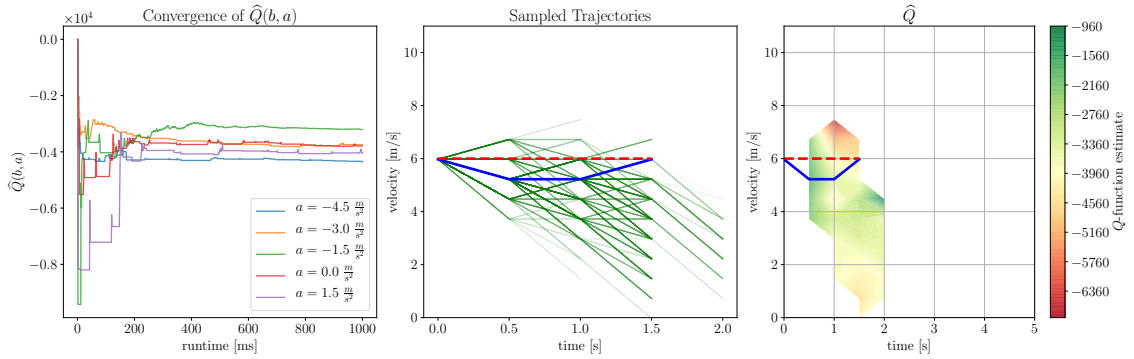
Table 6.4.: Parameters for progressive widening evaluation.

Results

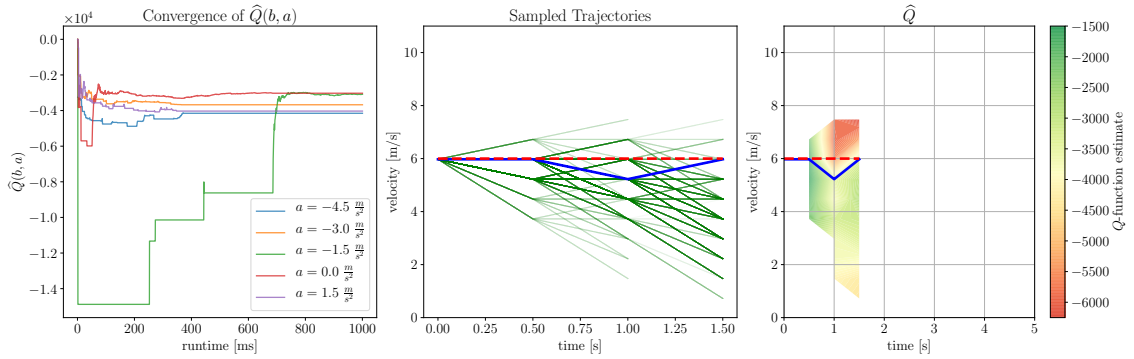
The results are shown in Fig. 6.8. It is unambiguous, that applying progressive widening only deteriorates the solution, as it limits the tree from growing deep (see Figs. 6.8(b) and 6.8(c)). The reason for this is that every action node (Q-node) has several observation



(a) Results for observation widening factor k_{obs}=0.5.



(b) Results for observation widening factor k_{obs}=3.0.



(c) Results for observation widening factor k_{obs}=5.0.

Figure 6.8.: Planning results for the evaluation scenario with $m = 5$ tree search particles, a UCB constant $c = 5000$, and a runtime $T_{run} = 1000$ms.

nodes (V-nodes), such that during search there are more observation nodes than action nodes added to the tree. On each expansion of such a node the recursion of SIMULATE() stops and the result are many short episodes.

Thus, the planning algorithm in this work does not use progressive widening.

Summary

In this section, the influence of the parameters runtime T_{run}, exploration constant c, number of search particles m and the observation widening factor k_{obs} have been investigated. Now, the findings are summarized, shortly.

In Fig. 6.5, it could be observed, that the longer the given runtime for the solver is, the deeper the tree and the better the Q-function estimate becomes. Even though one second is quite short, it is a reasonable trade-off for the scenario used in this evaluation.

Fig. 6.6 showed, that exploration is computational complex. Hence, when increasing the UCB exploration constant c the algorithm should also be given more runtime. A good range to choose the exploration constant from is inside and around the interval $\frac{C_{coll}}{2}$ and C_{coll}. It was also observed, that when runtime is limited, reducing the exploration constant leads to longer planning horizons.

Based on Fig. 6.7, it can be concluded, that particle sets for tree search should not be chosen too large as the computational effort increases quickly. However, using more than one particle was observed to be beneficial and a particle set size around five particles was a good trade-off.

As a last result, it turned out that progressive widening on the observation space impairs the planning algorithm (see Fig. 6.8). Therefore, progressive widening is not employed in the PFT algorithm in this work.

To conclude this section with the objective from the beginning, it is possible to find a good parameter set for this algorithm to be applied in a near real-time setting with a runtime limited to 1s, such that it still finds feasible solutions to the motion planning problem.

6.4. Model Parameter Analysis

The parameter analysis in the previous section was based on theoretical considerations and focused on important solver parameters, that turned out to have great influence on the planned trajectory. In this section, the analysis is dedicated to the automated driving domain as the influence of POMDP model parameters on the search tree and the Q-function estimate is examined.

Based on the findings of the previous section, the solver parameters for the subsequent experiments are chosen according to Tab. 6.5. The number of particles m and the exploration constant c are slightly reduced in order to obtain trajectories for longer horizons and, as noted before, no observation widening is applied. Finally, the runtime is limited to 1s.

Parameter	Value
m	3
c	4000
k_{obs}	0.5
T_{run}	$1s$

Table 6.5.: Solver parameters for POMDP model parameter evaluation.

IDM Reference Velocity of Other Vehicle $v_{ref,k}$

The model parameter analysis focuses on parameters of the intelligent driver model, which is used in the transition model of the POMDP to model the behavior of other vehicles. As noted in Section 5.2.2, the reference velocity $v_{ref,k}$ has a major impact on other vehicles' behavior. Therefore, this parameter will be subject of evaluation in this section.

The evaluation (Fig. 6.9) is conducted analog to the previous section, where three plots were analyzed for every parameter combination:

The center and the right plots are equivalent to the previous section (Sec. 6.3). The left plot shows the predicted particles on the planned trajectory after a time horizon of 3s - that time, where the collision occurred in the evaluation scenario. Analog to the previous section, the driving scenario is taken from `DR_DEU_Roundabout_OF` at timestamp `146000` as given in Fig. 6.4(a). Additionally, the ground truth position of the ego vehicle and the other vehicle after 3s is depicted for comparison to the prediction of the planner in high transparency.

Using the parameter set from Tab. 6.5, the reference velocity for the ego vehicle is set to $v_{ref,ego} = 6\frac{m}{s}$, such that it is encouraged to drive at constant speed. Then, the desired velocity $v_{ref,k}$ is varied and the values are chosen from the set $\{3.5, 4.0, 7.0, 10.0\}\frac{m}{s}$, where $v_{ref,k} = 7.0\frac{m}{s}$ was used for the experiments in the previous section (Sec. 6.3). The true parameter of vehicle ID60 lies in between 4 and $5\frac{m}{s}$ as it drives with speeds in this range between $t = 146.0s$ and $t = 149.0$ in the data set.

Parameter	Value
$v_{ref,ego}$	$6.0\frac{m}{s}$
$v_{ref,k}$	$\{3.5, 4.0, 7.0, 10.0\}\frac{m}{s}$

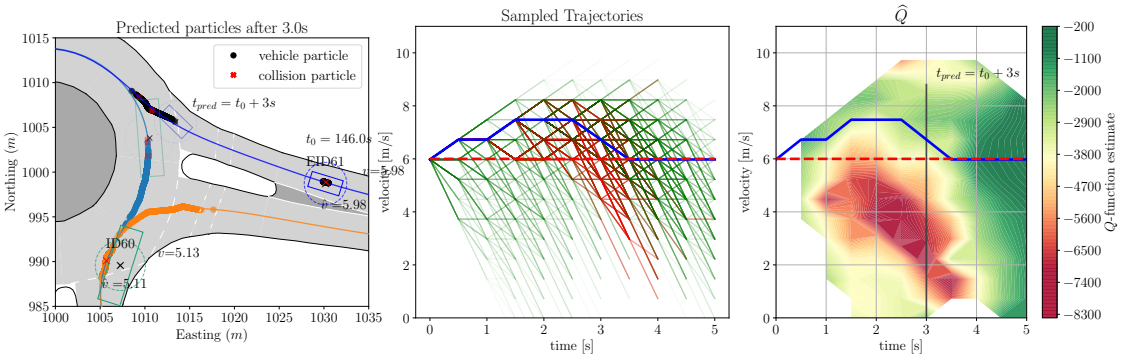
Table 6.6.: Parameters for POMDP model evaluation.

Results

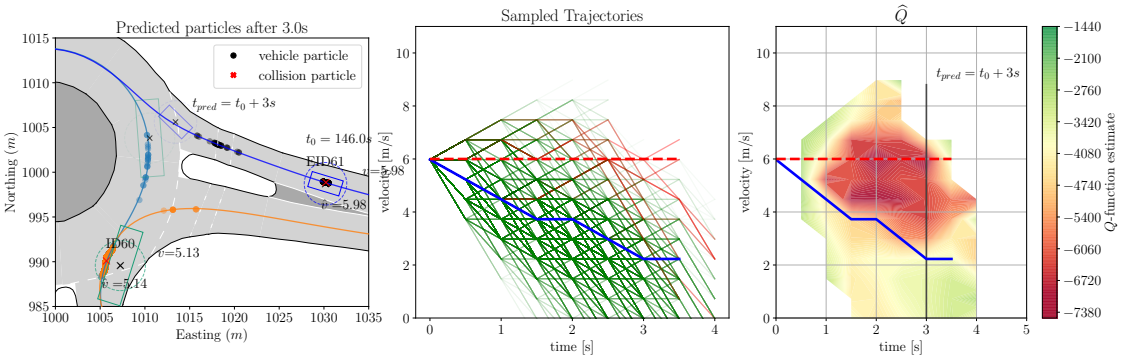
The results are depicted in Fig. 6.9. The Q-function approximations in the right plots demonstrate the soundness of the POMDP planner. If the planner assumes that the other vehicle will slow down ($v_{ref,k}$ low, see Figs. 6.9(a) and 6.9(b)), the region of high collision probability is at lower speeds in the time-velocity diagram of $\widehat{Q}$ (red regions in the right plots). Conversely, if the ego vehicle expects the other vehicle to speed up the region of high collision probability is at higher ego vehicle velocities, as vehicle ID60 might already have passed the entry to the roundabout when the ego vehicles arrives there (see Figs. 6.9(c) and 6.9(d)). In all cases, the planned trajectory (blue line in the center and right plots) is evidently influenced by the predicted collision region.

Moreover, the predicted particle positions of the ego vehicle (black particles) and the other vehicle (colored particles) are consistent with the planned velocity profile and the parameter $v_{ref,k}$, respectively (see left plots in Fig. 6.9).

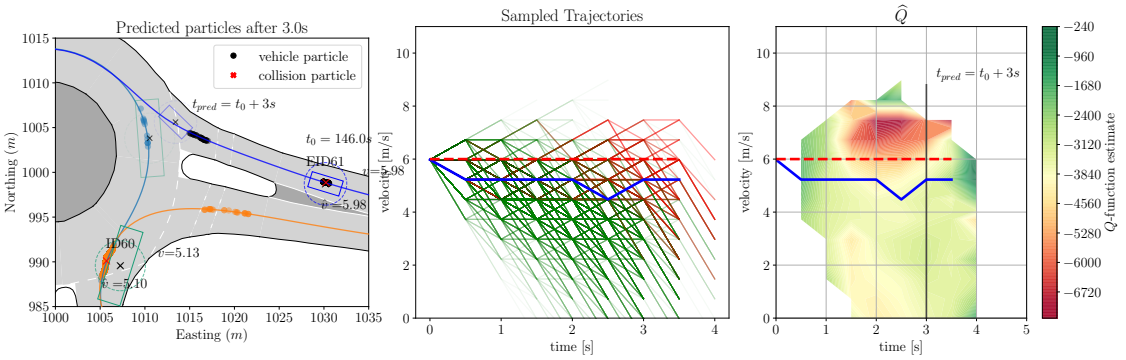
An interesting observation can be made in transition from $v_{ref,k} = 3.5\frac{m}{s}$ to $v_{ref,k} = 4.0\frac{m}{s}$, as for these to values the planner finds trajectories from different homotopy classes. In the first case (depicted in Fig. 6.9(a)), the planner expects vehicle ID60 slowing down strongly and plans a (rather aggressive) trajectory for entering the roundabout in front of the other vehicle. In the latter case (see Fig. 6.9(b)), entering in front of the other vehicle has a too high collision probability and hence the planner decides to break and yields to the other vehicle.



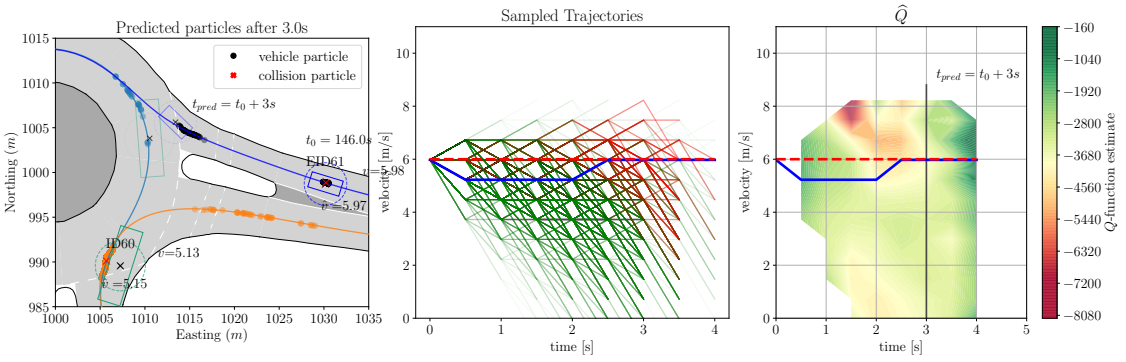
(a) Results for IDM reference velocity $v_{ref,k} = 3.5\frac{m}{s}$.



(b) Results for IDM reference velocity $v_{ref,k} = 4.0\frac{m}{s}$.



(c) Results for IDM reference velocity $v_{ref,k} = 7.0\frac{m}{s}$.



(d) Results for IDM reference velocity $v_{ref,k} = 10.0\frac{m}{s}$.

Figure 6.9.: Planning results for the evaluation scenario with $m = 3$ tree search particles, a UCB constant $c = 4000$, and a runtime $T_{run} = 1000$ms.

All in all, it can be concluded that the desired velocity of other vehicles has a great influence on the Q-function estimate and hence the planned trajectory for the ego vehicle. This is reasonable, since in general the behavior of the ego vehicle strongly depends on the behavior of surrounding vehicles. Hence, setting this parameter is a strong assumption about future behavior of other vehicles, which might be wrong in many cases. Therefore, the results of these experiments support the idea from Section 5.2.2, to estimate $v_{ref,k}$ for each vehicle V_k from observations and use this estimate for tree search. However, implementing this idea is left for future work.

Further Experiments

Several other experiments have been conducted with the presented POMDP planner and some more experiments might be interesting. However, due to time and space limitations, these experiments and results could not be included in this thesis. Therefore, only the main ideas and findings are sketched here.

Besides varying the reference velocity of the other vehicles $v_{ref,k}$, the reference velocity of the ego vehicle, which is used in the reward model (see Sec. 5.2.4), could be changed as well. In this scenario (Fig. 6.4) it was observed, that even if the vehicle is encouraged to speed up (for high reference velocities), it slows down to avoid the likely collisions.

Instead of varying the reference velocities, one could also increase the level of uncertainty by changing the noise parameters in the transition or observation model accordingly. In case of an increased level of self localization noise for the ego vehicle, the trajectories generated by the planner have been shown to be more conservative, i.e. the ego vehicle planned stronger braking maneuvers to reduce the collision probability. Another interesting experiment would be to vary also the noise of the model for the other vehicles. It is expected, that this also leads to more conservative trajectories, which has already been demonstrated by Hubmann et al. [HSB+18].

So far, in this work only single-timestep scenarios have been used for evaluation of the planner (in contrast to the particle filter). Therefore, in future works the planner should be evaluated in online open- or closed-loop multi-timestep scenarios, where the other vehicles either behave according to recorded data or are steered by other models or planners themselves. Then, the POMDP planner could be investigated for model mismatch between the other drivers and the models used in the transition model. Moreover, the generated behavior could be directly compared with human behavior from the data set for example.

Finally, to evaluate the full interactive planning potential of the planner, more complex scenarios with more vehicles must be used. Those experiments would reveal whether the planner is still able to provide safe motion plans in real-time when the predictions of many other road users need to be considered.

7. Conclusion and Future Work

After briefly revisiting the motivation and objectives of this work, the contributions and results are outlined in this chapter. In the end, directions for future research are set out. These include ideas for improving the POMDP solver as well as the POMDP model with the overall objective to enhance behavior generation.

7.1. Conclusion

The motivation for this thesis was to consider the uncertainties of the environment in the domain of automated driving during motion planning, but at the same time avoid the generation of overly conservative or defensive driving behavior. Therefore, the framework of Partially Observable Markov Decision Processes (POMDPs) was used to model this kind of decision problems, as it allows to integrate uncertainties in models, the environment or future behavior of other agents.

For this reason, the goals were to formulate and implement a POMDP model for the motion planning problem in urban scenarios and to develop an algorithm which is able to find a policy for this driving POMDP online in near real-time.

The main contribution of this work is a POMDP motion planner for automated vehicles, that enhances state of the art POMDP planners by combining weighted particle filters for state and driver intent estimation with Monte-Carlo tree search for decision-making. Additionally, a C++ and Python software framework for the Information Particle Filter Tree algorithm was developed and used for solving the motion planning problem.

The presented POMDP model includes all vehicles (including the ego-vehicle) in a traffic scene in the state space and models the route intention of the other vehicles as hidden, unobservable variable. The action space is discretized in a few different equally spaced acceleration values, which means that the POMDP planner is designed to generate an optimal acceleration profile along a predefined path for the ego-vehicle. The reward model encourages the ego vehicle to select accelerations, such that it drives at a desired speed and avoids collisions with surrounding road users.

In the transition model, the behavior of the other vehicles is modelled by the Intelligent Driver Model (IDM) with additional random noise to accommodate for different driving styles. The observations are generated in the observation model from single states with added observation noise sampled from Gaussian distributions. Additionally, by providing an explicit observation likelihood model, a weighted particle filter can be used for state

estimation and belief updating. For this, the route variable is estimated based on features that combine map information with true or simulated observations of other vehicles. The so gained knowledge can then be used directly for subsequent planning steps.

The results have shown that in this way the POMDP planner is able to precisely infer the intentions of other drivers online and that including the heading of other vehicles in intention estimation is very advantageous.

In the solver parameter study, the influence of the runtime, the exploration constant and the particle set size used for tree search on planning horizon and solution quality has been investigated and a parameter combination, such that a trajectory is planned in near real-time has been found. It was also observed, that in case of limited runtime, reducing the exploration constant led to longer planning horizons and that the tree search particle set size should be chosen rather small to allow for the generation of more episodes in each planning step. Moreover, it turned out that progressive widening should not be used, when observations are resampled and not reused during tree search.

At last, using these findings, the planner has been examined with respect to changes in the POMDP model. Here, subject of evaluation was the reference or desired velocity in the IDM for the other vehicle. The results showed that this parameter had a strong influence on the trajectory generated by the POMDP planner.

7.2. Future Work

There are several approaches to further improve the POMDP planner presented in this work. They all aim to either reduce runtime or refine the models used for planning with the overall goal to generate better, more human alike behavior for the autonomous vehicle.

To begin with, in Section 6.4 it was observed that the behavior and interaction models of other traffic participants have a strong impact on the ego vehicle's behavior. This points out that future work could focus on building or learning models from real traffic data, that predict human behavior better. Using such improved models would lead to a more efficient use of samples and hence computational resources for finding an approximation to the optimal policy.

Another aspect, which could be further improved is the generation of search trees, because so far, they are built from scratch in every time step. This means that the planner performs multiple potentially redundant simulations without reusing the results, which is very inefficient. Hence, future work should reuse parts from search trees over multiple time steps and refine the solution for the current belief state, respectively.

Finally, the rollout is another starting point for future research. In this work, a simple constant-velocity rollout policy has been used. By introducing more realistic rollout policies the initial guess for the Q-values, and probably also the convergence and runtime of the algorithm would be improved. Another research direction focusing on this aspect is to learn a Q-function approximation, instead of behavior models, directly from data sets or from multiple runs of this algorithm. This learned Q-function approximation could then be used instead of many forward simulations of the rollout policy to provide an initial guess of the Q-value.

A. Appendix

A.1. Evaluation Parameters

Parameter	Value	Unit	Description
Δt	0.5	s	time step of the discrete motion models
		Transition Model	
$v_{ref,k}$	7.0	$\frac{m}{s}$	IDM desired/reference velocity of vehicle V_k
δ	4	–	IDM free acceleration exponent
T	1.5	s	IDM desired time gap
d_{min}	2.0	m	IDM minimum jam distance
a_{max}	0.73	$\frac{m}{s^2}$	IDM maximum acceleration
b	1.67	$\frac{m}{s^2}$	IDM desired deceleration
w_{lane}	4.5	m	approximate lanewidth
σ_{IDM}	1.5	$\frac{m}{s^2}$	IDM noise std
$\sigma_{\mathbf{x},V_0}$	0.1	m	position noise std ego vehicle
σ_{v,V_0}	0.2	$\frac{m}{s}$	position noise std ego vehicle
		Generative Observation Model	
$\sigma_{\mathbf{x},V_k}$	0.5	m	position noise std other vehicle
σ_{v,V_k}	1.0	$\frac{m}{s}$	velocity noise std other vehicle
σ_{θ,V_k}	0.087	rad	yaw-angle noise std other vehicle
		Observation Likelihood Model	
$\sigma_{\mathbf{f}_l,V_0,\text{PF}}$	1.0	m	position noise std ego vehicle
$\sigma_{v,V_0,\text{PF}}$	0.5	$\frac{m}{s}$	velocity noise std ego vehicle
$\sigma_{\mathbf{f}_l,V_k,\text{PF}}$	4.0	m	position noise std other vehicles
$\sigma_{v,V_k,\text{PF}}$	2.0	$\frac{m}{s}$	velocity noise std other vehicles
$\sigma_{d,\text{PF}}$	0.9	m	std of the lateral distance route classifier
$\sigma_{\Delta\theta,\text{PF}}$	0.175	rad	std of the yaw angle deviation route classifier
		Reward Model	
v_{ref,V_0}	6.0	$\frac{m}{s}$	constant reference velocity ego vehicle
C_{v+}	-100	–	velocity cost scaling factor $v_0 > v_{\text{ref}}$
C_{v-}	-150	–	velocity cost scaling factor $v_0 < v_{\text{ref}}$
C_{acc}	-50	–	acceleration cost scaling factor
C_{coll}	-10 000	–	collision cost
$d_{\text{safe,offs}}$	1.5	m	collision circle safety distance offset
$a_{\text{th,emergency}}$	-7.0	$\frac{m}{s^2}$	emergency braking threshold

Table A.1.: Parameters of the driving POMDP model.

Parameter	Value	Unit	Description
d	10	–	maximum search tree depth
T_{run}	1000	ms	search time / runtime per move
m	3	–	particle set size in tree search
m_{PF}	5000	–	number of particles for the particle filter
k_{obs}	0.5	–	progressive widening parameter
α_{obs}	0.05	–	progressive widening parameter
γ	0.95	–	discount factor
c	4000	–	UCB exploration constant

Table A.2.: Solver Parameters of the Particle Filter Tree algorithm.

A.2. Source Code

The source code developed within this work is distributed across three different packages:

- "IPFT Solver package": A pure C++ package containing the implementation of the IPFT algorithm as presented in [FT20].
- "Driving POMDP Planner package": A C++ package containing the POMDP model implementation with python bindings as interface to the simulation environment.
- "Python POMDP planner package": A pure Python package that implements the simulators interface and calls the Python bindings of the "Driving POMDP Planner package". Additionally, this package contains python code for data processing of the search trees and the plot scripts used to generate the plots for this thesis.

At the time of submission of this thesis the source code is not in a stage to be published. Hence, the final names have not been determined. As soon as the code is published on github, a link to the repository will be provided here:

`https://github.com/maximilianmbeck`

List of Figures

List of Tables

Bibliography

[ABTC10] M. Araya, O. Buffet, V. Thomas, and F. Charpillet, "A pomdp extension with belief-dependent rewards," in *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Curran Associates, Inc., 2010, pp. 64–72. [Online]. Available: http://papers.nips.cc/paper/3971-a-pomdp-extension-with-belief-dependent-rewards.pdf

[ACBF02] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[AMGC02] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174–188, 2002.

[Bar17] T. D. Barfoot, *State estimation for robotics*. Cambridge, Massachusetts: Cambridge University Press, 2017.

[Bar19] J. T. Barron, "A general and adaptive robust loss function," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019.

[BDCK20] K. Brown, K. Driggs-Campbell, and M. J. Kochenderfer, "A taxonomy and review of algorithms for modeling and predicting human driver behavior," *arXiv:2006.08832v3 [eess.SY]*, Nov. 2020, last retrieved: 2021-04-10. [Online]. Available: https://arxiv.org/abs/2006.08832

[Bel57] R. Bellman, *Dynamic Programming*. Princeton, New Jersey: Princeton University Press, 1957.

[BPW+12] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[BSBK20] R. P. Bhattacharyya, R. Senanayake, K. Brown, and M. J. Kochenderfer, "Online parameter estimation for human driver behavior prediction," in *2020 American Control Conference (ACC)*, Jul. 2020, pp. 301–306.

[BZS14] P. Bender, J. Ziegler, and C. Stiller, "Lanelets: Efficient map representation for autonomous driving," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, 2014, pp. 420–425.

[CGM07] O. Cappé, S. J. Godsill, and E. Moulines, "An overview of existing methods and recent advances in sequential monte carlo," *Proceedings of the IEEE*, vol. 95, no. 5, pp. 899–924, 2007.

[CHS+11] A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard, "Continuous Upper Confidence Trees," in *LION'11: Proceedings of the 5th International Conference on Learning and Intelligent OptimizatioN*, Italy, Jan. 2011, p. TBA. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00542673

[DC05] R. Douc and O. Cappe, "Comparison of resampling schemes for particle filtering," in *ISPA 2005. Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis, 2005.*, 2005, pp. 64–69.

[DdFG01] A. Doucet, N. de Freitas, and N. Gordon, *An Introduction to Sequential Monte Carlo Methods*. New York: Springer New York, 2001, pp. 3–14. [Online]. Available: https://doi.org/10.1007/978-1-4757-3437-9_1

[Dil20] H. Dilk, "Automatisiertes fahren nach level 3 ab 2021," 2020, accessed: 2020-12-29. [Online]. Available: https://www.mobile.de/magazin/artikel/automatisiertes-fahren-nach-level-3-ab-2020-12374

[DJ09] A. Doucet and A. M. Johansen, "A tutorial on particle filtering and smoothing: Fifteen years later," *Handbook of nonlinear filtering*, vol. 12, no. 656-704, p. 3, 2009.

[Eri05] C. Ericson, *Real-Time Collision Detection*. San Francisco, CA: Elsevier, 2005.

[ESDK16] A. Eck, L.-K. Soh, S. Devlin, and D. Kudenko, "Potential-based reward shaping for finite horizon online pomdp planning," *Autonomous Agents and Multi-Agent Systems*, vol. 30, no. 3, pp. 403–445, 2016.

[Fis19] J. Fischer, "Integrating information measures into pomdp algorithms for continuous spaces," Master's thesis, Institute of Measurement and Control, Karlsruhe Institute of Technology, 2019.

[FT20] J. Fischer and Ö. S. Tas, "Information particle filter tree: An online algorithm for pomdps with belief-based rewards on continuous domains," in *International Conference on Machine Learning*. PMLR, 2020, pp. 3177–3187.

[Gra18] A. Gramacki, *Nonparametric kernel density estimation and its computational aspects*. Cham, Switzerland: Springer, 2018.

[GSS93] N. J. Gordon, D. J. Salmond, and A. F. M. Smith, "Novel approach to nonlinear/non-gaussian bayesian state estimation," *IEE Proceedings F - Radar and Signal Processing*, vol. 140, no. 2, pp. 107–113, 1993.

[Gus10] F. Gustafsson, "Particle filter theory and practice with positioning applications," *IEEE Aerospace and Electronic Systems Magazine*, vol. 25, no. 7, pp. 53–82, 2010.

[Hau97] M. Hauskrecht, "Incremental methods for computing bounds in partially observable markov decision processes," in *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence*. AAAI Press, 1997, p. 734–739.

[Hau00] ——, "Value-function approximations for partially observable markov decision processes," *Journal of Artificial Intelligence Research*, vol. 13, pp. 33–94, Aug. 2000. [Online]. Available: https://doi.org/10.1613/jair.678

[HK08] W. S. L. Hanna Kurniawati, David Hsu, "SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces," in *Proceedings of Robotics: Science and Systems IV*, Zurich, Switzerland, Jun. 2008. [Online]. Available: http://roboticsproceedings.org/rss04/p9.html

[HSB+18] C. Hubmann, J. Schulz, M. Becker, D. Althoff, and C. Stiller, "Automated driving in uncertain environments: Planning with interaction and uncertain maneuver prediction," *IEEE Transactions on Intelligent Vehicles*, vol. 3, no. 1, pp. 5–17, 2018.

[HSX+18] C. Hubmann, J. Schulz, G. Xu, D. Althoff, and C. Stiller, "A belief state planner for interactive merge maneuvers in congested traffic," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 1617–1624.

[JRM17] W. Jakob, J. Rhinelander, and D. Moldovan, "pybind11 – seamless operability between c++11 and python," 2017, https://github.com/pybind/pybind11.

[Koc15] M. J. Kochenderfer, *Decision Making Under Uncertainty - Theory and Application*, ser. Intelligent Robotics and Autonomous Agents series. Cambridge, Massachusetts: MIT Press, 2015.

[KS06] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Proceedings of the 17th European Conference on Machine Learning*, ser. ECML'06. Berlin, Heidelberg: Springer-Verlag, 2006, p. 282–293. [Online]. Available: https://doi.org/10.1007/11871842_29

[KTH10] A. Kesting, M. Treiber, and D. Helbing, "Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 368, no. 1928, pp. 4585–4605, 2010.

[KZ86] K. Kant and S. W. Zucker, "Toward efficient trajectory planning: The path-velocity decomposition," *The international journal of robotics research*, vol. 5, no. 3, pp. 72–89, 1986.

[LCK95] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling, "Learning policies for partially observable environments: Scaling up," in *Machine Learning Proceedings 1995*, A. Prieditis and S. Russell, Eds. San Francisco (CA): Morgan Kaufmann, 1995, pp. 362 – 370. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9781558603776500529

[Lov91] W. S. Lovejoy, "Computationally feasible bounds for partially observed markov decision processes," *Operations Research*, vol. 39, p. 162–175, Jan. 1991. [Online]. Available: https://www.jstor.org/stable/171496

[OcT20] Ömer Şahin Taş, "P3iv: Probabilistic prediction and planning simulator for intelligent vehicles," 2020, unpublished software.

[PGT03] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for pomdps," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, ser. IJCAI'03. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, p. 1025–1030.

[PGT06] ——, "Anytime point-based approximations for large pomdps," *Journal of Artificial Intelligence Research*, vol. 27, p. 335–380, Jan. 2006. [Online]. Available: http://dx.doi.org/10.1613/jair.2078

[PPJ+18] F. Poggenhans, J.-H. Pauls, J. Janosovits, S. Orf, M. Naumann, F. Kuhnt, and M. Mayr, "Lanelet2: A high-definition map framework for the future of automated driving," in *Proc. IEEE Intell. Trans. Syst. Conf.*, Hawaii, USA, Nov. 2018. [Online]. Available: http://www.mrt.kit.edu/z/publ/download/2018/Poggenhans2018Lanelet2.pdf

[PT87] C. H. Papadimitriou and J. N. Tsitsiklis, "The complexity of markov decision processes," *Mathematics of operations research*, vol. 12, no. 3, pp. 441–450, 1987.

[RN10] S. Russel and P. Norvig, *Artificial Intelligence - A modern Approach.* Upper Saddle River, New Jersey: Prentice Hall, 2010, ch. IV Uncertain knowledge and reasoning, pp. 480–684.

[Ros17] P. E. Ross, "The audi a8: the world's first production car to achieve level 3 autonomy," 2017, accessed: 2020-12-29. [Online]. Available: https://spectrum.ieee.org/cars-that-think/transportation/self-driving/the-audi-a8-the-worlds-first-production-car-to-achieve-level-3-autonomy

[RPPCd08] S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa, "Online planning algorithms for pomdps," *Journal of Artificial Intelligence Research*, vol. 32, p. 663–704, Jul. 2008. [Online]. Available: http://dx.doi.org/10.1613/jair.2567

[SAE14] SAE International, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," 2014.

[Sär13] S. Särkkä, *Bayesian filtering and smoothing.* Cambridge, Massachusetts: Cambridge University Press, 2013, vol. 3.

[SHLB18] J. Schulz, C. Hubmann, J. Löchner, and D. Burschka, "Interaction-aware probabilistic behavior prediction in urban environments," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 3999–4006.

[SK18] Z. Sunberg and M. Kochenderfer, "Online algorithms for pomdps with continuous state, action, and observation spaces," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 28, no. 1, 2018.

[SK20] ——, "Improving automated driving through planning with human internal states," *arXiv:2005.14549v1 [cs.AI]*, 2020, last retrieved: 2021-04-10. [Online]. Available: https://arxiv.org/abs/2005.14549

[SPK13] G. Shani, J. Pineau, and R. Kaplow, "A survey of point-based pomdp solvers," *Autonomous Agents and Multi-Agent Systems*, vol. 27, no. 1, p. 1–51, Jul. 2013. [Online]. Available: https://doi.org/10.1007/s10458-012-9200-2

[SS73] R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable markov processes over a finite horizon," *Operations Research*, vol. 21, no. 5, pp. 1071–1088, 1973. [Online]. Available: http://www.jstor.org/stable/168926

[SS04] T. Smith and R. Simmons, "Heuristic search value iteration for pomdps," in *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, ser. UAI '04. Arlington, Virginia, USA: AUAI Press, 2004, p. 520–527.

[SS05] ——, "Point-based pomdp algorithms: Improved analysis and implementation," in *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, ser. UAI'05. Arlington, Virginia, USA: AUAI Press, 2005, p. 542–549.

[SV10] D. Silver and J. Veness, "Monte-carlo planning in large pomdps," in *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Curran Associates, Inc., 2010, pp. 2164–2172. [Online]. Available: http://papers.nips.cc/paper/4031-monte-carlo-planning-in-large-pomdps.pdf

[SYHL13] A. Somani, N. Ye, D. Hsu, and W. S. Lee, "Despot: Online pomdp planning with regularization," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 1772–1780. [Online]. Available: http://papers.nips.cc/paper/5189-despot-online-pomdp-planning-with-regularization.pdf

[TBF06] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*, ser. Intelligent Robotics and Autonomous Agents series. Cambridge, Massachusetts: MIT Press, 2006.

[THH00] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Phys. Rev. E*, vol. 62, pp. 1805–1824, Aug. 2000. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.62.1805

[TS18] Ö. Ş. Taş and C. Stiller, "Limited visibility and uncertainty aware motion planning for automated driving," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1171–1178.

[YLCT20] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58 443–58 469, 2020.

[ZBDS14] J. Ziegler, P. Bender, T. Dang, and C. Stiller, "Trajectory planning for bertha — a local, continuous method," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, 2014, pp. 450–457.

[ZBS+14] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, T. Dang, U. Franke, N. Appenrodt, C. G. Keller, E. Kaus, R. G. Herrtwich, C. Rabe, D. Pfeiffer, F. Lindner, F. Stein, F. Erbs, M. Enzweiler, C. Knöppel, J. Hipp, M. Haueis, M. Trepte, C. Brenk, A. Tamke, M. Ghanaat, M. Braun, A. Joos, H. Fritz, H. Mock, M. Hein, and E. Zeeb, "Making bertha drive—an autonomous journey on a historic route," *IEEE Intelligent Transportation Systems Magazine*, vol. 6, no. 2, pp. 8–20, 2014.

[ZSW+19] W. Zhan, L. Sun, D. Wang, H. Shi, A. Clausse, M. Naumann, J. Kümmerle, H. Königshof, C. Stiller, A. de La Fortelle, and M. Tomizuka, "INTERACTION Dataset: An INTERnational, Adversarial and Cooperative moTION Dataset in Interactive Driving Scenarios with Semantic Maps," *arXiv:1910.03088 [cs, eess]*, 2019.