SAFETY AND EFFICIENCY IN AUTONOMOUS VEHICLES THROUGH PLANNING WITH UNCERTAINTY

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Zachary Nolan Sunberg

June 2018

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Mykel J. Kochenderfer) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Marco Pavone)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Mac Schwager)

Approved for the Stanford University Committee on Graduate Studies

Abstract

Effective autonomous air and ground vehicles will need to maintain safety while accomplishing tasks efficiently in terms of time and other resources. Unfortunately, the objectives of safety and efficiency are fundamentally opposed because safety constraints prohibit some efficient actions. Moreover, the presence of uncertainty about the environment makes planning safe and efficient actions more difficult.

The Markov decision process (MDP) is a systematic framework for modelling sequential decision problems with outcome uncertainty, and the partially observable Markov decision process (POMDP) adds the additional ability to model state uncertainty. MDPs and POMDPs are suitable models for a wide range of situations that an autonomous vehicle might face. However, obtaining the exact solution to a general POMDP is an intractable problem. This thesis considers approximate MDP and POMDP solutions and seeks to quantify their utility for autonomous vehicles. Specifically, it contains three contributions.

The first chapter analyzes the use of a certifiable safety constraint alongside approximate optimization in the context of unmanned aerial vehicle (UAV) collision avoidance. UAV collision avoidance is challenging in particular because small unmanned vehicles often do not have the performance capability or legal permission to perform conventional altitude-based conflict resolution maneuvers, so they must perform more complex horizontal conflict resolution. In order to ensure safety, aerospace systems have particularly stringent certification requirements that likely preclude approximate randomized planning techniques capable of handling uncertainty. This work evaluates the performance price that comes with using a simple certified policy and shows that MDP and POMDP optimization can significantly reduce that price and improve both safety and efficiency simultaneously.

The second chapter considers the effects of modeling uncertainty in a difficult lane changing task for a self-driving car. Specifically, the research estimates the value of planning with the internal states of other human drivers such as their intentions and dispositions. While several other researchers have used internal-state-aware planning methods to interact with human drivers in desirable ways, they have not evaluated whether these methods offer a substantial quantitative improvement in performance over conventional approaches. This thesis shows that, in a simplified simulated setting, planning with internal states using a POMDP formulation can significantly improve both safety and efficiency simultaneously. Moreover, the thesis describes an experimental method for investigating other cases in which internal-state-aware planning may improve performance.

The benefits of POMDP planning can only be realized with algorithms that can handle real-world domains that are continuous and irregular. To that end, the third contribution of the thesis is a pair of new algorithms for solving POMDPs with continuous state, action, and observation spaces. These algorithms are motivated by analysis and numerical experiments that show that leading online POMDP solvers cannot handle continuous observation spaces. We prove that one of the previous solvers exhibits suboptimal behavior, and explain that the failure is due to two problems. First, the large observation space causes policy trees to become too wide and not deep enough. Second, the number of state particles used to represent beliefs collapses to one, causing overconfidence. The new algorithms, POMCPOW and PFT-DPW, handle these problems using progressive widening and weighted particle belief representations. Numerical experiments show that they are able to solve problems where previous methods fail.

A great deal of future work remains, including further mathematical analysis of the algorithms and testing with more realistic models of human behavior. But the contributions of this thesis to understanding the affects of uncertainty modeling and developing algorithms that apply to realistic problems are two vital steps on the path to safe and efficient autonomy.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Autonomous Vehicles

1.1.1 A World with Autonomous Transportation

- Better Safety
- More Efficiency
- Less wasted time and stress
- Better access to transportation

1.1.2 Current Progress

(Not sure what to say here)

1.1.3 Remaining Challenges

Technical

Legal and Ethical

Business

1.2 Decision Making Under Uncertainty

1.2.1 Uncertainty in Decision Making

Outcome Uncertainty

Model Uncertainty

State Uncertainty

1.2.2 Markov Decision Processes

The Markov decision process (MDP) is a mathematical formalism that can represent a wide range of sequential decision making problems. In an MDP, an agent takes *actions* that affect the *state* of the system and collects *rewards* based on the state and actions. The *Markov property* is a key attribute of MDPs which states that the next state depends (possibly stochastically) on the current state and action, and not on any previous states or actions.

Formally, an MDP is defined by the 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$. The state space, $\mathcal{S}$, is the set of all possible states. The action space, $\mathcal{A}$, is the set of all actions available to the agent. The transition model, $\mathcal{T}$, represents likelihood of transitions, where $\mathcal{T}(s' \mid s, a)$ denotes the probability that the system will transition to state s' given that action a is taken in state s. The reward function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ represents the rewards received while interacting in the environment, where $\mathcal{R}(s, a, s')$ denotes the reward for transitioning from s to s' when action a is taken. Finally, γ governs how reward is discounted in the future.

POMDP methods also require a reward function. In some cases, constructing the reward function that

The objective in an MDP is to find a policy, $\pi : \mathcal{S} \to \mathcal{A}$, that maps each encountered state to an action and, when $a_t = \pi(s_t)$, maximizes the cumulative expected reward,

$$E\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t, s_{t+1})\right] \tag{1.1}$$

where the subscript t is the time index. The state action value function, $Q(s, a)$, is defined as the expectation of the future cumulative reward given that the agent starts in state s, immediately takes action a, and then follows the optimal policy.

1.2.3 Partially Observable Markov Decision Processes

A partially observable Markov decision process (POMDP) is similar to and MDP except that the agent cannot directly observe the state. Instead, the agent only has access to observations that are generated probabilistically based on the actions and latent true states. A POMDP is defined by the 7-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, \mathcal{Z}, \gamma)$, where $\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, $\mathcal{R}$, and γ have the same meaning as in an MDP. Additionally, $\mathcal{O}$, is the observation space, and $\mathcal{Z}$ is the observation model. $\mathcal{Z}(o \mid s, a, s')$ is the probability or probability density of receiving observation o in state s' given that the previous state and action were s and a.

Information about the state may be inferred from the entire history of previous actions and observations and the initial information, b_0. Thus, in a POMDP, the agent's policy is a function mapping each possible history, $h_t = (b_0, a_0, o_1, a_1, o_2, \ldots, a_{t-1}, o_t)$ to an action. In some cases, each state's probability can be calculated based on the history. This distribution is known as a *belief*, with $b_t(s)$ denoting the probability of state s.

The belief is a sufficient statistic for optimal decision making. That is, there exists a policy, π^* such that, when $a_t = \pi^*(b_t)$, the expected cumulative reward or "value function" is maximized for the POMDP [30, 34]. Given the POMDP model, each subsequent belief can be calculated using Bayes' rule according to

$$b'(s') = \frac{\int_{s \in \mathcal{S}} \mathcal{Z}(o \mid s, a, s') \mathcal{T}(s' \mid s, a) b(s) ds}{\int_{s' \in \mathcal{S}} \int_{s \in \mathcal{S}} \mathcal{Z}(o \mid s, a, s') \mathcal{T}(s' \mid s, a) b(s) ds ds'}. \tag{1.2}$$

When the state space is discrete, the integrals may be replaced with sums.

Talk about PSPACE completeness

Generative Models

or many problems, it can be difficult to explicitly determine or represent the probability distributions $\mathcal{T}$ or $\mathcal{Z}$. Some solution approaches, however, only require samples from the state transitions and observations. A generative model, G, stochastically generates a new state, reward, and observation in the partially observable case, given the current state and action, that is $s', r = G(s, a)$ for an MDP, or $s', o, r = G(s, a)$ for a POMDP. A generative model implicitly defines $\mathcal{T}$ and $\mathcal{Z}$, even when they cannot be explicitly represented.

Belief MDP

Every POMDP is equivalent to an MDP where the state space of the MDP is the space of possible beliefs. The reward function of this "belief MDP" is the expectation of the state-action reward function with respect to the belief. The Bayesian update of the belief serves as a generative model for the belief space MDP.

1.2.4 Value Iteration

1.2.5 Monte Carlo Tree Search

MCTS is an effective and widely studied algorithms for online decision-making [14]. It works by incrementally creating a policy tree consisting of alternating layers of state nodes and action nodes and estimating the state-action value function, $Q(s, a)$, at each of the action nodes. Only a generative model, G, is required by the algorithm. The tree is constructed by running n Monte Carlo simulations with four phases, although there are many variations of this algorithm.

1. *Search.* In the initial phase of the simulation, the policy defined by the tree is used. At each state node, a selection criterion based on Q is used to choose

a favorable action, and the tree is traversed through the node and to the next state node determined by G.

2. *Expansion.* Eventually, the simulation reaches an action node that does not have any children. At this point, a new state is sampled with G and a corresponding node created along with children corresponding to each action.

3. *Rollout.* After the expansion step, the simulation is continued with a rollout policy, often consisting of randomly selected actions, until the future accumulated reward will be negligible because of the compounding discount factor.

4. *Update.* Once the simulation has terminated, the estimates of $Q(s,a)$ at each of the visited action nodes are updated with the discounted reward received during the simulation after visiting the node.

This approach builds a tree asymmetrically favoring regions of the state and action spaces that will be visited when the optimal policy is executed.

Upper Confidence Trees

The selection criterion used to choose actions in the search phase is very important. It must balance favoring actions with large Q values that are expected to yield good results with exploration of new actions. The most widely used approach for this is known as the upper confidence bound for trees (UCT) algorithm. At each state node, it chooses the action that maximizes the upper confidence bound

$$UCB(s,a) = Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}} \tag{1.3}$$

where $N(s,a)$ is the number of times the action node has been visited, $N(s) = \sum_{a\in\mathcal{A}} N(s,a)$, and c is a problem-specific parameter that governs the amount of exploration in the tree. The second term causes the algorithm to favor actions that have been taken less often.

Double Progressive Widening

In cases where the action and state spaces are large or continuous, the MCTS algorithm will produce trees that are very shallow. In fact, if the action space is continuous, the UCT algorithm will never try the same action twice (observe that, if $N(s, a) = 0$ then $UCB(s, a)$ in (1.3) is infinite, so untried actions are always favored). Moreover, if the state space is continuous and the transition probability density is finite, the probability of sampling the same state twice from G is zero. Because of this, simulations will never pass through the same state node twice and a tree below the first layer of state nodes will never be constructed.

In progressive widening, the number of children of a node is artificially limited to kN^α where N is the number of times the node has been visited and k and α are parameters chosen for the problem [17]. Originally, progressive widening was applied to the action space, and was found to be especially effective when a set of preferred actions was tried first [14]. The term *double* progressive widening refers to progressive widening in both the state and action space. When the number of state nodes is greater than the limit, instead of simulating a new state transition, one of the previously generated states is chosen with probability proportional to the number of times it has been previously generated.

1.2.6 Particle Filtering

Aside from a few special cases, for example when the system dynamics are linear and the transition and observation distributions are Gaussian, the integrals in the Bayesian belief update (1.2) are impossible or difficult to solve analytically. Thus, numerical approaches must be used. A popular technique for this is particle filtering, usually incorporating domain-specific heuristic modifications to prevent problems such as particle depletion [67].

Sequential importance resampling, one of the most common and effective variations, requires a state generative model, G_s that can sample from the transition distribution and an explicit representation of the observation distribution, $\mathcal{Z}(\cdot \mid s, a, s')$, which is often easier to represent than the transition distribution. The belief is approximated with a set of m particles, $\{s_i\}_{i=1}^m$ and associated weights, $\{w_i\}_{i=1}^m$. The probability of each state is approximated by the sum of the weights corresponding to particles with that state value,

$$b(s) \approx \sum_{i=1}^{m} w_i \delta_s(s_i) \tag{1.4}$$

where $\delta_s(\cdot)$ is the Dirac delta function centered at s. A belief update is approximated by sampling m states $\{\tilde{s}_i\}_{i=1}^m$ from the collection of particles with probability proportional to the associated weight, generating a particle, $s'_i = G(\tilde{s}_i, a)$ for each of these states, and finally setting the new weight proportional to the probability of generating the measured observation with this state, $w'_i \propto \mathcal{Z}(o \mid \tilde{s}_i, a, s'_i)$.

1.2.7 Approximate Solutions to POMDPs

Considerable progress has been made in solving large POMDPs. Initially, exact offline solutions to problems with only a few discrete states, actions, and observations were sought by using value iteration and taking advantage of the convexity of the value function [30], although solutions to larger problems were also explored using Monte Carlo simulation and interpolation between belief states [66]. Many effective offline planners for discrete problems use point based value iteration, where a selection of points in the belief space are used for value function approximation, [40]. Offline solutions for problems with continuous state and observation spaces have also been proposed [5, 12].

There are also various solution approaches that are applicable to specific classes of POMDPs, including continuous problems. For example, Platt et al. [52] simplify planning in large domains by assuming that the most likely observation will always be received, which can provide an acceptable approximation in some problems with unimodal observation distributions. Morere, Marchant, and Ramos [47] solve a monitoring problem with continuous spaces with a Gaussian process belief update. Hoey and Poupart [27] propose a method for partitioning large observation spaces without information loss, but demonstrate the method only on small state and action spaces that have a modest number of conditional plans. Other methods involve motion-planning techniques [46, 53, 15]. In particular, Agha-Mohammadi, Chakravorty, and Amato [1] present a method to take advantage of the existence of a stabilizing controller in belief space planning. Van Den Berg, Patil, and Alterovitz [70] perform local optimization with respect to uncertainty on a pre-computed path, and Indelman, Carlone, and Dellaert [29] devise a hierarchical approach that handles uncertainty in both the robot's state and the surrounding environment.

General purpose online algorithms for POMDPs have also been proposed. These algorithms are mostly derivatives of the MCTS algorithm for MDPs (Section 1.2.5). A conceptually straightforward way to solve a POMDP using MCTS is to apply it to the corresponding belief MDP. Indeed, many tree search techniques have been applied to POMDP problems in this way [55]. However, when the Bayesian belief update is used, this approach is computationally expensive. POMCP can tackle problems many times larger than its predecessors because it uses state trajectory simulations, rather than full belief trajectories, to build the tree. Determinized sparse partially observable tree (DESPOT) is a similar approach that attempts to achieve better performance by analyzing only a small number of random outcomes in the tree [63]. Adaptive belief tree (ABT) was designed specifically to accommodate changes in the environment without having to replan from scratch [41].

Add POMCP section(?), or at least specifics

1.2.8 QMDP

write QMDP section

1.3 Contributions

Chapter 2

Trusted and Optimized UAV Collision Avoidance

2.1 Collision Avoidance for Unmanned Aerial Vehicles

As unmanned aerial vehicles (UAVs) move toward full autonomy, it is vital that they be capable of effectively responding to anomalous events, such as the intrusion of another aircraft into the vehicle's flight path. Minimizing collision risk for aircraft in general, and UAVs in particular, is challenging for a number of reasons. First, avoiding collision requires planning in a way that accounts for the large degree of uncertainty in the future paths of the aircraft. Second, the planning process must balance the competing goals of ensuring safety and avoiding disruption of normal operations. Many approaches have been proposed to address these challenges [39, 50, 4, 25, 26, 48, 65, 35, 38, 31, 7, 28, 54].

At present, there are two fundamentally different approaches to designing a conflict resolution system.The first approach focuses on inspiring confidence and trust in the system by making it as simple as possible for government regulators and vehicle operators to understand. This is accomplished by constructing the algorithm using hand-specified rules. Several algorithms that fit this paradigm have been proposed,

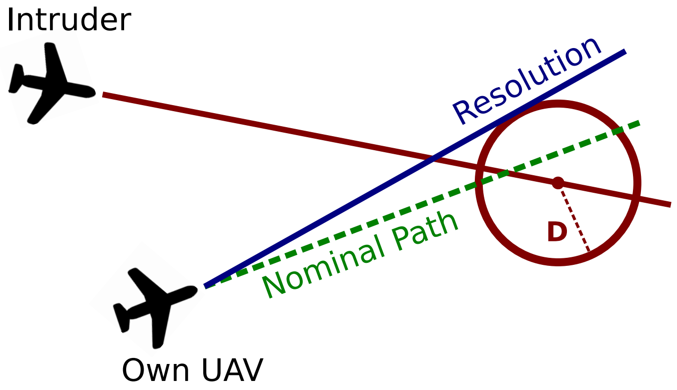


Figure 2.1: Trusted resolution logic. The near mid air collision exclusion zone (circle with radius D) moves with the intruder through time, but is shown here only at the time of closest approach. The TRL (Algorithm 1) finds a straight trajectory close to the nominal path that avoids this zone.

and research toward formally verifying their safety-critical properties is underway [4, 25, 26, 48]. In this chapter, I will refer to such algorithms as *trusted resolution logic* (TRL). Many TRL systems have a number of parameters that determine how conservatively the system behaves. These parameters may be tuned to meet specific performance goals, but their exact affects on performance can typically only be characterized empirically.

The second approach focuses on optimizing performance. This entails the offline or online computation of a “best” response action. Dynamic programming is widely used for this task [65, 35, 38, 31, 7, 28]. Conflict resolution systems designed using this approach will be referred to as *directly optimized* systems. Unfortunately, even if a conflict resolution system performs well in simulation, government regulators and vehicle operators will often (and sometimes rightly) be wary of trusting its safety due to perceived complexity and unpredictability. Even in the best case, such a system would require expensive and time-consuming development of tools for validation as was the case for the recently developed replacement for the traffic alert and collision avoidance system (TCAS) [28].

This chapter proposes two conflict resolution strategies that combine the strengths of trusted resolution logic and direct optimization approaches. In both strategies, dynamic programming is used to find near-optimal actions for each state. The difference lies in the set of actions available to the optimizer. In the *optimized TRL* approach, the actions are a set of certified TRL parameters. In the *trusted direct optimization* approach, the actions are a state-dependent set of immediate response actions that are deemed safe by certified TRL. Both strategies may achieve better performance than the base TRL with a similar relative ease of certification compared to plain direct optimization.

We test the new approaches in a scenario containing a UAV equipped with a perfect (noiseless) sensor to detect the state of an intruder and a simple reference TRL to resolve conflicts. This TRL, illustrated in Figure 2.1, determines a path that does not pass within a specified separation distance, D, of the intruder given that the intruder maintains its current heading (except in cases where no such path exists or when the TRL's heading resolution is too coarse to find such a path). This reference TRL is described in more detail in Section 2.2.4.

To account for uncertainty in the intruder's flight path, if D takes a constant value, say $\bar{D}$, the value must be very large to ensure safety, but this may cause unnecessary departures from normal operation. To overcome this limitation and ensure safety without being too conservative, I apply implementations of the two strategies outlined above. The optimized TRL approach computes a policy, π_D, that specifies a time varying separation distance, D_t, for each encounter state, while the direct optimization and trusted direct optimization approaches compute policies, π_ϕ and $\pi_{T\phi}$, that specifies a bank angle, ϕ_t for each encounter state. For this reason, symbols referring to the optimized TRL approach generally have subscript D, while those referring to direct optimization trusted direct optimization have a subscript ϕ.

The problem of dynamically selecting actions D_t or ϕ_t can be formulated as a Markov decision process (MDP). Online solution of π_D, π_ϕ, or $\pi_{T\phi}$ using an algorithm such as Monte Carlo Tree Search (MCTS) [16] would be conceptually straightforward, but would require significant computing power onboard the vehicle, and it would be difficult and time-consuming to rigorously certify the implementation of MCTS due to its reliance on pseudo-random number generation. A key contribution of this chapter is to devise an *offline* approach to compute the policies. Offline optimization is difficult because of the size of the state space of the MDP, which is the

Cartesian product of the continuous state spaces of the UAV and the intruder. To overcome this challenge, I devise a value function approximation scheme that uses grid-based features that exploit the structure of the state space. This value function is optimized using simulation-based approximate dynamic programming. The policy is then encoded using an approximate post-decision state value function. Armed with this post-decision value function, the UAV can easily extract the optimal action for the current state online by evaluating each possible action.

2.2 MDP Models

The problem of avoiding an intruder using one of the strategies above is formulated as an MDP, referred to as an *encounter* MDP. An encounter involves two aerial vehicles flying in proximity. The first is the vehicle for which the resolution controller is being designed, which will be referred as the "own UAV" (or simply "UAV"). The second is the intruder vehicle, which may be manned or unmanned and will be referred to as the "intruder". Specifically, the encounter MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, R)$, which consists of

- The state space, S: The state of the encounter, s, consists of (1) the state of the own UAV, $s^{(\mathrm{o})}$, (2) the state of the intruder, $s^{(\mathrm{i})}$, and (3) a boolean variable dev. The variable dev is set to true if the UAV has deviated from its nominal course. Collectively, the state is given by the triple

$$s = \left(s^{(\mathrm{o})}, s^{(\mathrm{i})}, \mathrm{dev}\right) . \tag{2.1}$$

  The state components $s^{(\mathrm{o})}$ and $s^{(\mathrm{i})}$ are specified in Section 2.2.2. To model termination, the state space S includes a "dummy" termination state, denoted with s_{term}.

- The action space, A: The action space differs for the two approaches as described in detail in Section 2.2.5. a will be used to refer to all types of action depending on the context.

- The state transition probability density function, $T : S \times A \times S \to \mathbb{R}$: The value $T(s, D, s')$ is the probability density of transitioning to state s' given that the separation parameter D is used within the TRL at state s. This function is implicitly defined by a generative model that consists of a state transition function $F(\cdot)$ (described in Section 2.2.3) and a stochastic process W (described in Section 2.2.2).
- The reward, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$: The reward function, defined in Section 2.2.6, rewards reaching a goal and penalizes near mid air collisions (NMACs), deviation from the nominal path, and time outside a goal region.

2.2.1 Model Assumptions

Two important simplifying assumptions were made for this initial study. First, the UAV and the intruder move only in the horizontal plane and at constant speed. Modeling horizontal maneuvers is necessary because UAVs will likely have to employ horizontal maneuvers in place of or in addition to the vertical maneuvers that current collision avoidance systems for manned aircraft such as TCAS rely on. This is due to both climb performance limitations and regulatory constraints such as the 400ft ceiling for small (<55lb) commercial UAVs in Federal Aviation Administration rules [62]. Constraining altitude and speed simplifies exposition and reduces the size of the state space, keeping the computational burden modest compared with a formulation that includes variable altitude and speed. Extensions to higher fidelity models (e.g., [37]) are possible and left for future research. A higher fidelity model would present a computational challenge, but perhaps not an insurmountable one. For example, the TRL could be extended to handle variable speed and altitude, but the policy could be optimized only on the most important dimensions of the model (e.g., the horizontal plane). See [36] for a similar successful example.

Second, the intruder dynamics are *independent* of the UAV's state; in other words, the intruder does not react to the flight path of the UAV. The cooperative setting where both the UAV and the intruder are equipped with a collision avoidance system (CAS) [54, 68, 35] is left for future research.

2.2.2 Vehicle States and Dynamics

This paper uses a very simple discrete-time model of an encounter between two aerial vehicles with time steps of duration Δt. Throughout this section, the superscripts $^{(\mathrm{o})}$ and $^{(\mathrm{i})}$ refer to the own UAV and intruder quantities, respectively, while the superscript $^{(\cdot)}$ may be replaced by either $^{(\mathrm{o})}$ or $^{(\mathrm{i})}$. Both the UAV's and the intruder's states consist of the horizontal position (x, y) and heading ψ, that is

$$s^{(\mathrm{o})} = \left(x^{(\mathrm{o})}, y^{(\mathrm{o})}, \psi^{(\mathrm{o})}\right), \quad s^{(\mathrm{i})} = \left(x^{(\mathrm{i})}, y^{(\mathrm{i})}, \psi^{(\mathrm{i})}\right). \tag{2.2}$$

The UAV and intruder also have similar dynamics. Both aircraft fly forward in the horizontal plane at constant speeds denoted by $v^{(\cdot)}$. They may turn at rates $\dot{\psi}^{(\cdot)}$ that remain constant over the simulation step. The following equations define the vehicle dynamics:

$$\begin{aligned}
x_{t+1}^{(\cdot)} &= \begin{cases} x_t^{(\cdot)} + v^{(\cdot)} \cos\left(\psi_t^{(\cdot)}\right) \Delta t & \text{if } \dot{\psi}^{(\cdot)} = 0 \\ x_t^{(\cdot)} + v^{(\cdot)} \frac{\sin\left(\psi_t^{(\cdot)} + w_t \Delta t\right) - \sin\left(\psi_t^{(\cdot)}\right)}{\dot{\psi}^{(\cdot)}} & \text{otherwise} \end{cases} \\
y_{t+1}^{(\cdot)} &= \begin{cases} y_t^{(\cdot)} + v^{(\cdot)} \sin\left(\psi_t^{(\cdot)}\right) \Delta t & \text{if } \dot{\psi}^{(\cdot)} = 0 \\ y_t^{(\cdot)} - v^{(\cdot)} \frac{\cos\left(\psi_t^{(\cdot)} + \dot{\psi}^{(\cdot)} \Delta t\right) - \cos\left(\psi_t^{(\cdot)}\right)}{\dot{\psi}^{(\cdot)}} & \text{otherwise} \end{cases} \\
\psi_{t+1}^{(\cdot)} &= \psi_t^{(\cdot)} + \dot{\psi}^{(\cdot)} \Delta t.
\end{aligned}$$

The intruder makes small random turns with

$$\dot{\psi}^{(\mathrm{i})} = w_t, \tag{2.3}$$

where w_t is a stochastic disturbance. We let W denote the stochastic process $\{w_t : t \in \mathbb{N}\}$. The random variables w_t in W are assumed independent and identically normally distributed with zero mean and a specified standard deviation, $\sigma_{\dot{\psi}}$. Subsequently, the intruder dynamics will be collectively referred to as $f^{(\mathrm{i})}$, and

$$s_{t+1}^{(\mathrm{i})} = f^{(\mathrm{i})}\left(s_t^{(\mathrm{i})}, w_t\right). \tag{2.4}$$

The dynamics of the UAV are simplified conventional fixed wing aircraft dynamics with a single input, namely the roll angle $\phi^{(\mathrm{o})}$. We assume that the roll dynamics are fast compared to the other system dynamics, so that the roll angle $\phi^{(\mathrm{o})}$ may be directly and instantaneously commanded by the control system. This assumption avoids the inclusion of roll dynamics, which would increase the size of the state space.

$$\dot{\psi}_t^{(\mathrm{o})} = \frac{g \tan \phi_t^{(\mathrm{o})}}{v^{(\mathrm{o})}}, \tag{2.5}$$

where g is acceleration due to gravity. The performance of the UAV is limited by a maximum bank angle

$$|\phi_t^{(\mathrm{o})}| \leq \phi_{\max}. \tag{2.6}$$

The UAV dynamics will be collectively referred to as $f^{(\mathrm{o})}$, and

$$s_{t+1}^{(\mathrm{o})} = f^{(\mathrm{o})}\left(s_t^{(\mathrm{o})}, \phi_t^{(\mathrm{o})}\right). \tag{2.7}$$

2.2.3 Transition Function

It will often be convenient to refer to all of the dynamics in the state transition with a single transition function, which will be defined below after defining additional behavior regarding goals and near mid-air collisions.

The goal region that the UAV is trying to reach is denoted by S_{goal}. This is the set of all states in S for which $\left\| \left(x^{(\mathrm{o})}, y^{(\mathrm{o})}\right) - \left(x_{\mathrm{goal}}^{(\mathrm{o})}, y_{\mathrm{goal}}^{(\mathrm{o})}\right) \right\| \leq D_{\mathrm{goal}}$, where $D_{\mathrm{goal}} > 0$ is a specified goal region radius, and $\left(x_{\mathrm{goal}}^{(\mathrm{o})}, y_{\mathrm{goal}}^{(\mathrm{o})}\right)$ is the goal center location. A near mid air collision (NMAC) occurs at time t if the UAV and intruder are within a minimum separation distance, $D_{\mathrm{NMAC}} > 0$, that is if $\left\| \left(x_t^{(\mathrm{o})}, y_t^{(\mathrm{o})}\right) - \left(x_t^{(\mathrm{i})}, y_t^{(\mathrm{i})}\right) \right\| \leq D_{\mathrm{NMAC}}$. If the UAV reaches the goal region at some time t, i.e., $s_t \in S_{\mathrm{goal}}$, or if an NMAC occurs, the overall encounter state s transitions to the terminal state s_{term} and remains there. If the UAV performs a turn, dev is set to true because the vehicle has now deviated from the nominal straight path to the goal.

Let the state transition function, defined by (2.4), (2.7), and the special cases above, be denoted concisely as F so that

$$s_{t+1} = F(s_t, \phi_t^{(\mathrm{o})}, w_t), \tag{2.8}$$

2.2.4 Reference Trusted Resolution Logic

The numerical tests use the simple reference TRL shown in Fig. 2.1. The TRL determines a heading angle, $\psi^{(\mathrm{o})}_{\mathrm{resolution}}$, that is close to the heading to the goal, $\psi^{(\mathrm{o})}_{\mathrm{goal}}$, and that will avoid future conflicts with the intruder given that the intruder maintains its current heading. This is shown in Figure 2.1.

Minimum Separation Distance

Given an initial state s, a candidate heading for the UAV, $\psi^{(\mathrm{o})}_{\mathrm{cand}}$, and that both vehicles maintain their heading, the distance between the vehicles at the time of closest approach is a simple analytical function. Specifically, consider the distance $d\left(s, \psi^{(\mathrm{o})}_{\mathrm{cand}}, \tau\right)$ between the vehicles τ time units in the future, i.e.,

$$d\left(s, \psi^{(\mathrm{o})}_{\mathrm{cand}}, \tau\right) = \sqrt{\Delta x(\tau)^2 + \Delta y(\tau)^2}, \tag{2.9}$$

where

$$\begin{aligned}
\Delta x(\tau) &= x^{(\mathrm{i})} - x^{(\mathrm{o})} + \tau v^{(\mathrm{i})} \cos\left(\psi^{(\mathrm{i})}\right) - \tau v^{(\mathrm{o})} \cos\left(\psi^{(\mathrm{o})}_{\mathrm{cand}}\right), \\
\Delta y(\tau) &= y^{(\mathrm{i})} - y^{(\mathrm{o})} + \tau v^{(\mathrm{i})} \sin\left(\psi^{(\mathrm{i})}\right) - \tau v^{(\mathrm{o})} \sin\left(\psi^{(\mathrm{o})}_{\mathrm{cand}}\right).
\end{aligned}$$

The minimum distance between the two vehicles is analytically found by setting the time derivative of $d\left(s, \psi^{(\mathrm{o})}_{\mathrm{cand}}, \tau\right)$ to zero. Specifically, the time at which the vehicles are closest is given by

$$\tau_{\min}\left(s, \psi^{(\mathrm{o})}_{\mathrm{cand}}\right) = \max\left\{\frac{a+b}{c-2d}, 0\right\}, \tag{2.10}$$

where

$$\begin{aligned}
a &:= -v^{(\mathrm{i})}x^{(\mathrm{i})}\cos(\psi^{(\mathrm{i})}) - v^{(\mathrm{i})}y^{(\mathrm{i})}\sin(\psi^{(\mathrm{i})}) \\
b &:= v^{(\mathrm{o})}x^{(\mathrm{i})}\cos(\psi^{(\mathrm{o})}_{\mathrm{cand}}) + v^{(\mathrm{o})}y^{(\mathrm{i})}\sin(\psi^{(\mathrm{o})}_{\mathrm{cand}}) \\
c &:= v^{(\mathrm{o})\,2} + v^{(\mathrm{i})\,2}\cos^2(\psi^{(\mathrm{i})}) + v^{(\mathrm{i})\,2}\sin^2(\psi^{(\mathrm{i})}) \\
d &:= v^{(\mathrm{o})}v^{(\mathrm{i})}(\cos(\psi^{(\mathrm{i})})\cos(\psi^{(\mathrm{o})}_{\mathrm{cand}}) + \sin(\psi^{(\mathrm{i})})\sin(\psi^{(\mathrm{o})}_{\mathrm{cand}})).
\end{aligned}$$

The minimum separation distance over all future time is then

$$d_{\min}\left(s, \psi^{(\mathrm{o})}_{\mathrm{cand}}\right) := d\left(s, \psi^{(\mathrm{o})}_{\mathrm{cand}}, \tau_{\min}\left(s, \psi^{(\mathrm{o})}_{\mathrm{cand}}\right)\right). \tag{2.11}$$

It will also be occasionally convenient to use $d_{\min}$ with only a state as an argument. In this case, $\psi^{(\mathrm{o})}_{\mathrm{cand}}$ should be understood to be the UAV heading angle corresponding to that state, that is $d_{\min}(s) := d_{\min}(s, \psi^{(\mathrm{o})})$ where $\psi^{(\mathrm{o})}$ is the UAV heading in s.

Discrete Heading Optimization

The TRL begins with a discrete set of potential heading values (each denoted $\psi^{(\mathrm{o})}_{\mathrm{cand}}$) for the UAV. It then determines, for a desired separation distance D, which of those will not result in a collision given that the UAV and intruder maintain their headings. Finally, it selects the value from that set which is closest to $\psi^{(\mathrm{o})}_{\mathrm{goal}}$. The TRL is outlined in Algorithm 1.

2.2.5 Action Spaces and Control Systems

The numerical experiments consider policies generated by solving three different MDPs, each with a different action space. For the sake of conciseness, a will be used to represent any type of action depending on the context.

Direct Optimization

In the direct optimization approach, the actions are simply the bank angles less than $\phi_{\max}$, that is $\mathcal{A}_\phi = \{\phi \in \mathbb{R} : |\phi| \leq \phi_{\max}\}$.

Algorithm 1 Trusted Resolution Logic

```
Input: Encounter state $s$, desired separation distance $D$
Output: Resolution heading angle $\psi^{(\mathrm{o})}_{\mathrm{resolution}}$
  function TRL($s$, $D$)
      $\Psi \leftarrow \left\{\psi^{(\mathrm{o})} + n\pi/N : n \in \{-N, \dots, N\}\right\}$          ▷ range of values for heading
      $D^* = \max_{\psi^{(\mathrm{o})}_{\mathrm{cand}} \in \Psi} d_{\min}\left(s, \psi^{(\mathrm{o})}_{\mathrm{cand}}\right)$
      if $D^* < D$ then                                                  ▷ conflict inescapable
          $\Psi \leftarrow \left\{\psi^{(\mathrm{o})}_{\mathrm{cand}} \in \Psi : d_{\min}\left(s, \psi^{(\mathrm{o})}_{\mathrm{cand}}\right) = D^*\right\}$
          return $\operatorname{argmin}_{\psi^{(\mathrm{o})}_{\mathrm{cand}} \in \Psi} |\psi^{(\mathrm{o})}_{\mathrm{cand}} - \psi^{(\mathrm{o})}_{\mathrm{goal}}|$
      else
          $\Psi \leftarrow \left\{\psi^{(\mathrm{o})}_{\mathrm{cand}} \in \Psi : d_{\min}\left(s, \psi^{(\mathrm{o})}_{\mathrm{cand}}\right) \geq D\right\}$
          return $\operatorname{argmin}_{\psi^{(\mathrm{o})}_{\mathrm{cand}} \in \Psi} |\psi^{(\mathrm{o})}_{\mathrm{cand}} - \psi^{(\mathrm{o})}_{\mathrm{goal}}|$
```

Trusted Direct Optimization

In the trusted direct optimization approach, the set of actions is state dependent. Specifically, it is the set of bank angles that the TRL deems safe, that is $\mathcal{A}_{T\phi}(s) = \{\phi \in \mathcal{A}_\phi : d_{\min}(F(s, \phi, 0)) > D_{\mathrm{NMAC}}\}$.

Optimized TRL

In the optimized TRL approach, the actions are the possible values for the separation distance, that is $\mathcal{A}_D = D \in \mathbb{R}_{\geq 0}$ used in the TRL (see Figure 2.1). A simple low level control system converts the desired heading from the TRL (denoted $\psi^{(\mathrm{o})}_{\mathrm{resolution}}$) and into a bank angle for the vehicle. We write this as

$$\phi^{(\mathrm{o})}_t = c\left(s^{(\mathrm{o})}_t, \psi^{(\mathrm{o})}_{\mathrm{resolution}}\right), \tag{2.12}$$

where $c(\cdot)$ represents the low level controller. When the action is a separation distance, the transition function, F, should be understood to contain the TRL and low level control system, that is

$$F(s_t, D_t, w_t) := F(s_t, c(s_t^{(\mathrm{o})}, \mathrm{TRL}(s_t, D_t), w_t). \tag{2.13}$$

2.2.6 Reward

This section considers optimization of two competing metrics. The first goal is to minimize the risk of a NMAC. This aspect of performance is quantified with the fraction of NMACs prevented. The second goal is to minimize the probability of deviation from the nominal path. This metric was chosen (as opposed to a metric that penalizes the size of the deviation) because, in a commercial setting, any deviation from the normal operating plan might have a large cost in the form of disrupting schedules, preventing a mission from being completed, or requiring manual human monitoring. The MDP reward function is designed to encourage the policy towards a solution that performs well with respect to these goals.

Specifically, the utility associated with an encounter is the sum of the stage-wise rewards throughout the entire encounter

$$\sum_{t=0}^{\infty} R(s_t, a_t). \tag{2.14}$$

In order to meet both goals, the stage-wise reward is

$$\begin{aligned} \mathcal{R}(s_t, a_t) := & - c_{\text{step}} + r_{\text{goal}} \times \text{in\_goal}\left(s_t^{(\mathrm{o})}\right) \\ & - c_{\text{dev}} \times \text{causes\_deviation}(s_t, a_t) \\ & - \lambda \times \text{is\_NMAC}(s_t), \end{aligned} \tag{2.15}$$

for positive constants c_{step}, r_{goal}, c_{dev}, and λ. The first term is a small constant cost accumulated in each step to push the policy to quickly reach the goal. The function in_goal indicates that the UAV is within the goal region, so the second term is a reward for reaching the goal. The third term is a penalty for deviating from the nominal path. The function causes_deviation returns 1 if the action will cause a deviation from the nominal course and 0 otherwise. It will only return 1 if the vehicle has not previously deviated and dev is false, so the penalty may only occur once during an encounter. This behavior makes the inclusion of dev in the state necessary. Constants c_{step}, r_{goal}, and c_{dev} represent relative weightings for the terms that incentivize a policy that reaches the goal quickly and minimizes the probability of deviation. Example values for these constants are given in Section 2.4. The fourth term is the cost for a collision. The weight λ balances the two performance goals. We heuristically expect there to be a value of λ for which the solution to the MDP meets the desired risk ratio if it is attainable. Bisection, or even a simple sweep of values can be used to find a suitable value, and this method has been used previously to analyze the performance of aircraft collision avoidance systems [7, 35].

2.2.7 Problem statement

The problem we consider is to find a feedback control policy $\pi^* : S \to A$, mapping an encounter state s_t into an action a_t, that maximizes the expected reward (2.14) subject to the system dynamics (2.8):

$$\begin{array}{ll} \underset{\pi}{\text{maximize}} & E\left[\displaystyle\sum_{t=0}^{\infty} R(s_t, \pi(s_t))\right] \\ \text{subject to} & s_{t+1} = F(s_t, \pi(s_t), w_t), \end{array} \tag{2.16}$$

for all initial states $s_0 \in S$. To make the solution of problem (2.16) practical, we present an approximate dynamic programming approach that yields a suboptimal policy, $\tilde{\pi}$, in the next section.

2.3 Solution Approach

Since the problem (2.16) has continuous state and action spaces and complex dynamics, it is difficult to solve. In order to make it more tractable, two approximations are used.

First, only a small number of discrete actions, which will be referred to as $\tilde{\mathcal{A}}$, are considered. Specifically, for the optimized TRL, direct optimization, and trusted direct optimization approaches, the following action spaces are used:

$$\tilde{\mathcal{A}}_D = \{D_{\text{NMAC}}, 1.5D_{\text{NMAC}}, 2D_{\text{NMAC}}, 3D_{\text{NMAC}}, 4D_{\text{NMAC}}\} \tag{2.17}$$

$$\tilde{\mathcal{A}}_\phi = \left\{-\phi_{\max}, -\frac{\phi_{\max}}{2}, 0, \frac{\phi_{\max}}{2}, \phi_{\max}\right\} \tag{2.18}$$

$$\tilde{\mathcal{A}}_{T\phi}(s) = \left\{\phi \in \tilde{\mathcal{A}}_\phi : d_{\min}\left(F(s,\phi,0)\right) > D_{\text{NMAC}}\right\}. \tag{2.19}$$

The second approximation is the approximate value iteration algorithm [8] used to optimize the policy. The value function, V, represents the expected value of the future reward given that the encounter is in state s and an optimal policy will be executed in the future. We approximate V with a linear architecture of the form [8]

$$\tilde{V}(s) = \beta(s)^\top \theta, \tag{2.20}$$

where the feature function β returns a vector of N_β feature values, and $\theta \in \mathbb{R}^{N_\beta}$ is a vector of weights [8]. At each step of value iteration, the weight vector θ is fitted to the results of a large number of single-step simulations by solving a linear least-squares problem. After value iteration has converged, $\tilde{V}$ is used to estimate a *post-decision state* value function, $\tilde{V}_q$, which is also approximated using a linear combination of features. The policy is extracted online in real time by selecting the action that results in the post-decision state that has the highest value according to $\tilde{V}_q$. The choice of working with post-decision states will be discussed in Section 2.3.2.

2.3.1 Approximate Value Iteration

The bulk of the computation is carried out offline before vehicle deployment using simulation. Specifically, the first step is to estimate the optimal value function for problem (2.16) using value iteration [8]. On a continuous state space such as the encounter state space used in this work, the Bellman operator used in value iteration cannot be applied for each of the uncountably infinite number of states, so an approximation must be used. In this paper we adopt projected value iteration [8], which uses a finite number of parameters to approximate the value function. Each successive approximation, $\tilde{V}_k$, is the result of the Bellman operation projected onto a linear subspace with respect to the Euclidean norm, that is

$$\tilde{V}_{k+1}(s) = \Pi\mathcal{B}[\tilde{V}_k](s), \tag{2.21}$$

where $\mathcal{B}$ is the Bellman operator, and Π is a projection onto the linear subspace Φ spanned by the N_β basis functions (see [8] for a detailed discussion of this approach).

To perform the approximate value iteration (2.21), we resort to Monte Carlo simulations. Specifically, for each iteration, N_{state} states are uniformly randomly selected. If the states lie within the grids used in the feature function (see Section 2.3.4) the sample is "snapped" to the nearest grid point to prevent approximation errors due to the Gibbs phenomenon [21]. At each sampled state $s^{[n]}$, $n = 1, \dots, N_{\text{state}}$, the stage-wise reward and the expected value of the value function are evaluated for each action a in $\tilde{\mathcal{A}}$. The expectation embedded in the Bellman operator is approximated using N_{EV} *single-step* intruder simulations, each with a randomly generated noise value, w_m, $m = 1, \dots, N_{\text{EV}}$. However, since the own UAV dynamics are deterministic, only one own UAV simulation is needed. The maximum over $\tilde{A}$ is selected and stored as the nth entry of a vector v_{k+1}:

$$v_{k+1}[n] := \max_{a\in\tilde{A}} \left\{ R(s^{[n]}, a) + \frac{1}{N_{\text{EV}}} \sum_{m=1}^{N_{\text{EV}}} \beta(F(s^{[n]}, a, w_m))^\top \theta_k \right\}, \tag{2.22}$$

for $n = 1, \dots, N_{\text{state}}$. Here v_{k+1} provides an approximation to the (unprojected) value function. To project v_{k+1} onto Φ, we compute the weight vector θ_{k+1} by solving the least-squares optimization problem

$$\theta_{k+1} = \underset{\theta\in\mathbb{R}^{N_\beta}}{\text{argmin}} \sum_{n=1}^{N_{\text{state}}} \left(\beta\left(s^{[n]}\right)^\top \theta - v_{k+1}[n]\right)^2. \tag{2.23}$$

Iteration is terminated after a fixed number of steps, N_{VI}, and the resulting weight vector, denoted by θ, is stored for the next processing step (Section 2.3.2).

2.3.2 Post Decision Value Function Extraction

For reasons discussed in Section 2.3.3, our second step is to approximate a value function defined over post-decision states [8]. A post-decision state, denoted by q, is defined as a state in S consisting of the own UAV state and dev *at one time step into the future* and the intruder state *at the current time*, that is

$$q_t = \left(s_{t+1}^{(\mathrm{o})}, s_t^{(\mathrm{i})}, \mathrm{dev}_{t+1}\right). \tag{2.24}$$

Correspondingly, let $g : S \times A \to S$ be the function that maps the current state and action to the post-decision state. In other words, function $g(s_t, a_t)$ returns q_t consisting of

$$\begin{aligned} s_{t+1}^{(\mathrm{o})} &= f^{(\mathrm{o})}\left(s_t^{(\mathrm{o})}, a_t\right) \\ s_t^{(\mathrm{i})} &= s_t^{(\mathrm{i})} \\ \mathrm{dev}_{t+1} &= \max\{\mathrm{dev}, \mathrm{causes\_deviation}(s_t, a_t)\}. \end{aligned} \tag{2.25}$$

The approximate value function over post-decision states, $\tilde{V}_q$, is computed as follows. Let $h : S \times \mathbb{R} \to S$ be a function that returns the next encounter state given the post decision state and intruder heading noise value, that is $h(q_t, w_t)$ returns s_{t+1} consisting of

$$s_{t+1}^{(\mathrm{o})} = s_{t+1}^{(\mathrm{o})} \tag{2.26}$$

$$s_{t+1}^{(\mathrm{i})} = f^{(\mathrm{i})}(s_t^{(\mathrm{i})}, w_t) \tag{2.27}$$

$$\mathrm{dev}_{t+1} = \mathrm{dev}_{t+1}, \tag{2.28}$$

where $\left(s_{t+1}^{(\mathrm{o})}, s_t^{(\mathrm{i})}, \mathrm{dev}_{t+1}\right)$ are the members of q_t.

The value function $\tilde{V}_q$ can then be expressed in terms of $\tilde{V}$ as

$$\tilde{V}_q(q) = \underset{w_t}{E}\left[\tilde{V}\left(h\left(q, w_t\right)\right)\right], \tag{2.29}$$

where w_t denotes, as usual, a random variable with Gaussian normal density. Equation (2.29) implies that $\tilde{V}_q$ can also be approximated according to a linear architecture

$$\tilde{V}_q(q) = \beta(q)^\top \theta^q, \tag{2.30}$$

where $\beta(q)$ is the feature vector for post-decision states $q \in S$, and $\theta^q \in \mathbb{R}^{N_\beta}$ is the corresponding weight vector.

Specifically, N_q post decision states are randomly selected using the same method as described in Section 2.3.1 and are denoted as $q^{[n]}$, $n = 1, \dots, N_q$. For each sampled state $q^{[n]}$, the expectation in (2.29) is approximated using N_{EV} single-step simulations. The results are used to solve a least squares optimization problem

$$\theta^q = \operatorname*{argmin}_{\theta \in \mathbb{R}^{N_\beta}} \sum_{n=1}^{N_q} \left(\beta\left(q^{[n]}\right)^\top \theta - v^q[n] \right)^2, \tag{2.31}$$

where

$$v^q[n] := \frac{1}{N_{\mathrm{EV}}} \sum_{m=1}^{N_{\mathrm{EV}}} \beta\left(h\left(q^{[n]}, w_m\right)\right)^\top \theta, \tag{2.32}$$

where w_m, $m = 1, \dots, N_{\mathrm{EV}}$, is a noise value sampled from the distribution of a random variable in W.

2.3.3 Online Policy Evaluation

The first two steps (explained, respectively, in Sections 2.3.1 and 2.3.2) are performed offline. The last step, namely policy evaluation, is performed online. Specifically a suboptimal control at any state s is computed as

$$\tilde{\pi}(s) = \operatorname*{argmax}_{a \in \tilde{A}} \tilde{V}_q\left(g(s, a)\right). \tag{2.33}$$

Since g (defined in (2.25)) is a *deterministic* function of a state-action pair, this calculation does not contain any computationally costly or difficult-to-certify operations such as estimating an expectation.

Two comments regarding the motivation for using the post-decision value function are in order. First, if the value functions could be exactly calculated, the post-decision state approach would be equivalent to the more common Q-factor-based approach, wherein a control is computed by solving

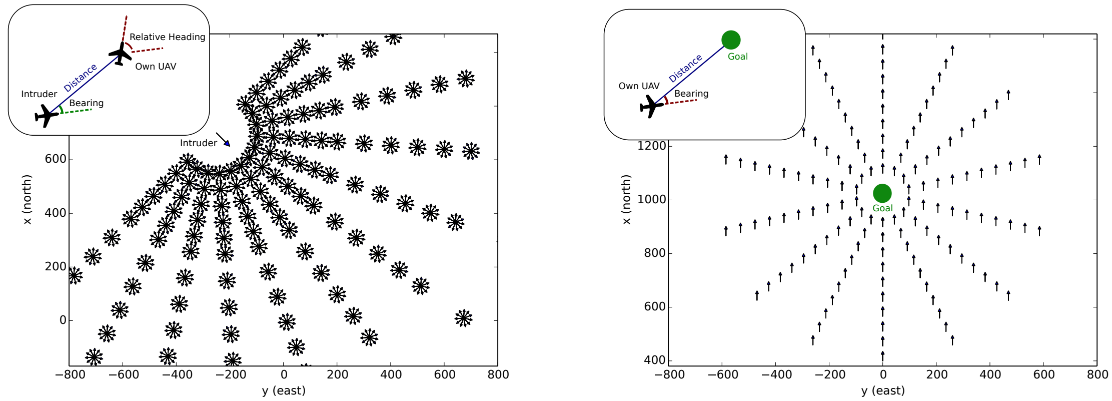
$$\pi(s) = \operatorname*{argmax}_{a \in \tilde{A}} Q(s, a), \tag{2.34}$$

and $Q(s, a)$ (the Q-factor) represents the expected value of taking action a in state s and then following an optimal policy [8]. In fact, one can readily show $Q(s, a) = V_q\left(g(s, a)\right)$. However, when approximations are used, post-decision states provide a more robust (i.e., less susceptible to approximation error) way of deriving control actions [8]. One reason for this is that the cost function is approximated in the space of post-decision states, rather than in the larger space of state-control pairs, and hence the post-decision method is less susceptible to complications with inadequate exploration [8]. Second, the post decision value function is not used for the value iteration portion of the offline solution as it would require full simulations of both the own UAV and the intruder dynamics and, therefore, would be more computationally demanding.

2.3.4 Selection of Features

The primary value function approximation features are the interpolation weights for points in a grid [18]. A grid-based approximation is potentially inefficient compared to a small number of global features (e.g. heading, distance, and trigonometric functions of the variables), however, it is well known that the function approximation used in value iteration, must have suitable convergence properties [9] in addition to approximating the final value function. Indeed, we experimented with a small number of global features, but were unable to achieve convergence and resorted to using a grid. Since a grid defined over the entire six dimensional encounter state space with a reasonable resolution would require far too many points to be computationally feasible, the grid must be focused on important parts of the state space.

Our strategy is to separate features into two groups (along with a constant),



(a) Intruder interpolation grid for β_{intruder}, visualized when the intruder is located at (700 m, −250 m) at heading 135. The top left inset shows the variables used. In the main plot, each of the small arrows represents a grid point. At each of the point locations, there are twelve small arrows radiating out. Each arrow represents a different own UAV heading.

(b) Goal interpolation grid for β_{goal} when the own UAV's heading is directly north. The grid takes advantage of symmetry in the bearing variable, so, for each arrow on the left side, there is an arrow on the right side that corresponds with the *same* point in the grid.

Figure 2.2: Value function approximation interpolation grids

specifically

$$\beta(s) = [\beta_{\text{intruder}}(s^{(\text{o})} - s^{(\text{i})}), \beta_{\text{goal}}(s^{(\text{o})}), 1]. \tag{2.35}$$

The first group, β_{intruder}, captures the features corresponding to a near midair collision and is a function of only the position and orientation of the own vehicle relative to the intruder. The second group, β_{goal}, captures the value of being near the goal and is a function of only the own vehicle state.

Since the domain of β_{intruder} is only three dimensional and the domain of β_{goal} is only two dimensional, relatively fine interpolation grids can be used for value function approximation without requiring a prohibitively large number of features. The β_{intruder} feature group consists of a NMAC indicator function and interpolation weights for a grid (Figure 2.2a) with nodes at regularly spaced points along the following three variables: (1) the distance between the UAV and intruder, (2) the bearing from the intruder to the UAV, and (3) the relative heading between the vehicles. The

β_{goal} vector consists of a goal indicator function, the distance between the UAV and the goal, and interpolation weights for a grid with nodes regularly spaced along the distance between the UAV and the goal and the absolute value of the bearing to the goal from the UAV. The total number of features is $N_\beta = 1813$.

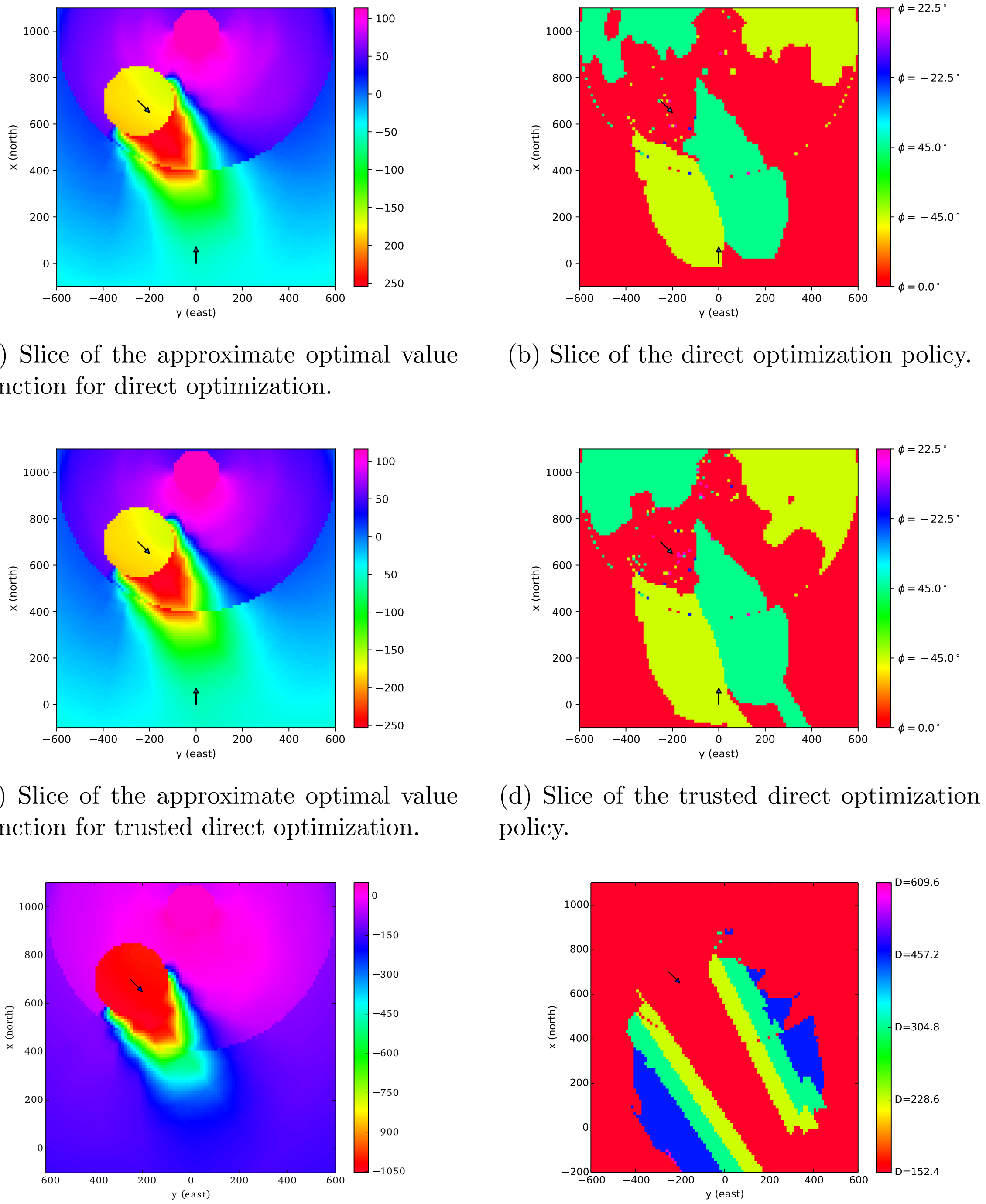
2.4 Results

This section presents results from numerical experiments that illustrate the effectiveness of our approach. The experiments are designed to compare the four approaches discussed in Section 2.1: the "static TRL" approach, "direct optimization", and the new "optimized TRL" and "trusted direct optimization" approaches proposed in this chapter. The static TRL law uses the TRL described in Algorithm 1 with a constant value for the separation distance (denoted with $\bar{D}$), while the other approaches use the approximate optimization procedure described in Section 2.3. The various parameters used in the numerical experiments are listed in Table 2.1.

2.4.1 Policies

This section includes visualizations of two-dimensional "slices" of several UAV policies and their associated value functions. In each of the slices, the intruder is located at (700 m,−250 m) pointed at heading 135 as indicated by the arrow. The goal is at (1000 m,0). Each pixel on this image represents the value function or policy evaluated with the UAV at that position pointed directly north. As expected, for all approaches, there is a low value region in front of and to the south of the intruder and an increase in the value near the goal.

Figures 2.3a and 2.3b show the value function and policy for the directly optimized approach, and Figs. 2.3c and 2.3d show the same for the trusted directly optimized approach. The policies for both approaches are similar, with regions in front of the intruder where sharp turns are commanded. The trusted direct optimization policy is slightly more conservative with larger turn regions because of the restricted action space.



(a) Slice of the approximate optimal value function for direct optimization.

(b) Slice of the direct optimization policy.

(c) Slice of the approximate optimal value function for trusted direct optimization.

(d) Slice of the trusted direct optimization policy.

(e) Slice of the approximate optimal value function for the optimized TRL.

(f) Slice of the optimized TRL policy.

Figure 2.3: UAV Collision avoidance policy visualizations

Figure 2.3f shows the value function and policy for the optimized TRL approach. When multiple actions result in the same post decision state value, the least conservative action is chosen, so the policy yields the lowest value of D (500ft $\approx$ 152.4m) on most of the state space. Because the own UAV is pointed north, the policy is conservative in a region in front of and to the south of the intruder. The band corresponding to small D that stretches across the middle of the conservative region (from (700 m,−250 m) to (−100 m,200 m)) is present because all values of D result in the same post decision value.

2.4.2 Numerical Performance Evaluation

The control polices are evaluated by executing them in a large number of complete (from $t = 0$ to the end state) encounter simulations. The same random numbers used to generate intruder noise were reused across all collision avoidance strategies to ensure fairness of comparisons. In each of the simulations, the own UAV starts pointed north at position $(0, 0)$ in a north-east coordinate system with the goal at $(1000\text{m}, 0)$.

The evaluation simulations use the same intruder random turn rate model with standard deviation $\sigma_{\dot\psi}$ that was used for value iteration. A robustness study using different models is not presented here, but previous research [36] suggests that this method will offer good performance when evaluated against both a range of noise parameters and structurally different models. The intruder initial position is randomly generated between 800 m and 1500 m from the center point of the encounter area at $(500\text{m}, 500\text{m})$ with an initial heading that is within 135° of the direction from the initial position to the center point.

The conservativeness of each control law is characterized by counting the number of deviations from the nominal path in 10,000 simulations with initial conditions shown in Fig. 2.4. Of these simulations, 1009 result in a NMAC if the UAV follows its nominal path, but for most a deviation would not be necessary to avoid the intruder.

The fraction of collisions avoided is estimated using a separate set of 10,000 simulations. Each of these simulations has an initial condition in the same region described

Table 2.1: Parameters for numerical experiments

Description	Symbol	Value
Own UAV speed	$v^{(\mathrm{o})}$	30 m/s
Maximum own UAV bank angle	$\phi_{\max}$	45°
Intruder speed	$v^{(\mathrm{i})}$	60 m/s
Intruder turn rate standard deviation	$\sigma_{\dot{\psi}}$	10°/s
Near mid air collision radius	D_{NMAC}	500 ft
Step cost	c_{step}	1
Reward for reaching goal	r_{goal}	100
Cost for deviation	c_{dev}	100
Step simulations for expectation estimate	N_{EV}	20
Single step simulations per round of value iteration (optimized TRL)	N_{state}	10,000
Single step simulations per round of value iteration (directly optimized)	N_{state}	50,000
Number of value iteration rounds	N_{VI}	35
Single step simulations for post decision value function extraction	N_q	50,000

above, but initial conditions and noise trajectories are chosen by filtering random trials so that each of the simulations *will result in a NMAC if the own UAV follows its nominal path.*

Figure 2.5 shows the Pareto optimal frontiers for the different control laws. Each curve is generated by using various values of λ in the reward function of the MDP, or by using various static values of $\bar{D}$ in the static TRL case. It is clear that the optimization provides better performance than the static TRL. For example, if the desired fraction of NMACs prevented is 96%, interpolation between data points suggests that direct optimization will cause approximately 20% fewer deviations than the static TRL policy. This difference may be interpreted as the *price* if using trusted resolution logic rather than optimization.

Fortunately, both the optimized TRL and trusted direct optimization approaches offer ways to reduce this price. Neither should be much more difficult to certify than the TRL because they are both based on the TRL. In particular, neither will ever command an action that is deemed unsafe by the TRL. Thus, they provide the same

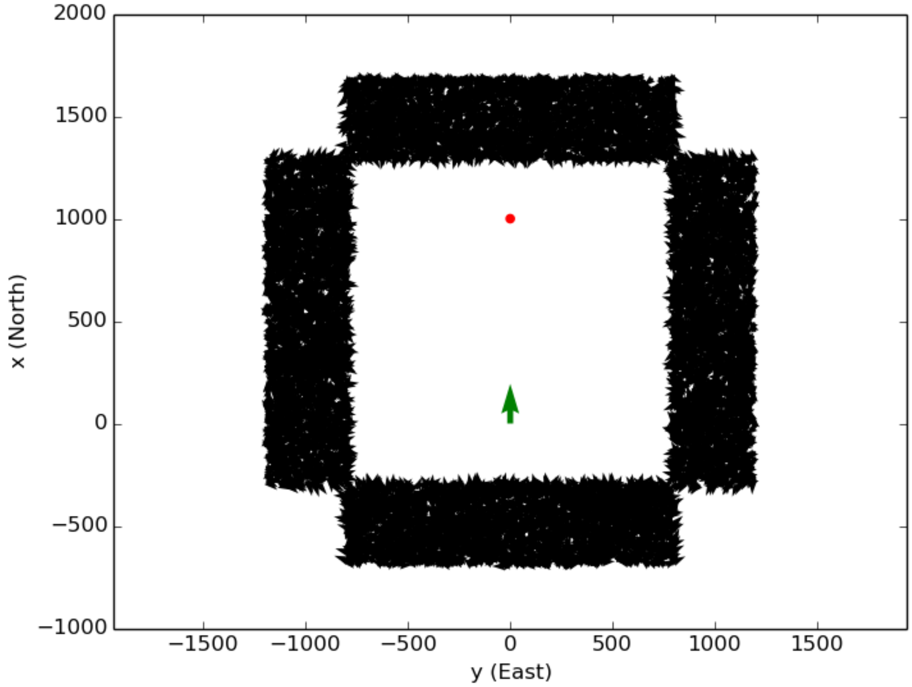


Figure 2.4: Intruder initial conditions for evaluation simulations. Each small black arrow is an intruder initial state. The large green arrow is the UAV's initial state. The red dot is the goal.

level of trust as the TRL and thus reduce the price of trust. In this case, trusted direct optimization is able to nearly close the gap between static TRL and direct optimization, while optimized TRL reduces the gap by about half, so trusted direct optimization would be the preferred approach.

The software used to simulate these experiments was written in the Julia programming language, and is freely available at `https://github.com/zsunberg/UASEncounter`.

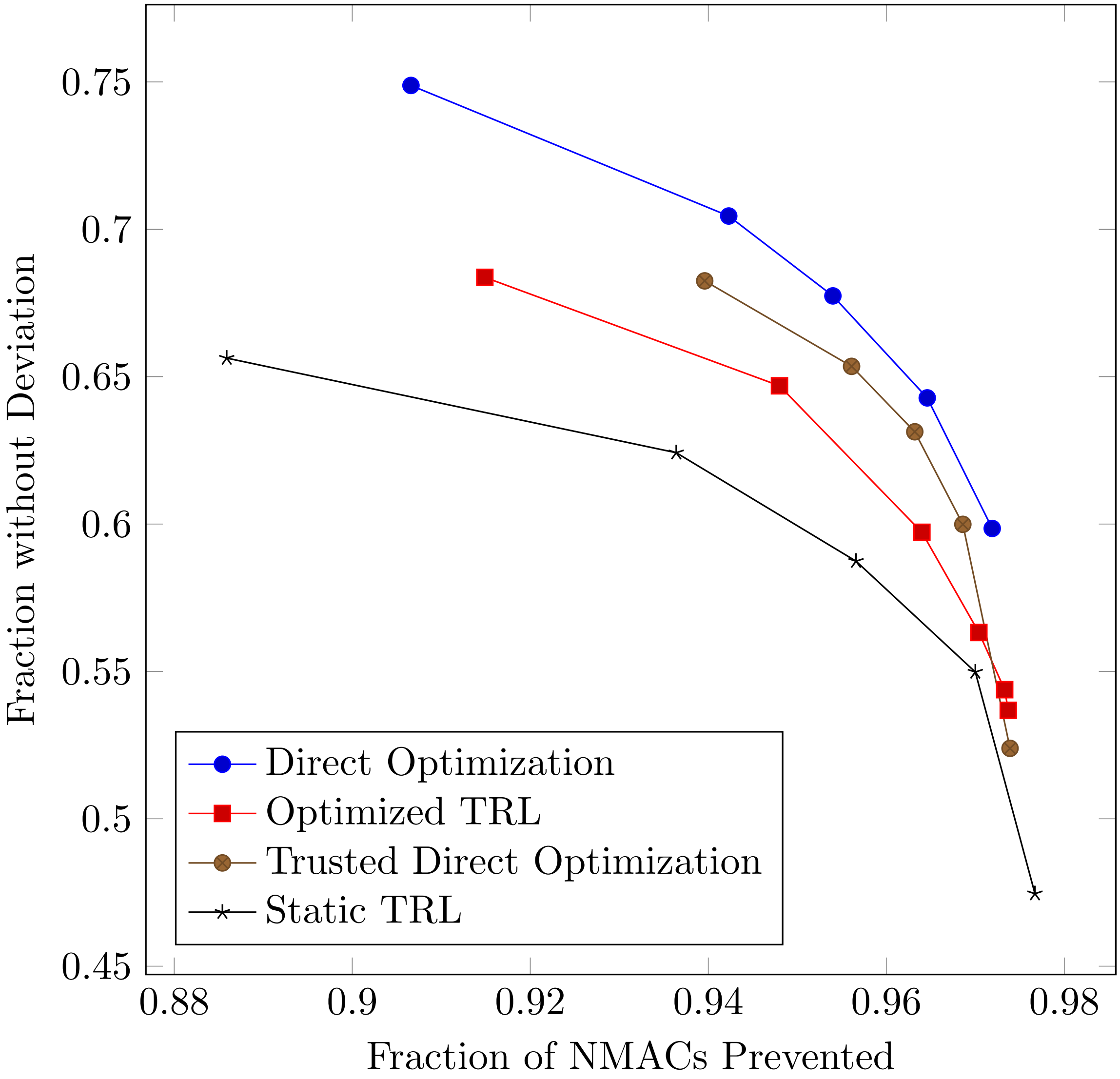


Figure 2.5: Policy performance comparison. The horizontal axis variable is the number of deviations in 10,000 encounter simulations. The values of λ used to generate the datapoints are 100, 316, 1000, 3160, 10^4, and 3.16×10^4 for the optimized TRL policy and 300, 500, 700, 1000, and 1500 for the directly optimized approach. The values of $\bar{D}$ for the static TRL policy are 250 m, 300 m, 350 m, 400 m, and 500 m.

Chapter 3

Planning with Internal States in Freeway Driving

3.1 Human-Robot Interaction in Autonomous Driving

One challenge in introducing autonomous automobiles is ensuring that they interact safely with human drivers. In order to navigate complex driving scenarios, human drivers routinely predict what other drivers will do and make driving decisions based on these predictions. Autonomous vehicles typically take an overly conservative approach, which can result in physical danger, reduced efficiency, and an uncomfortable experience. In a 2015 study, autonomous vehicles drove over 1.2 million miles without being legally responsible for any accidents. However, the autonomous vehicles actually had a higher accident rate than average for a conventional vehicle in the United States because of accidents for which they were not legally responsible [59]. This result suggests that there is significant room for improvement in autonomous-human vehicle interaction.

One approach to improve interaction would be to program ad-hoc logic for each situation into the vehicles. However, this approach is time-consuming and error prone, and edge cases that the programmers have not foreseen can present a safety risk.

Furthermore, this approach limits the performance of the system to the capability of the human programmer. In contrast, artificial intelligence and machine learning techniques have the potential to provide a more robust approach to such decision-making tasks. This chapter explores MDP and POMDP techniques.

POMDPs are particularly well suited for modeling decisions for autonomous vehicles because they explicitly capture the limitations of the vehicle's sensors in measuring the relevant state variables [11, 56, 6]. Though sensors can accurately measure many of the relevant variables pertaining to the physical state of the vehicles, the internal state (e.g., intentions and aggressiveness) of other drivers and road users can only be indirectly inferred [56, 6, 42, 20]. The hypothesis explored in this chapter is that inferring and planning with an estimate of the internal states of the traffic participants will improve safety and efficiency.

Driving strategies derived from MDPs and POMDPs depend on several ingredients to be successful. First, an accurate stochastic model of the environment, including the behavior of other drivers is necessary. Though this chapter uses a very simple model, there has been significant work on creating better models in recent years [23, 71, 58], and the POMDP planning methods used here can easily be adapted to use these new models.

Before investing the effort required to develop and test a POMDP-based decision making system for real autonomous vehicles, it is important to quantify the potential performance improvement. This chapter presents a method that involves comparing solutions obtained from several variations of Monte Carlo Tree Search [13]. For this research, I chose to investigate these ideas in the context of making lane changes on a freeway (a situation that has been anecdotally noted to be difficult [49]). I present a method for quantifying the performance gains that could result from perfect estimation of and planning with hidden behavior model parameters. In addition, I show that when model parameters are correlated, estimating the parameters online using physical measurements can greatly improve performance. Planning using a POMDP problem formulation that dynamically takes uncertainty into account can further improve performance.

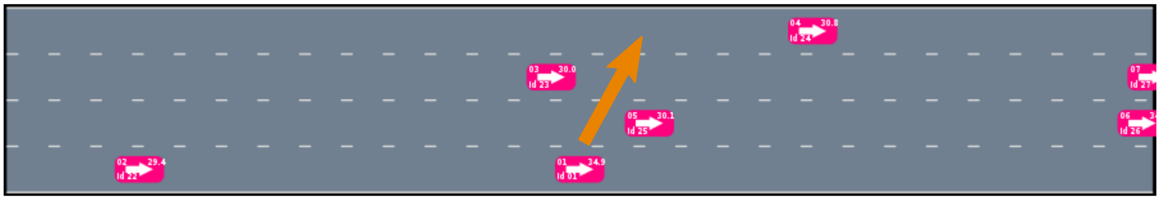


Figure 3.1: The example decision making scenario for this chapter. An autonomous vehicle (bottom center) must travel from the rightmost to leftmost lane within a limited distance.

3.2 Freeway Driving POMDP

The focus of this chapter is on freeway driving. I investigate a scenario in which a vehicle must navigate from the rightmost to the leftmost lane of a four lane freeway within a specified distance while maintaining safety and comfort (see Fig. 3.1).

Throughout this section, x denotes position in the *longitudinal* direction, that is, the direction that the cars move along the road in meters, and y denotes position in the *lateral* direction, that is, the lane the car occupies in lane units. The problem can be stated as a discrete-time POMDP defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, \mathcal{Z})$, which consists of

- The state space, $\mathcal{S}$: A system state,

  $$s = (q_0, \{(q_i, \theta_i)\}_{i \in 1..N}) \in \mathcal{S},$$

  consists of the physical state of the ego vehicle (q_0), and physical state and behavior model for each of the N other cars in the scene. The physical state,

  $$q_i = (x_i, y_i, \dot{x}_i, \dot{y}_i),$$

  consists of the car's longitudinal and lateral position and velocity. The internal state (behavior model parameters), θ_i, is drawn from a set of behaviors Θ.

- The action space, $\mathcal{A}$: An action, $u = (\ddot{x}_e, \ddot{y}_e) \in \mathcal{A}$, consists of the longitudinal acceleration and lateral velocity of the ego vehicle. The action space is discrete and pruned to prevent crashes (see Section 3.2.3).

- The state transition model, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$: The value $\mathcal{T}(s, u, s')$ is the probability of transitioning to state s' given that action u is taken by the ego at state s. This function is implicitly defined by a generative model that consists of a state transition function, $F(\cdot)$, and a stochastic noise process (see Section 3.2.2).
- The reward model, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$: The reward function, defined in Section 3.2.4, rewards reaching the left lane within the distance limit and penalizes unsafe actions.
- The observation space, $\mathcal{O}$: An observation, $o \in \mathcal{O}$ consists of the physical states of all of the vehicles, that is $o = \{p_i\}_{i \in 1..N}$. No information about the internal state is directly included in the observation.
- The observation model, $Z : S \times O \to \mathbb{R}$: The value $Z(s', o)$ is the probability of receiving observation o when the system transitions to state s'. In these experiments, the physical state is assumed to be known exactly, though it is not difficult to relax this assumption.

The remainder of this section elaborates on this model.

3.2.1 Driver Modeling

The driver models for each car have two components: an acceleration model that governs the longitudinal motion and a lane change model that determines the lateral motion. In this chapter, the acceleration model is the Intelligent Driver Model (IDM) [69], and the lane change model is the “Minimizing Overall Braking Induced by Lane change” (MOBIL) model [33]. Both of these models have a small number of parameters that determine the behavior of the drivers. The distribution of these parameters in the population of vehicles will be denoted Θ.

IDM

The IDM Model was developed as a simple model for “microscopic” simulations of traffic flows and is able to reproduce some phenomena observed in real-world traffic flows. It determines the longitudinal acceleration for a human-driven car, $\ddot{x}$, based on the desired distance gap to the preceding car, g, the absolute velocity, $\dot{x}$, and the velocity relative to the preceding car $\Delta\dot{x}$. The longitudinal acceleration is governed by the following equation:

$$\ddot{x}_{\text{IDM}} = a\left[1 - \left(\frac{\dot{x}}{\dot{x}_0}\right)^{\delta} - \left(\frac{g^*(\dot{x}, \Delta\dot{x})}{g}\right)^2\right], \tag{3.1}$$

where g^* is the desired gap given by

$$g^*(\dot{x}, \Delta\dot{x}) = g_0 + T\dot{x} + \frac{\dot{x}\Delta\dot{x}}{2\sqrt{ab}}. \tag{3.2}$$

Brief descriptions and values for the parameters not defined here are provided later in Table 3.1.

A small amount of noise is also added to the acceleration

$$\ddot{x} = \ddot{x}_{\text{IDM}} + w, \tag{3.3}$$

where w is a random variable with a triangular distribution with support between $-a/2$ and $a/2$. In cases where the noise might cause a hard brake or lead to a state where a crash is unavoidable, the distribution is scaled appropriately.

MOBIL

The MOBIL model makes the decision to change lanes based on maximizing the acceleration for the vehicle and its neighbors. When considering a lane change, MOBIL first ensures that the safety criterion $\tilde{\ddot{x}}_{\text{follow}} \geq -b_{\text{safe}}$, where $\ddot{x}_n$ will be the acceleration of the following car if the lane change is made and b_{safe} is the safe braking limit. It then makes the lane change if the following condition is met

$$\tilde{\ddot{x}}_c - \ddot{x}_c + p\left(\tilde{\ddot{x}}_n - \ddot{x}_n + \tilde{\ddot{x}}_o - \ddot{x}_o\right) > \Delta a_{\text{th}} \tag{3.4}$$

where the quantities with tildes are calculated assuming that a lane change is made, the quantities with subscript c are quantities for the car making the lane change decision, those with n are for the new follower, and those with o are for the old follower. The parameter $p \in [0,1]$ is the politeness factor, which represents how much the driver values allowing other vehicles to increase their acceleration. The parameter Δa_{th} is the threshold acceleration increase to initiate a lane changing maneuver. Parameter values are listed in Table 3.1.

3.2.2 Physical Dynamics

The physical dynamics are simplified for the sake of computational efficiency. Time is divided into discrete steps of length Δt. The longitudinal dynamics assume constant acceleration, and the lateral dynamics assume constant velocity over a time step, that is

$$\begin{aligned} x' &= x + \dot{x}\Delta t + \frac{1}{2}\ddot{x}\Delta t^2 \\ \dot{x}' &= \dot{x} + \ddot{x}\Delta t \\ y' &= y + \dot{y}\Delta t. \end{aligned}$$

There is a physical limit to the braking acceleration, b_{max}. Lateral velocity is allowed to change instantly because cars on a freeway can achieve the lateral velocity needed for a lane change in time much shorter than Δt by steering. If MOBIL determines that a lane change should be made, the lateral velocity, $\dot{y}$, is set to $\dot{y}_{\text{lc}}$. Lane changes are not allowed to reverse. Once a lane change has begun, $\dot{y}$ remains constant until the lane change is completed (this is the reason that $\dot{y}$ is part of the state). When a vehicle passes over the midpoint of a lane, lateral movement is immediately stopped so that lane changes always end at exactly the center of a lane.

Since MOBIL only considers cars in adjacent lanes, there must be a coordination mechanism so that two cars do not converge into the same lane simultaneously. In order to accomplish this, if two cars begin changing into the same lane simultaneously, and the front vehicle is within g^* of the rear vehicle, the rear vehicle's lane change is canceled.

In order to reduce the computational demands of decision-making, only 50 m of road in front of the ego and 50 m behind are modeled. Thus, a model for vehicle entry into this section is needed. If there are fewer than N_{max} vehicles on the road, a new vehicle is generated. First, a behavior for the new vehicle is drawn from Θ, and the initial speed is set to $\dot{x}_0 + \sigma_{\text{vel}} w_0$, where $\dot{x}_0$ is the desired speed from the behavior model and w_0 is a zero-mean, unit-variance, normally distributed random variable that is independent for each car. If this speed is greater than the ego's speed, the new vehicle will appear at the back of the road section; if it is less, it will appear at the front. For each lane, g^* is calculated, either for the new vehicle if the appearance is at the back or for the nearest following vehicle if the appearance is at the front. The new vehicle appears in the lane where the clearance to the nearest car is greatest. If no clearance is greater than g^*, the new vehicle does not appear.

Once the ego reaches the target lane ($y = y_{\text{target}}$) or passes the distance limit ($x \geq L$), the problem terminates.

For convenience, throughout this chapter, the behavior described so far will be denoted compactly by the state transition function

$$s' = F(s, u, w). \tag{3.5}$$

3.2.3 Action Space for Crash-Free Driving

At each time step, the planner for the ego must choose the longitudinal and lateral acceleration. For simplicity, the vehicle chooses from up to ten discrete actions which are shown in Fig. 3.2. The vehicle may make an incremental decrease or increase in speed or maintain speed, and it may begin a left or right lane change or maintain the current lane. The combination of these adjustments make up nine of the actions. The final action is a braking action determined dynamically based on the speed and position of the vehicle ahead. At each time step, the maximum permitted acceleration, a_{max}, is the maximum acceleration that the ego could take such that, if the vehicle ahead immediately begins braking at the physical limit, b_{max}, to a stop, the ego will still be able to stop before hitting it without exceeding physical braking limits itself.

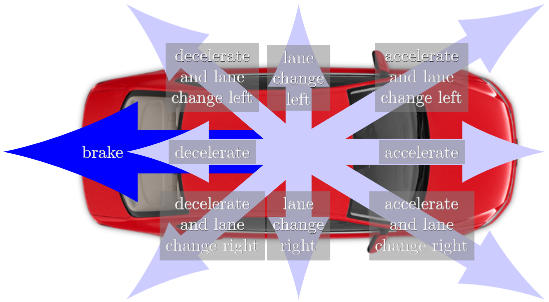

decelerate and lane change left
lane change left
accelerate and lane change left
brake
decelerate
accelerate
decelerate and lane change right
lane change right
accelerate and lane change right

Figure 3.2: Lane changing action space

The braking action is $(\ddot{x}_e, \ddot{y}_e) = (\min\{a_{\text{max}}, -b_{\text{nominal}}\}, 0)$.

The inclusion of the dynamic braking action guarantees that there will always be an action available to the ego to avoid a crash. At each step, the action space is pruned so that if $\ddot{x}_e > a_{\text{max}}$ or if a lane change leads to a crash, that action is not considered. Since the IDM and MOBIL models are both crash-free [32], and actions that lead to crashes for the ego are not considered, no crashes occur in the simulation. Eliminating crashes in our model is justifiable because it is likely that in an actual autonomous vehicle a high-level planning system would be augmented with a low-level crash prevention system to increase safety and facilitate certification. In addition, it is difficult to model driver behavior in the extraordinary case of a crash.

3.2.4 Reward Function and Objectives

The qualitative objectives in solving this problem are to reach the target lane within a specified distance, L, and maintain the comfort and safety of both the ego and the other nearby vehicles. Thus, the following two metrics will be used to evaluate planning performance: 1) the fraction of episodes in which the ego reaches the target lane, and 2) the fraction of episodes in which *any* vehicle operates in an unsafe manner. For this work, hard braking and unusually slow velocity are considered unsafe. A hard braking maneuver is defined as $\ddot{x} < -b_{\text{hard}}$ and slow velocity as $\dot{x} < \dot{x}_{\text{slow}}$, where b_{hard} and $\dot{x}_{\text{slow}}$ are chosen to be uncomfortably abrupt deceleration or slow travel that might result in an accident in real conditions (see Table 3.2). In addition to quantifying safety, hard braking also serves as a proxy for comfort.

In order to encourage the planner to choose actions that will maximize these metrics, the reward function for the POMDP is defined as follows:

$$\mathcal{R}(s, a, s') = \text{in\_goal}(s') - \lambda \left(\text{any\_hard\_brakes}(s, s') + \text{any\_too\_slow}(s')\right) \tag{3.6}$$

where

$$\text{in\_goal}(s') = \mathbf{1}(y_e = y_{\text{target}}, x_e \leq L), \tag{3.7}$$

$$\text{any\_hard\_brakes}(s, s') = \max_{i \in 1..N} \{\mathbf{1}(\dot{x}_i' - \dot{x}_i < -b_{\text{hard}} \Delta t)\}, \tag{3.8}$$

$$\text{any\_too\_slow}(s') = \max_{i \in 1..N} \{\mathbf{1}(\dot{x}_i < \dot{x}_{\text{slow}})\}. \tag{3.9}$$

That is, there is a positive reward for reaching the target lane within the distance limit, and hard brakes and slow velocity for any car are penalized. The weight λ balances the competing goals and can be adjusted to create an approximate curve of Pareto-optimal solutions.

3.2.5 Initial Scenes

Initial scenes for the simulations are generated by beginning a simulation with only the ego on the road section and then simulating 200 steps to allow other vehicles to accumulate in the scene.

3.3 Solution Approaches

Each of the solution techniques is based on MCTS-DPW (Sections 1.2.5 to 1.2.5), but handles uncertainty in the internal states differently.

3.3.1 Approach 1: Assume normal behavior

The first performance baseline is established by planning as if all cars behave according to a single static “normal” internal state (see Table 3.1). In this case, the problem is an MDP, which is solved using the MCTS-DPW algorithm. This is an overconfident baseline — it plans assuming it knows more about the other drivers than is justified by the information it has collected.

3.3.2 Approach 2: Model all uncertainty as outcome uncertainty (Naive MDP)

The second performance baseline is established by planning as if all uncertainty is simply outcome uncertainty, that is, as if the problem were an MDP with a state consisting only of the physical state and the internal states random variables, independent at each timestep, distributed according to the internal state distribution Θ. This model would be the result of fitting a Markov model with only the physical state based on data from all drivers. The MDP is again solved using the MCTS-DPW algorithm.

3.3.3 Approach 3: Mean Model Predictive Control

Since information about the human's internal state can be inferred by observing the car's physical motion, performance superior to either of the baselines can be achieved by estimating θ online. This is accomplished with a particle filter (see Section 1.2.6, [67]). Filtering is independent for each car, but all of the behavior parameters for a given car are estimated jointly. There are two versions of the filter. In the first version, a particle, $\hat{\theta}$, consists of values of all model parameters. In the second version, all parameters are assumed perfectly correlated (see Section 3.4.1), so a particle consists of only a single value, the "aggressiveness".

The belief at a given time consists of the exactly known physical state, q, and a collection of M particles, $\{\hat{\theta}^k\}_{k=1}^M$, along with associated weights, $\{W^k\}_{k=1}^M$. To update the belief when action u is taken, M new particles are sampled with probability proportional to the weights, and sampled noise values $\{\hat{w}^k\}_{k=1}^M$ are used to generate new states according to $\hat{s}^{k\prime} = F((q, \hat{\theta}^k), u, \hat{w}^k)$. The new weights are determined by approximating the conditional probability of the particle given the observation:

$$W^{k\prime} = \left\{ \begin{array}{ll} \max\left\{0, \frac{a-2\left|\dot{x}'-\hat{\dot{x}}'\right|}{a}\right\} & \text{if } y' = \hat{y}' \\ \gamma_{\text{lane}} \max\left\{0, \frac{a-2\left|\dot{x}'-\hat{\dot{x}}'\right|}{a}\right\} & \text{o.w.} \end{array} \right\} \underset{\approx}{\propto} \Pr\left(\hat{\theta}^k \middle| o\right)$$

where $\dot{x}'$ and y' are taken from the observation, $\hat{\dot{x}}'$ and $\hat{y}'$ are from $\hat{s}^{k\prime}$, the max expression is proportional to the probability density of the acceleration noise triangular distribution (ignoring the effects of the collision mitigation modifications), and $\gamma_{\text{lane}} \in [0, 1]$ is a hand-tuned parameter that penalizes incorrect lane changes (see Table 3.2).

Model predictive control (MPC) is a widely used family of control techniques that use an imperfect model and feedback measurements to choose actions [22]. At each time step, a model predictive controller calculates a sequence of control actions that will maximize a reward function of the states visited up to a future horizon given that the system behaves according to a model. The first control action in this optimized sequence is executed, and the process is repeated after a new measurement is received.

In the mean model predictive control (MMPC) approach, this particle filter is used to estimate the internal state for each driver. At each step, MPC uses the MDP that results from assuming that each driver has the internal state corresponding to the mean of the particles in the belief approximation as the model for planning. Each time a new observation is received, the particle filter is updated and MCTS-DPW determines the best action for the resulting MDP.

3.3.4 Approach 4: QMDP

The fourth approach uses MCTS-DPW to solve the QMDP approximation of the POMDP (see Section 1.2.8).

3.3.5 Approach 5: POMCPOW

The final approach uses the POMCPOW solver described in Section 4.2.3. Though there is no theoretical guarantee that this approach will converge to an optimality, this is the closest approximation to the exact POMDP solution

3.4 Results

The computational results from this study are designed to meet the two goals of 1) quantifying the size of the gap between the baseline control algorithm and the maximum potential lane change performance and 2) showing which cases internal state estimation and POMDP planning can approach the upper bound on performance. Experiments are carried out in three scenarios, each with a different distribution of internal states. In each of these scenarios, each of the approaches described in **??** are compared with an approximate upper performance bound obtained by planning with perfect knowledge of the behavior models.

3.4.1 Driver Model Distribution Scenarios

For the numerical testing, three internal state distribution scenarios were considered. In all of these scenarios, drivers behave according to the IDM and MOBIL models presented in Section 3.2.1, however the IDM and MOBIL parameter values are distributed differently.

Table 3.1 shows typical parameter values for aggressive, timid, and normal drivers. The values are taken from Kesting, Treiber, and Helbing [32], but some have been adjusted slightly so that the parameters for the normal driver are exactly half way between values for the timid and aggressive drivers. In all three of the scenarios, the *marginal* distributions of the parameters are uniformly distributed between the aggressive and timid values. The difference between the scenarios is the correlation of the parameter values. In Scenario 1, all of the parameters are independently distributed. In Scenario 2, all of the parameters are perfectly correlated so that all parameters are deterministic functions of the aggressiveness of the driver. Scenario

3 uses a distribution between these two extremes. In this scenario, values are drawn from a Gaussian copula with covariance matrix

$$\Sigma = \begin{bmatrix} 1 & \dots & \rho \\ \vdots & \ddots & \vdots \\ \rho & \dots & 1 \end{bmatrix}, \tag{3.10}$$

that is, a matrix with 1 along the diagonal and ρ elsewhere. The values are then scaled and translated to lie between the aggressive and normal limits. For Scenario 3, the value of ρ is 0.75, and Scenarios 1 and 2 can be thought of as limiting cases where ρ approaches 0 and 1, respectively. In Scenario 1, the first version of the particle filter, which estimates all of the model parameters jointly, is used, whereas in Scenarios 2 and 3, the second version of the particle filter that assumes fully correlated parameters is used, that is, it only estimates a single "aggressiveness" parameter for each car. The small scatter plots in Figs. 3.3 and 3.4 illustrate the level of correlation by plotting sampled values of two of the parameters.

Table 3.1: IDM and MOBIL parameters for different driver types.

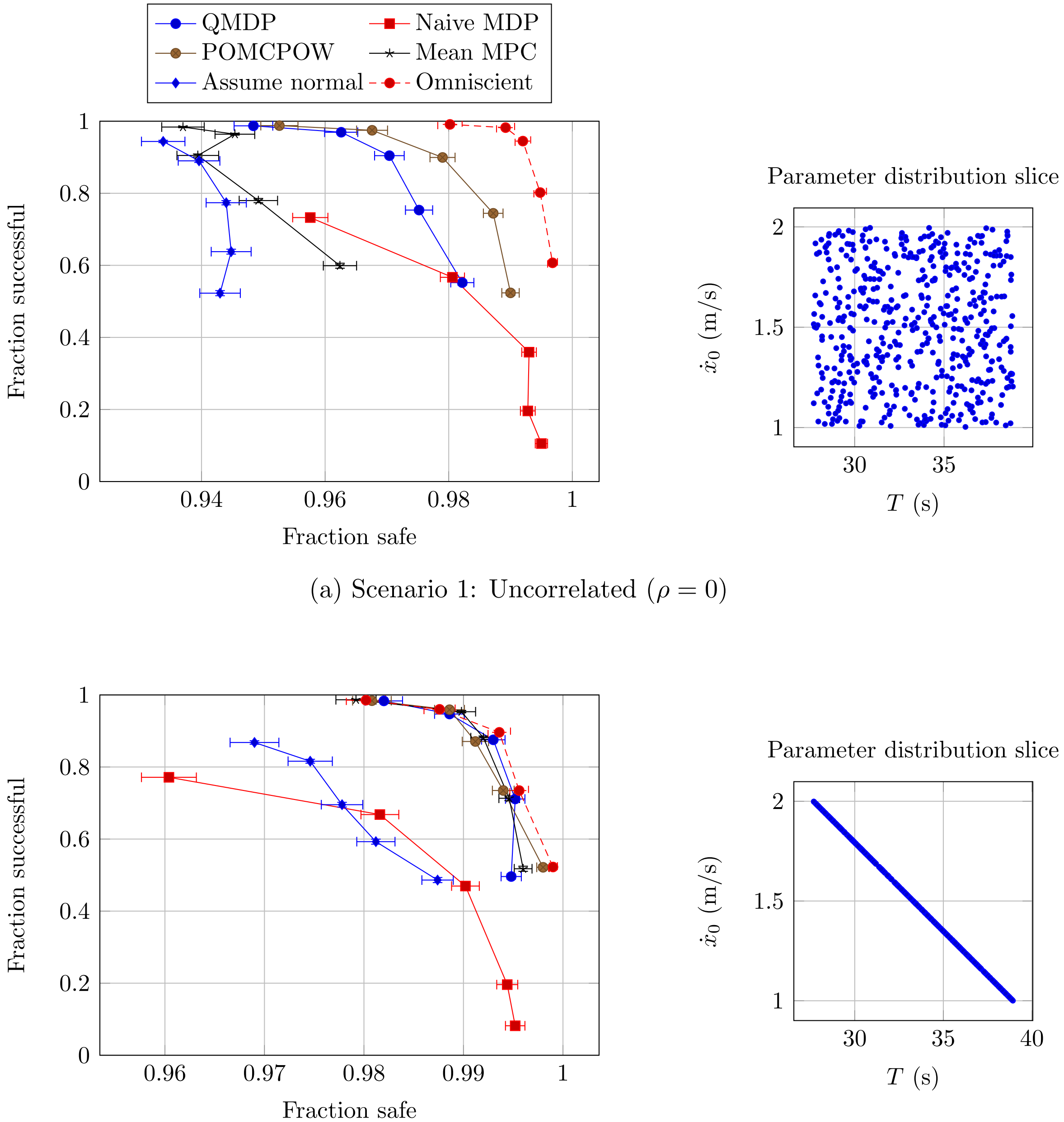
IDM Parameter		Timid	Normal	Aggressive
Desired speed (m/s)	$\dot{x}_0$	27.8	33.3	38.9
Desired time gap (s)	T	2.0	1.5	1.0
Jam distance (m)	g_0	4.0	2.0	0.0
Max acceleration (m/s^2)	a	0.8	1.4	2.0
Desired deceleration (m/s^2)	b	1.0	2.0	3.0
MOBIL Parameter		Timid	Normal	Aggressive
Politeness	p	1.0	0.5	0.0
Safe braking (m/s^2)	b_{safe}	1.0	2.0	3.0
Acceleration threshold (m/s^2)	a_{thr}	0.2	0.1	0.0

3.4.2 Performance Results

Figures 3.3 and 3.4 show approximate Pareto curves illustrating the performance in terms of safety and efficiency of each of the approaches described in **??**. Each of the points on the curve shows the results of 5000 independent simulations of the scenario

Table 3.2: Various simulation parameters

Parameter	Symbol	Value
Simulation time step	Δt	0.75 s
Max vehicles on road	N_{max}	10
Lane change rate	$\dot{y}_{\text{lc}}$	0.67 lanes/s
Distance limit	L	1000 m
Velocity noise standard deviation	σ_{vel}	0.5 m/s
Physical braking limit	b_{max}	8.0 m/s^2
Penalized hard braking limit	b_{hard}	4.0 m/s^2
Penalized minimum speed	$\dot{x}_{\text{slow}}$	15 m/s
UCT exploration parameter	c	8
DPW linear parameter	k	4.5
DPW exponent parameter	α	0.1
MCTS search depth		40
MCTS iterations per step		1000
Particle filter wrong lane factor	γ_{lane}	0.05
Number of Particles (Joint Parameter Filter)	M	5000
Number of Particles (Aggressiveness Filter)	M	2000
Reward ratios for points on approximate Pareto curves	λ	0.5, 1, 2, 4, 8



(a) Scenario 1: Uncorrelated ($\rho = 0$)

(b) Scenario 2: Fully correlated ($\rho = 1$)

Figure 3.3: Approximate Pareto performance curves for model distributions at correlation extremes. The scatter plots at right illustrate the level of correlation with samples from the joint parameter distribution. Error bars indicate the standard error of the mean.

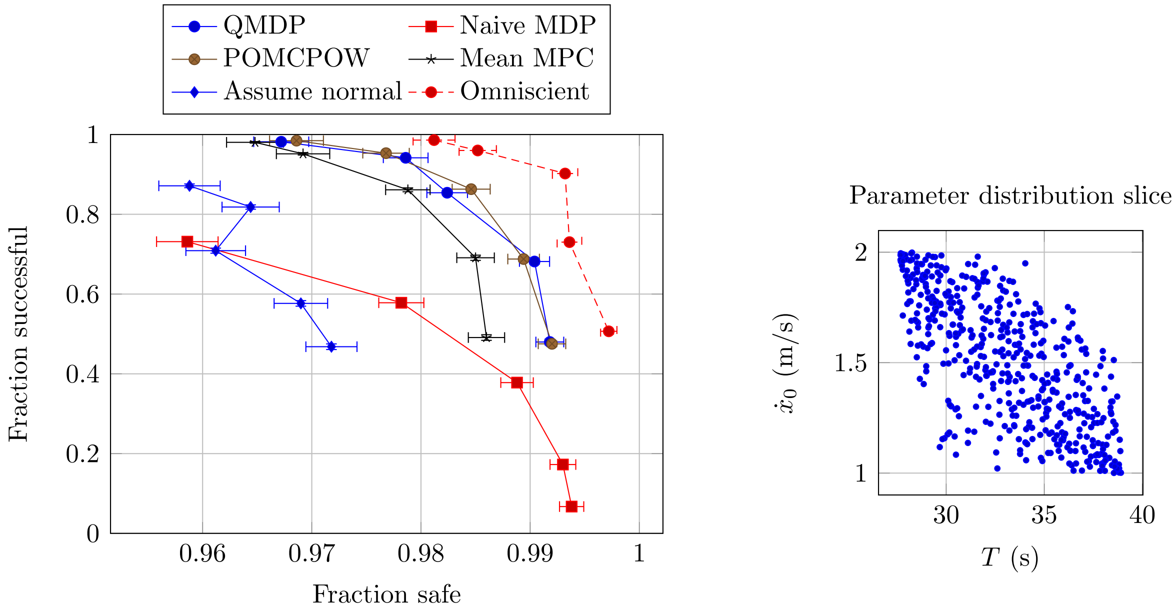


Figure 3.4: Approximate Pareto performance curves for partially correlated model distribution (Scenario 3: $\rho = 0.75$). Error bars indicate the standard error of the mean.

with the given safety-efficiency tradeoff weight, λ.

Control approach comparison

The baseline and upper bound approaches perform as expected. The baseline planner that assumes all vehicles act with normal behavior parameters creates over-confident plans. That is, it is able to reach the goal a large proportion of the time, but it causes many safety violations. On the other hand, the naive MDP approximation is over-cautious. That is, it can attain a high level of safety, but it is never able to meet the goal more than 80 % of the time. The omniscient upper bound planner achieves performance equal to or greater than all other approaches.

As expected, better plans are attained as more accurate uncertainty is modeled in planning. The mean MPC approach usually performs better than the baselines because it estimates the model parameters based on the physical states it observes, but it is still overconfident (achieving a high success rate, but sacrificing safety) because it plans without any internal state uncertainty. QMDP performs better than mean MPC because it considers samples from the entire estimated internal state distribution when planning. Since the vehicle does not have to take costly information-gathering actions to accomplish its goal, POMCPOW only rarely outperforms QMDP.

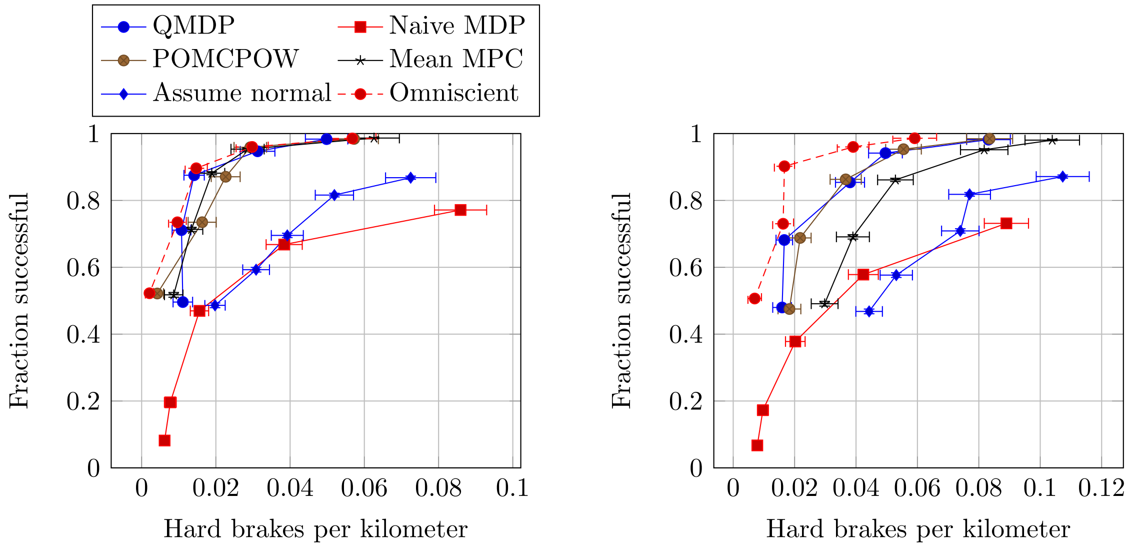
Convexity violations

One immediate concern that should be raised about the approximate Pareto frontiers in Figs. 3.3 and 3.4 is that they are not all convex. In an optimization problem where the objective is a linear combination of several performance metrics, all optimal solutions for all possible linear combinations of objectives must lie on the boundary of their mutual convex hull (see Boyd and Vandenberghe [10], Example 2.27). Particularly egregious violations of convexity can be found in the mean MPC curve in Fig. 3.3a and the normal behavior assumption curve in Fig. 3.4, where there are "kinks" at the third point from the top ($\lambda = 2$) that prevent these curves from even being monotonic.

The lack of convexity may be due to some combination of the following reasons:

1. The performance objectives plotted in the graphs do not exactly match the stepwise reward function 3.6. For example, the planner observes a larger penalty if there are multiple safety violations, but this is not reflected in the plots.
2. The MCTS-DPW solution method is itself stochastic and has no guarantees of convergence in finite time.
3. The solvers are solving inaccurate approximations of the true POMDP problem (except, perhaps, for POMCPOW).

One compelling explanation for the kinks mentioned above is that, as λ is increased, since the planner is penalized more severely for unsafe actions, it plans a more conservative trajectory and stays on the road longer. The longer time on the road gives more chances for unsafe events to occur which are difficult for the planner to avoid because of its inaccurate model. This explanation is corroborated by the results in Fig. 3.5. In both places where there were previously kinks, the number of hard brakes per kilometer decreases as λ increases.



(a) Fully correlated ($\rho = 1$)

(b) Partially correlated ($\rho = 0.75$)

Figure 3.5: Average hard braking frequency and success rate

Correlation comparison

It is also interesting to consider the Affect that the correlation between model parameters has on the relative effectiveness of the control approaches. Figure 3.3a shows that when there is no correlation, QMDP offers a significant advantage over mean MPC, and POMCPOW offers a further significant advantage over QMDP. In this case, since the parameters are uncorrelated, there is a large amount of uncertainty in them even when some (e.g. $\dot{x}_0$) are easy to observe, and since POMCPOW is able to plan into the future considering this uncertainty, it performs better. On the other hand, when the parameters are fully correlated as shown in Fig. 3.3b, all of the parameters are easy to estimate by observing only a few, so there is not a significant performance gap between mean MPC, QMDP, and POMCPOW; all are able to close the gap and achieve nearly the same performance as the upper bound. Figure 3.4 shows the expected behavior between the extremes.

Figure 3.6 shows the performance gaps at more points between $\rho = 0$ and 1. As the correlation increases, the approximate POMDP planning approaches get steadily closer to closing the performance gap with the upper bound. These results have significant implications for the real world. It suggests that if most human driver behavior is correlated with easily measurable quantities, near-optimal performance can be achieved by simpler approaches like mean MPC. If there is little correlation, planning with internal states offers a less pronounced benefit, and more advanced planners that carry the uncertainty further into the future are needed to realize that benefit.

add robustness study

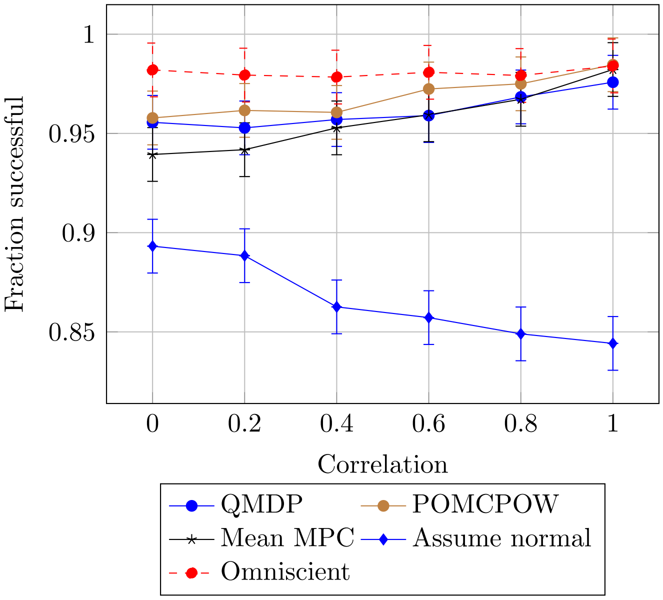


Figure 3.6: Performance variation with Θ correlation. Success is defined as reaching the target lane within the distance limit without any unsafe actions. Error bars indicate the 68 % (corresponding to one standard deviation in a normal distribution) confidence region determined by the Hoeffding Bound. The Naive MDP performance is not shown because it is significantly lower than the other approaches.

Chapter 4

Online Algorithms for Continuous POMDPs

4.1 Background

Although the research surveyed in Section 1.2.7 has yielded effective solution techniques for many classes of POMDPs, there remains a need for simple, general purpose online solvers that can handle continuous spaces, especially continuous observation spaces. This chapter takes some first steps toward addressing that need.

POMCP and other online methods can easily accomodate continuous state spaces without any modification [24]. However, there has been less progress on problems with continuous observation spaces. This chapter presents two similar algorithms which address the challenge of solving POMDPs with continuous state, action, and observation spaces. The first is based on POMCP and is called partially observable Monte Carlo planning with observation widening (POMCPOW). The second solves the belief-space MDP and is called particle filter trees with double progressive widening (PFT-DPW).

There are two challenges that make tree search difficult in continuous spaces. The first is that, since the probability of sampling the same real number twice from a continuous random variable is zero, the width of the planning trees explodes on the first step, causing them to be too shallow to be useful (see Fig. 4.1). POMCPOW

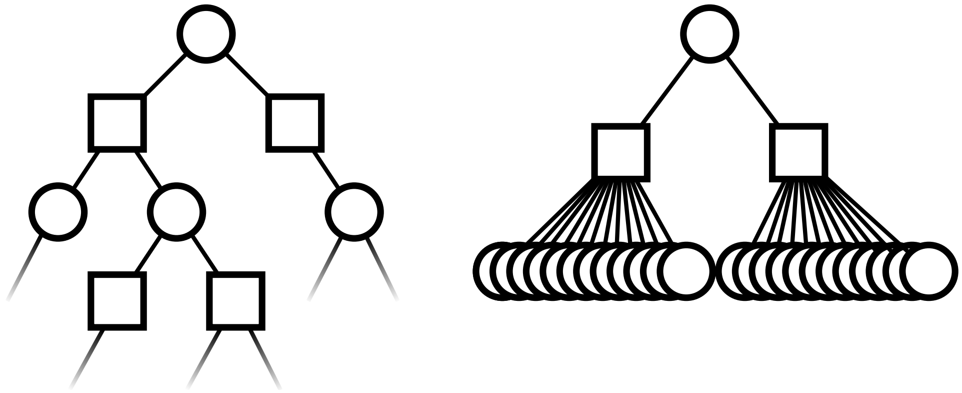


Figure 4.1: POMCP tree for a discrete POMDP (left), and for a POMDP with a continuous observation space (right). Because the observation space is continuous, each simulation creates a new observation node and the tree cannot extend deeper.

and PFT-DPW resolve this issue with a technique called double progressive widening (DPW) [17]. The second issue is that, even when DPW is applied, the belief representations used by current solvers collapse to a single state particle, resulting in overconfidence. As a consequence, the solutions obtained resemble QMDP policies, and there is no incentive for information gathering. POMCPOW and PFT-DPW overcome this issue by using the observation model to weight the particles used to represent beliefs.

A small amount of previous research has sought online solutions to continuous POMDPs. ABT has been extended to use generalized pattern search for selecting locally optimal continuous actions, an approach which is especially effective in problems where high precision is important [60]. Continuous observation Monte Carlo tree search (COMCTS) constructs observation classification trees to automatically partition the observation space in a POMCP-like approach, however it did not perform much better than a Monte Carlo rollout approach in experiments [51].

4.2 Algorithms

This section presents and discusses several methods for handling continuous POMDPs. First, it presents three new MCTS-DPW-based algorithms. The first of these is

POMCP-DPW. Theorem 1 shows that this algorithm is suboptimal in some cases. The second algorithm is PFT-DPW, a straightforward application of MCTS-DPW to the belief MDP with particle filters to approximate belief updates. The third algorithm is POMCPOW, which fixes the problems noted about POMCP-DPW and successfully extends POMCP to problems with continuous observation spaces. These three algorithms are followed by brief discussion about the discretization alternative to these algorithms and the additional observation distribution requirement of POMCPOW and PFT-DPW.

The three new algorithms in this chapter share a common structure. For all algorithms, the entry point for the decision making process is the Plan procedure, which takes the current belief, b, as an input (Plan differs slightly for PFT-DPW in Algorithm 2). The algorithms also share the same ActionProgWiden function to control progressive widening of the action space. These components are listed in Listing 2. The difference between the algorithms is in the Simulate function.

The following variables are used in the listings and text: h represents a history $(b, a_1, o_1, \ldots a_k, o_k)$, and ha and hao are shorthand for histories with a and (a, o) appended to the end, respectively; d is the depth to explore, with $d_{\max}$ the maximum depth; C is a list of the children of a node (along with the reward in the case of PFT-DPW); N is a count of the number of visits; and M is a count of the number of times that a history has been generated by the model. The list of states associated with a node is denoted B, and W is a list of weights corresponding to those states. Finally, $Q(ha)$ is an estimate of the value of taking action a after observing history h. C, N, M, B, W, and Q are all implicitly initialized to 0 or $\emptyset$. The Rollout procedure, runs a simulation with a default rollout policy, which can be based on the history or fully observed state, for d steps and returns the discounted reward.

4.2.1 POMCP-DPW

The first algorithm that we consider is POMCP with double progressive widening (POMCP-DPW). In this algorithm, listed in Algorithm 1, the number of new children sampled from any node in the tree is limited by DPW using the parameters k_a, α_a,

Listing 2 Common procedures

```
procedure PLAN($b$)
    for $i \in 1 : n$ do
        $s \leftarrow$ sample from $b$
        SIMULATE($s, b, d_{\max}$)
    return $\underset{a}{\operatorname{argmax}}\, Q(ba)$
procedure 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 $\underset{a \in C(h)}{\operatorname{argmax}}\, Q(ha) + c\sqrt{\frac{\log N(h)}{N(ha)}}$
```

k_o, and α_o. In the case where the simulated observation is rejected (line 14), the tree search is continued with an observation selected in proportion to the number of times, M, it has been previously simulated (line 15) and a state is sampled from the associated belief (line 16).

This algorithm obtained remarkably good solutions for a very large autonomous freeway driving POMDP with multiple vehicles (up to 40 continuous fully observable state dimensions and 72 continuous correlated partially observable state dimensions) [64]. To our knowledge, that is the first work applying progressive widening to POMCP, and it does not contain a detailed description of the algorithm or any theoretical or experimental analysis other than the driving application.

This algorithm may converge to the optimal solution for POMDPs with discrete observation spaces; however, on continuous observation spaces, POMCP-DPW is suboptimal. In particular, it finds a QMDP policy, that is, the solution under the assumption that the problem becomes fully observable after one time step [44, 34]. In fact, for a modified version of POMCP-DPW, it is easy to prove analytically that it will converge to such a policy. This is expressed formally in Theorem 1 below. A complete description of the modified algorithm and problem requirements including the definitions of polynomial exploration, the regularity hypothesis for the problem, and exponentially sure convergence are given in Appendix A.

Algorithm 1 POMCP-DPW

```
procedure Simulate($s$, $h$, $d$)
    if $d = 0$ then
        return 0
    $a \leftarrow$ ActionProgWiden($h$)
    if $|C(ha)| \leq k_o N(ha)^{\alpha_o}$ then
        $s', o, r \leftarrow G(s, a)$
        $C(ha) \leftarrow C(ha) \cup \{o\}$
        $M(hao) \leftarrow M(hao) + 1$
        append $s'$ to $B(hao)$
        if $M(hao) = 1$ then
            $\mathit{total} \leftarrow r + \gamma \textsc{Rollout}(s', hao, d-1)$
        else
            $\mathit{total} \leftarrow r + \gamma \textsc{Simulate}(s', hao, d-1)$
    else
        $o \leftarrow$ select $o \in C(ha)$ w.p. $\frac{M(hao)}{\sum_o M(hao)}$
        $s' \leftarrow$ select $s' \in B(hao)$ w.p. $\frac{1}{|B(hao)|}$
        $r \leftarrow R(s, a, s')$
        $\mathit{total} \leftarrow r + \gamma \textsc{Simulate}(s', hao, d-1)$
    $N(h) \leftarrow N(h) + 1$
    $N(ha) \leftarrow N(ha) + 1$
    $Q(ha) \leftarrow Q(ha) + \frac{\mathit{total} - Q(ha)}{N(ha)}$
    return $\mathit{total}$
```

Definition 1 (QMDP value). *Let $Q_{MDP}(s, a)$ be the optimal state-action value function assuming full observability starting by taking action a in state s. The* QMDP value *at belief b, $Q_{MDP}(b, a)$, is the expected value of $Q_{MDP}(s, a)$ when s is distributed according to b.*

Theorem 1 (Modified POMCP-DPW convergence to QMDP). *If a bounded-horizon POMDP meets the following conditions: 1) the state and observation spaces are continuous with a finite observation probability density function, and 2) the regularity hypothesis is met, then modified POMCP-DPW will produce a value function estimate, $\hat{Q}$, that converges to the QMDP value for the problem. Specifically, there exists a constant $C > 0$, such that after n iterations,*

$$\left|\hat{Q}(b, a) - Q_{MDP}(b, a)\right| \leq \frac{C}{n^{1/(10 d_{\max} - 7)}}$$

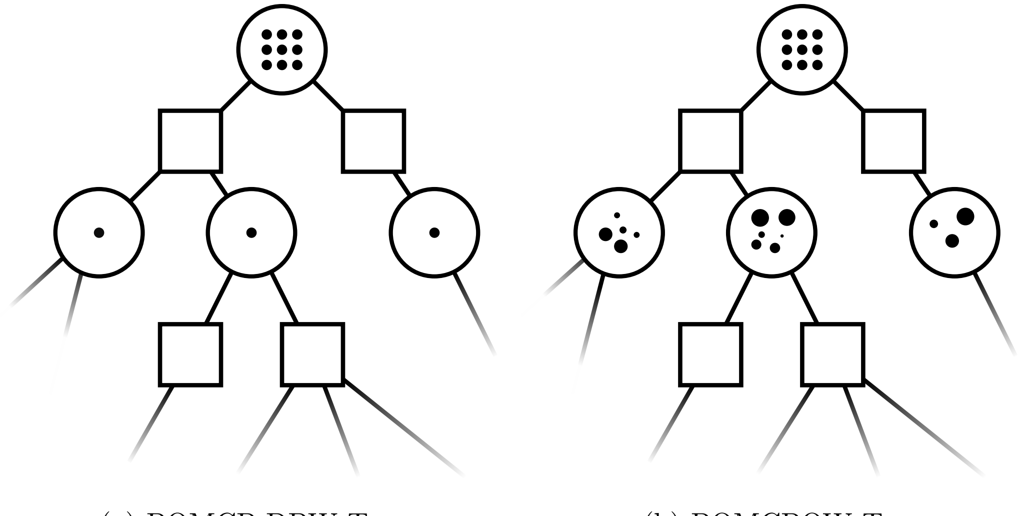
exponentially surely in n, for every action a.

A proof of this theorem that leverages work by Auger, Couetoux, and Teytaud [3] is given in Appendix A, but I will provide a brief justification here. The key is that belief nodes will contain only a single state particle (see Fig. 4.2). This is because, since the observation space is continuous with a finite density function, the generative model will (with probability one) produce a unique observation o each time it is queried. Thus, for every generated history h, only one state will ever be inserted into $B(h)$ (line 9, Algorithm 1), and therefore h is merely an alias for that state. Since each belief node corresponds to a state, the solver is actually solving the fully observable MDP at every node except the root node, leading to a QMDP solution.

As a result of Theorem 1, the action chosen by modified POMCP-DPW will match a QMDP policy (a policy of actions that maximize the QMDP value) with high precision exponentially surely (see Corollary 1 of Auger, Couetoux, and Teytaud [3]). For many problems this is a very useful solution,[1] but since it neglects the value of information, a QMDP policy is suboptimal for problems where information gathering is important [44, 34].

[1] Indeed, a useful online QMDP tree search algorithm could be created by deliberately constructing a tree with a single root belief node and fully observable state nodes below it.

Although Theorem 1 is only theoretically applicable to the modified version of POMCP-DPW, it helps explain the behavior of other solvers. Modified POMCP-DPW, POMCP-DPW, DESPOT, and ABT all share the characteristic that a belief node can only contain two states if they generated exactly the same observation. Since this is an event with zero probability for a continuous observation space, these solvers exhibit suboptimal, often QMDP-like, behavior. The experiments in Section 4.3.6 show this for POMCP-DPW and DESPOT, and this is presumably the case for ABT as well.



(a) POMCP-DPW Tree

(b) POMCPOW Tree

Figure 4.2: Tree structure comparison. Each square is an action node, and each unfilled circle is an observation node. Each black dot corresponds to a state particle with the size representing its weight. In continuous observation spaces, the beliefs in a POMCP-DPW tree degenerate to a single particle, while POMCPOW maintains weighted particle mixture beliefs.

4.2.2 PFT-DPW

Another algorithm that one might consider for solving continuous POMDPs online is MCTS-DPW on the equivalent belief MDP. Since the Bayesian belief update is usually computationally intractable, a particle filter is used. This new approach will be referred to as particle filter trees with double progressive widening (PFT-DPW). It is shown in Algorithm 2, where $G_{\mathrm{PF}(m)}(b, a)$ is a particle filter belief update performed with a simulated observation and m state particles which approximates the belief MDP generative model. The authors are not aware of any mention of this algorithm in prior literature, but it is very likely that MCTS with particle filters has been used before without double progressive widening under another name.

PFT-DPW is fundamentally different from POMCP and POMCPOW because it relies on simulating approximate belief trajectories instead of state trajectories. This distinction also allows it to be applied to problems where the reward is a function of the belief rather than the state such as pure information-gathering problems [19, 2].

The primary shortcoming of this algorithm is that the number of particles in the filter, m, must be chosen a-priori and is static throughout the tree. Each time a new belief node is created, an $\mathcal{O}(m)$ particle filter update is performed. If m is too small, the beliefs may miss important states, but if m is too large, constructing the tree is expensive. Fortunately, the experiments in Section 4.3 show that it is often easy to choose m in practice; for all the problems studied here, a value of $m = 20$ resulted in good performance.

4.2.3 POMCPOW

In order to address the suboptimality of POMCP-DPW, we now propose a new algorithm, POMCPOW, shown in Algorithm 3. In this algorithm, the belief updates are weighted, but they also expand gradually as more simulations are added. Furthermore, since the richness of the belief representation is related to the number of times the node is visited, beliefs that are more likely to be reached by the optimal policy have more particles. At each step, the simulated state is inserted into the weighted particle collection that represents the belief (line 10), and a new state is sampled from that belief (line 16). A simple illustration of the tree is shown in Figure 4.2 to contrast with a POMCP-DPW tree. Because the resampling in line 16 can be efficiently implemented with binary search, the computational complexity is $\mathcal{O}(nd\log(n))$.

Algorithm 2 PFT-DPW

```
procedure PLAN($b$)
    for $i \in 1:n$ do
        SIMULATE($b, d_{\max}$)
    return $\underset{a}{\operatorname{argmax}}\, Q(ba)$
procedure SIMULATE($b$, $d$)
    if $d = 0$ then
        return 0
    $a \leftarrow$ ACTIONPROGWIDEN($b$)
    if $|C(ba)| \leq k_o N(ba)^{\alpha_o}$ then
        $b', r \leftarrow G_{\mathrm{PF}(m)}(b, a)$
        $C(ba) \leftarrow C(ba) \cup \{(b', r)\}$
        $total \leftarrow r + \gamma$ROLLOUT($b', d-1$)
    else
        $b', r \leftarrow$ sample uniformly from $C(ba)$
        $total \leftarrow r + \gamma$SIMULATE($b', d-1$)
    $N(b) \leftarrow N(b) + 1$
    $N(ba) \leftarrow N(ba) + 1$
    $Q(ba) \leftarrow Q(ba) + \frac{total - Q(ba)}{N(ba)}$
    return $total$
```

Algorithm 3 POMCPOW

```
procedure Simulate($s$, $h$, $d$)
    if $d = 0$ then
        return 0
    $a \leftarrow$ ActionProgWiden($h$)
    $s', o, r \leftarrow G(s, a)$
    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 $\mathcal{Z}(o \mid s, a, s')$ to $W(hao)$
    if $o \notin C(ha)$ then                                        ▷ new node
        $C(ha) \leftarrow C(ha) \cup \{o\}$
        $total \leftarrow r + \gamma$Rollout$(s', hao, d-1)$
    else
        $s' \leftarrow$ select $B(hao)[i]$ w.p. $\frac{W(hao)[i]}{\sum_{j=1}^{m} W(hao)[j]}$
        $r \leftarrow R(s, a, s')$
        $total \leftarrow r + \gamma$Simulate$(s', hao, d-1)$
    $N(h) \leftarrow N(h) + 1$
    $N(ha) \leftarrow N(ha) + 1$
    $Q(ha) \leftarrow Q(ha) + \frac{total - Q(ha)}{N(ha)}$
    return $total$
```

4.2.4 Discretization

Discretization is perhaps the most straightforward way to deal with continuous observation spaces. In this approach, the continuous observation space is simply divided into small discrete regions and solved as a discrete POMDP using conventional methods. The results in Table 4.1 show that this approach is only sometimes effective.

4.2.5 Observation Distribution Requirement

It is important to note that, while POMCP, POMCP-DPW, and DESPOT only require a generative model of the problem, both POMCPOW and PFT-DPW require a way to query the relative likelihood of different observations ($\mathcal{Z}$ in line 11). One may object that this will limit the application of POMCPOW to a small class of POMDPs, but we think it will be an effective tool in practice for two reasons.

First, this requirement is no more stringent than the requirement for a standard importance resampling particle filter, and such filters are used widely, at least in the field of robotics that the authors are most familiar with. Moreover, if the observation model is complex, an approximate model may be sufficient.

Second, given the implications of Theorem 1, it is difficult to imagine a tree-based decision-making algorithm or a robust belief updater that does not require some way of measuring whether a state belongs to a belief or history. The observation model is a straightforward and standard way of specifying such a measure. Finally, in practice, except for the simplest of problems, using POMCP or DESPOT to repeatedly observe and act in an environment already requires more than just a generative model. For example, the authors of the original paper describing POMCP [61] use heuristic particle reinvigoration in lieu of an observation model and importance sampling.

4.3 Experiments

Numerical simulation experiments were conducted to evaluate the performance of POMCPOW and PFT-DPW compared to other solvers. The open source code for the experiments is built on the POMDPs.jl framework (Chapter 5) and is hosted at `https://github.com/zsunberg/ContinuousPOMDPTreeSearchExperiments.jl`. In all experiments, the solvers were limited to 1 second of computation time per step. Belief updates were accomplished with a particle filter independent of the planner, and no part of the tree was saved for re-use on subsequent steps. Hyperparameter values are shown in Section 4.3.7.

4.3.1 Laser Tag

The Laser Tag benchmark (Fig. 4.3) is taken directly from the work of Somani et al. [63] and included for the sake of calibration. A complete description is given in that work. DESPOT outperforms the other methods in this test. The score for DESPOT differs slightly from that reported by Somani et al. [63] likely because of bounds implementation differences. POMCP performs much better than reported by Somani et al. [63] because this implementation uses a state-based rollout policy.

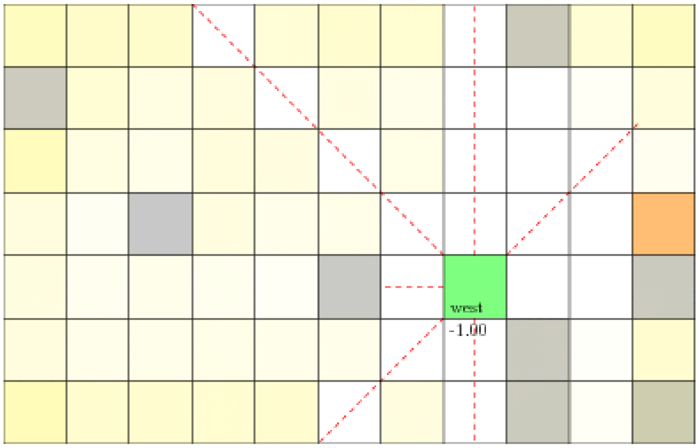


Figure 4.3: Laser Tag benchmark. The green square is the robot; the orange square is the target. The varying shades of yellow show the belief about the position of the target. The red dashed lines represent the laser sensors.

4.3.2 Light Dark

In the Light Dark domain, the state is an integer between -60 and 60, and the agent can choose how to move deterministically ($s' = s + a$) from the action space

Table 4.1: Experimental Results

	Laser Tag (D, D, D)	Light Dark (D, D, C)	Sub Hunt (D, D, C)
POMCPOW	−10.3 ± 0.2 🟩	56.1 ± 0.6 🟩	69.2 ± 1.3 🟩
PFT-DPW	−11.1 ± 0.2 🟩	57.2 ± 0.5 🟩	77.4 ± 1.1 🟩
QMDP	−10.5 ± 0.2 🟩	−6.4 ± 1.0 🟥	28.0 ± 1.3 🟫
POMCP-DPW	−10.6 ± 0.2 🟩	−7.3 ± 1.0 🟥	28.3 ± 1.3 🟫
DESPOT	−8.9 ± 0.2 🟩	−6.8 ± 1.0 🟥	26.8 ± 1.3 🟫
POMCP^D	−14.1 ± 0.2 🟫	61.1 ± 0.4 🟩	28.0 ± 1.3 🟫
DESPOT^D		54.2 ± 1.1 🟩	27.4 ± 1.3 🟫
	VDP Tag (C, C, C)	**Multilane (C, D, C)**	
POMCPOW	29.3 ± 0.8 🟩	30.9 ± 0.9 🟩	
PFT-DPW	27.2 ± 0.8 🟩	21.4 ± 0.9 🟫	
QMDP			
POMCP-DPW	16.4 ± 1.0 🟫	29.6 ± 0.9 🟩	
DESPOT		36.0 ± 0.8 🟩	
POMCP^D	14.7 ± 0.9 🟫		
DESPOT^D	14.3 ± 1.0 🟫		

The three C or D characters after the solver indicate whether the state, action, and observation spaces are continuous or discrete, respectively. For continuous problems, solvers with a superscript D were run on a version of the problem with discretized action and observation spaces, but they interacted with continuous simulations of the problem.

$\mathcal{A} = \{-10, -1, 0, 1, 10\}$. The goal is to reach the origin and take an action there. If action 0 is taken at the origin, a reward of 100 is given and the problem terminates; If action 0 is taken at another location, a penalty of -100 is given. There is a cost of -1 at each step before termination. The agent receives a more accurate observation in the "light" region around $s = 10$. Specifically, observations are continuous ($\mathcal{O} = \mathbb{R}$) and normally distributed with standard deviation $\sigma = |s - 10|$.

Table 4.1 shows the mean reward from 1000 simulations for each solver, and Fig. 4.4 shows an example experiment. The optimal strategy involves moving toward the light region and localizing before proceeding to the origin. QMDP and solvers predicted to behave like QMDP attempt to move directly to the origin, while POMCPOW and PFT-DPW perform better. In this one-dimensional case, discretization allows POMCP to outperform all other methods and DESPOT to perform well, but in subsequent problems where the observation space has more dimensions, discretization does not provide the same performance improvement (see Section 4.2.4).

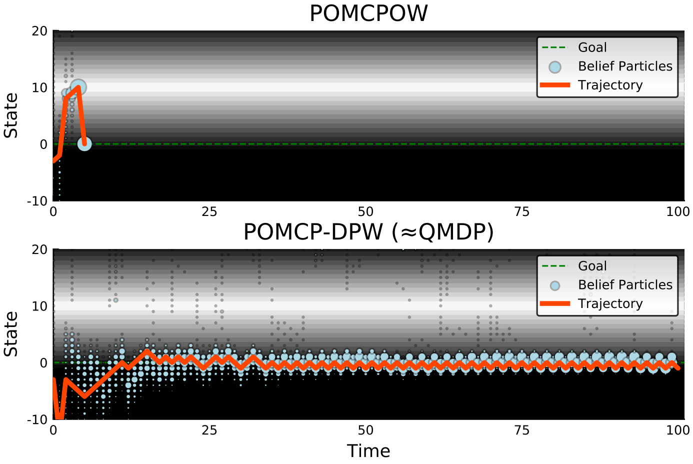


Figure 4.4: Example trajectories in the Light Dark domain. POMCPOW travels to the light region and accurately localizes before moving to the goal. POMCP-DPW displays QMDP-like behavior: it is unable to localize well enough to take action 0 with confidence. The belief particles far away from 0 in the POMCP-DPW plot are due to particle reinvigoration that makes the filter more robust.

4.3.3 Sub Hunt

In the Sub Hunt domain, the agent is a submarine attempting to track and destroy an enemy sub. The state and action spaces are discrete so that QMDP can be used to solve the problem for comparison. The agent and the target each occupy a cell of a 20 by 20 grid. The target is either aware or unaware of the agent and seeks to reach a particular edge of the grid unknown to the agent ($\mathcal{S} = \{1, .., 20\}^4 \times \{\text{aware}, \text{unaware}\} \times \{N, S, E, W\}$). The target stochastically moves either two steps towards the goal or one step forward and one to the side. The agent has six actions, move three steps north, south, east, or west, engage the other submarine, or ping with active sonar. If the agent chooses to engage and the target is unaware and within a range of 2, a hit with reward 100 is scored; The problem ends when a hit is scored or the target reaches its goal edge.

An observation consists of 8 sonar returns ($\mathcal{O} = \mathbb{R}^8$) at equally-spaced angles that give a normally distributed estimate ($\sigma = 0.5$) of the range to the target if the target is within that beam and a measurement with higher variance if it is not. The range of the sensors depends on whether the agent decides to use active sonar. If the agent does not use active sonar it can only detect the other submarine within a radius of 3, but pinging with active sonar will detect at any range. However, active sonar alerts the target to the presence of the agent, and when the target is aware, the hit probability when engaging drops to 60%.

Table 4.1 shows the mean reward for 1000 simulations for each solver. The optimal strategy includes using the active sonar, but previous approaches have difficulty determining this because of the reduced engagement success rate. The PFT-DPW approach has the best score, followed closely by POMCPOW. All other solvers have similar performance to QMDP.

4.3.4 Van Der Pol Tag

The final experimental problem is called Van Der Pol tag and has continuous state, action, and observation spaces. In this problem an agent moves through 2D space to try to tag a target ($\mathcal{S} = \mathbb{R}^4$) that has a random unknown initial position in

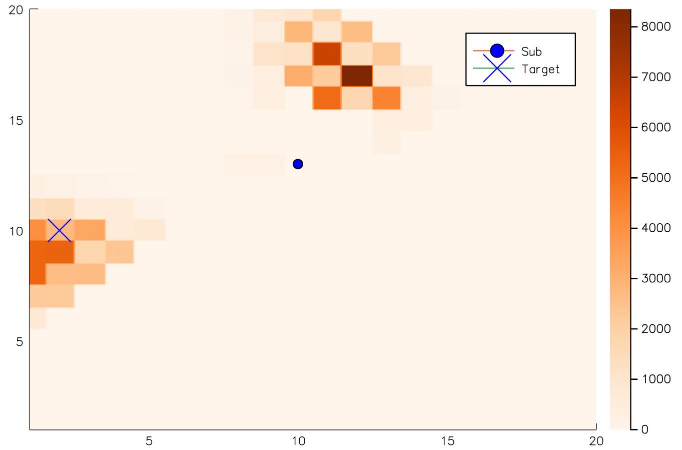


Figure 4.5: Sub Hunt benchmark. The color of each location square shows the number of belief particles containing the target at that location. In this case, the sub has incorrectly concluded that, with high probability, the target is moving towards the east and is chasing that belief concentration.

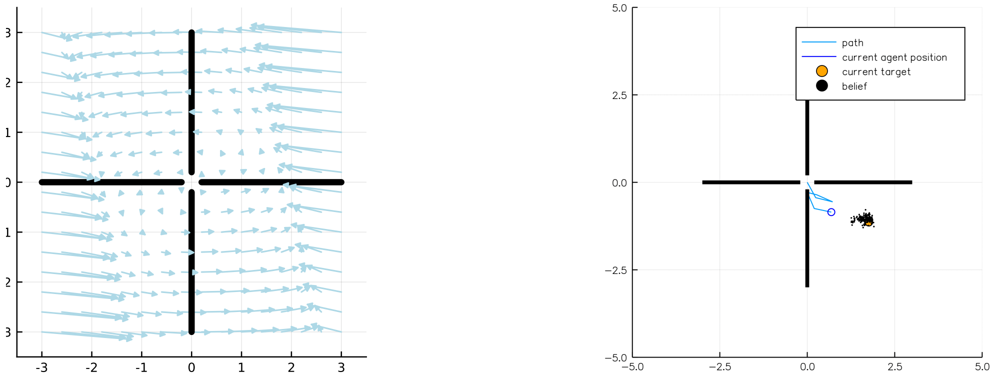
$[-4,4] \times [-4,4]$. The agent always travels at the same speed, but chooses a direction of travel and whether to take an accurate observation ($\mathcal{A} = [0, 2\pi) \times \{0, 1\}$). The observation again consists of 8 beams ($\mathcal{O} = \mathbb{R}^8$) that give measurements to the target. Normally, these measurements are too noisy to be useful ($\sigma = 5$), but, if the agent chooses an accurate measurement with a cost of 5, the observation has low noise ($\sigma = 0.1$). The agent is blocked if it comes into contact with one of the barriers that stretch from 0.2 to 3.0 in each of the cardinal directions (see Fig. 4.6), while the target can move freely through. There is a cost of 1 for each step, and a reward of 100 for tagging the target (being within a distance of 0.1).

The target moves following a two dimensional form of the Van Der Pol oscillation defined by the differential equations

$$\dot{x} = \mu\left(x - \frac{x^3}{3} - y\right) \quad \text{and} \quad \dot{y} = \frac{1}{\mu}x,$$

where $\mu = 2$. Gaussian noise ($\sigma = 0.05$) is added to the position at the end of each step. Runge-Kutta fourth order integration is used to propagate the state.

This problem has several challenging features that might be faced in real-world



(a) Van Der Pol Dynamics. The arrows show the target differential equation, and the thick black lines represent the barriers.

(b) Van Der Pol Tag. The agent attempts to tag the target, but cannot pass through barriers.

Figure 4.6: Van Der Pol tag problem

applications. First, the state transitions are more computationally expensive because of the numerical integration. Second, the continuous state space and obstacles make it difficult to construct a good heuristic rollout policy, so random rollouts are used. Table 4.1 shows the mean reward for 1000 simulations of this problem for each solver. Since a POMCPOW iteration requires less computation than a PFT-DPW iteration, POMCPOW simulates more random rollouts and thus performs slightly better.

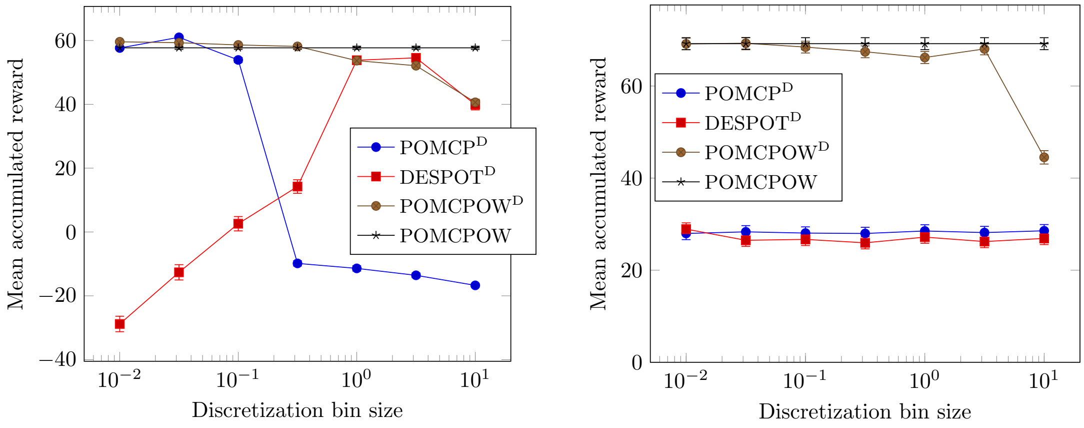
4.3.5 Multilane

The final problem for evaluation is the multiple-lane-change problem described in Section 3.2. Since costly information gathering is not an important part of this problem, POMCP-DPW and POMCPOW have similar performance. Because it uses a fixed number of scenarios and bounds to control exploration rather than progressive widening and a UCB heuristic, DESPOT is able to make better long term plans and is the most effective in this problem. It is also important to note that in this problem, the most realistic of the test problems, PFT-DPW performs significantly worse than POMCPOW.

4.3.6 Discretization granularity

Figure 4.7 shows the performance at different discretization granularities for the Light Dark and Sub Hunt problems.

Since the Light Dark domain has only a single observation dimension, it is easy to discretize. In fact, POMCP with fine discretization outperforms POMCPOW. However, discretization is only effective at certain granularities, and this is highly dependent on the solver and possibly hyperparameters. In the Sub Hunt problem, with its high-dimensional observation, discretization is not effective at any granularity. In Van Der Pol tag, both the action and observation spaces must be discretized. Due to the high dimensionality of the observation space, similar to Sub Hunt, no discretization that resulted in good performance was found.



(a) Light Dark

(b) Sub Hunt

Figure 4.7: Discretization granularity studies

4.3.7 Hyperparameters

Hyperparameters for POMCPOW and PFT-DPW were chosen using the cross entropy method [45], but exact tuning was not a high priority and some parameters were re-used across solvers so the parameters may not be perfectly optimized. The values used in the experiments are shown in Table 4.2. There are not enough experiments to draw broad conclusions about the hyperparameters, but it appears that

Table 4.2: Hyperparameters used in experiments

POMCPOW	Laser Tag	Light Dark	Sub Hunt	VDP Tag	Multilane
c	26.0	90.0	17.0	110.0	8.0
k_a	–	–	–	30.0	–
α_a	–	–	–	1/30	–
k_o	4.0	5.0	6.0	5.0	4.5
α_o	1/35	1/15	1/100	1/100	1/10
PFT-DPW	Laser Tag	Light Dark	Sub Hunt	VDP Tag	Multilane
m	20	20	20	20	15
c	26.0	100.0	100.0	70.0	8.0
k_a	–	–	–	20.0	–
α_a	–	–	–	1/25	–
k_o	4.0	4.0	2.0	8.0	4.5
α_o	1/35	1/10	1/10	1/85	1/10

For problems with discrete actions, all actions are considered and k_a and α_a are not needed.

performance is most sensitive to the exploration constant, c.

The values for the observation widening parameters, k_o and α_o, were similar for all the problems in this work. A small α_o essentially limits the number of observations to a static number k_o, resulting in behavior reminiscent of sparse UCT [14], preventing unnecessary widening and allowing the tree to grow deep. This seems to work well in practice with the branching factor (k_o) set to values between 2 and 8, and suggests that it may be sufficient to limit the number of children to a fixed number rather than do progressive widening in a real implementation.

4.4 Summary

The results in this chapter clearly show that the previous leading online solvers are unable to adequately solve POMDPs with continuous observation spaces where exploration is important. In particular, Theorem 1 explains why these methods behave similarly to QMDP. POMCPOW and PFT-DPW overcome this limitation and outperform QMDP. Though POMCPOW only performs strictly best on one of the problems in Section 4.3, it performs very well on all of them, so it might be considered the best overall algorithm in these tests.

However there are still several shortcomings. First, I have not proven analytically that POMCPOW or PFT-DPW converges toward the optimal solution. Second, double progressive widening is made more difficult in practice because of its tuning parameters. Third, the results from the Multilane experiment shows that DESPOT is able to find significantly better plans in more realistic problems. Thus, much work remains along this trajectory. DESPOT has several attractive attributes compared to the new algorithms presented here. In particular, its use of a fixed number of scenarios to control widening, and bounds to control exploration may make it more amenable to analysis and allow it to perform better in practice. Extending DESPOT to handle continuous observation spaces seems to be a promising future direction.

Chapter 5

POMDPs.jl: A Framework for Sequential Decision Making under Uncertainty

This chapter describes the POMDPs.jl software package created by the Stanford Intelligent Systems Lab (SISL) to make state-of-the-art POMDP solution methods easily accessible to students, researchers, and engineers. All of the research in Chapters 3 and 4 was conducted using this framework.

Use cases?

5.1 Challenges for POMDP-solving software

A successful POMDP software framework must have, at a minimum, the following attributes: speed, flexibility, and ease of use.

5.1.1 Speed

Since POMDPs are difficult to solve, any computational slowdown such as unnecessary memory allocation or runtime type inference significantly reduces the maximum problem size that the framework can handle. For this reason, POMDP algorithms

Make stronger statement about computational must be compiled to efficient processor instructions with low overhead.

5.1.2 Flexibility

The set of problems that can be represented as a POMDP is extremely large and there are many possible characteristics that such problems might have. A good POMDP software framework should try to accommodate as much of this set as possible. A few of the most important model characteristics to support are outlined below.

Partial and full observability

When studying a POMDP problem, it is almost always important to analyze the underlying fully-observable problem. Thus, a good POMDP framework should have first-class support for MDPs in addition to POMDPs.

Continuous and discrete problems

Some POMDPs have a finite number of states, actions, and observations, i.e. $|\mathcal{S}| < \infty$, $|\mathcal{A}| < \infty$, and $|\mathcal{O}| < \infty$. However, many real world problems, notably robotics problems, are naturally formulated in spaces with uncountably infinite cardinality, e.g. $\mathcal{S} = \mathbb{R}$, $\mathcal{A} = \mathbb{R}$, and $\mathcal{O} = \mathbb{R}$, or multi-dimensional vector spaces, e.g. $\mathcal{S} = \mathbb{R}^6$ This means that the framework cannot be constrained to use integers for state representation, but should be capable of using floating point vectors.

Explicit vs generative model representation

The explicit

Online and offline solvers

While some POMDP solution techniques seek exact offline solutions to small problems, many larger problems can only be practically solved online. Thus a good

POMDP framework must have first class support for both solving offline and efficiently executing a policy online or executing a planner that does significant computation online.

Policy representation

The policies that

Exact and approximate solutions

5.1.3 Ease of Use

5.2 Previous frameworks

5.3 Architecture

5.3.1 Concepts

5.3.2 Interfaces

5.4 Examples

Appendix A

Proof of Theorem 1

A version of Monte Carlo tree search with double progressive widening has been proven to converge to the optimal value function on fully observable MDPs by Auger, Couetoux, and Teytaud [3]. We utilize this proof to show that POMCP-DPW converges to a solution that is sometimes suboptimal.

First we establish some preliminary definitions taken directly from Auger, Couetoux, and Teytaud [3].

Definition 2 (Regularity Hypothesis)**.** *The* Regularity hypothesis *is the assumption that for any $\Delta > 0$, there is a non zero probability to sample an action that is optimal with precision Δ. More precisely, there is a $\theta > 0$ and a $p > 1$ (which remain the same during the whole simulation) such that for all $\Delta > 0$,*

$$Q(ha) \geq Q^*(h) - \Delta \textit{ with probability at least } \min(1, \theta\Delta^p). \tag{A.1}$$

Definition 3 (Exponentially sure in n)**.** *We say that some property depending on an integer n is exponentially sure in n if there exists positive constants C, h, and η such that the probability that the property holds is at least*

$$1 - C\exp(-hn^\eta).$$

In order for the proof from Auger, Couetoux, and Teytaud [3] to apply, the following four minor modifications to the POMCP-DPW algorithm must be made:

1. Instead of the usual logarithmic exploration, use *polynomial exploration*, that is, select actions based on the criterion

$$Q(ha) + \sqrt{\frac{N(h)^{e_d}}{N(ha)}} \tag{A.2}$$

   as opposed to the traditional criterion

$$Q(ha) + c\sqrt{\frac{\log N(h)}{N(ha)}}, \tag{A.3}$$

   and create a new node for progressive widening when $\lfloor N^{\alpha} \rfloor > \lfloor (N-1)^{\alpha} \rfloor$ rather than when the number of children exceeds kN^{α}.

2. Instead of performing rollout simulations, keep creating new single-child nodes until the maximum depth is reached.

3. In line 15, instead of selecting an observation randomly, select the observation that has been visited least proportionally to how many times it has been visited.

4. Use the depth-dependent coefficient values in Table 1 from Auger, Couetoux, and Teytaud [3] instead of choosing static values.

This version of the algorithm will be referred to as “modified POMCP-DPW”. The algorithm with these changes is listed in Algorithm 4.

We now define the “QMDP value” that POMCP-DPW converges to (this is repeated from the main text of the paper) and prove a preliminary lemma.

Definition 4 (QMDP value)**.** *Let $Q_{MDP}(s, a)$ be the optimal state-action value function assuming full observability starting by taking action a in state s. The* QMDP value *at belief b, $Q_{MDP}(b, a)$, is the expected value of $Q_{MDP}(s, a)$ when s is distributed according to b.*

Lemma 1. *If POMCP-DPW or modified POMCP-DPW is applied to a POMDP with a continuous observation space and observation probability density functions that are finite everywhere, then each history node in the tree will have only one corresponding state, that is $|B(h)| = 1, M(h) = 1\,\forall h$.*

Proof. Since the observation probability density function is finite, each call to the generative model will produce a unique observation with probability 1. Because of this, lines 18 and 19 of Algorithm 4 will only be executed once for each observation. □

We are now ready to restate and prove the theorem from the text.

Theorem 1 (Modified POMCP-DPW convergence to QMDP)**.** *If a bounded-horizon POMDP meets the following conditions: 1) the state and observation spaces are continuous with a finite observation probability density function, and 2) the regularity hypothesis is met, then modified POMCP-DPW will produce a value function estimate, $\hat{Q}$, that converges to the QMDP value for the problem. Specifically, there exists a constant $C > 0$, such that after n iterations,*

$$\left|\hat{Q}(b, a) - Q_{MDP}(b, a)\right| \leq \frac{C}{n^{1/(10d_{\max}-7)}}$$

exponentially surely in n, for every action a.

Proof. We prove that modified POMCP-DPW functions exactly as the Polynomial UCT (PUCT) algorithm defined by Auger, Couetoux, and Teytaud [3] applied to an augmented fully observable MDP, and hence converges to the QMDP value. We will show this by proposing incremental changes to Algorithm 4 that do not change its function that will result in an algorithm identical to PUCT.

Before listing the changes, we define the “augmented fully observable MDP” as follows: For a POMDP $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, \mathcal{Z}, \gamma)$, and belief b, the *augmented fully observable MDP*, $\mathcal{M}$, is the MDP defined by $(\mathcal{S}_A, \mathcal{A}, \mathcal{T}_A, \mathcal{R}, \gamma)$, where

$$\mathcal{S}_A = \mathcal{S} \cup \{b\} \tag{A.4}$$

and, for all $x, x' \in \mathcal{S}_A$,

$$\mathcal{T}_A(x'|x,a) = \begin{cases} \mathcal{T}(x'|x,a) & \text{if } x \in \mathcal{S} \\ \int_S b(s)\mathcal{T}(x'|s,a)ds & \text{if } x = b \end{cases} \tag{A.5}$$

This is simply the fully observable MDP augmented with a special state representing the current belief. It is clear that the value function for this problem $Q_{\mathcal{M}}(b,a)$ is the same as the QMDP value for the POMDP, $Q_{\text{MDP}}(b,a)$. Thus, by showing that modified POMCP-DPW behaves exactly as PUCT applied to $\mathcal{M}$, we show that it estimates the QMDP values.

Consider the following modifications to Algorithm 4 that do not change its behavior when the observation space is continuous:

1. Eliminate the state count M. *Justification*: By Lemma 1, its value will be 1 for every node.

2. Remove B and replace with a mapping H from each node to a state of $\mathcal{M}$; define $H(b) = b$. *Justification*: By Lemma 1 B always contains only a single state, so H contains the same information.

3. Generate states and rewards with $G_{\mathcal{M}}$, the generative model of $\mathcal{M}$, instead of G. *Justification*: Since the state transition model for the fully observable MDP is the same as the POMDP, these are equivalent for all $s \in \mathcal{S}$.

4. Remove the s argument of SIMULATE. *Justification*: The sampling in line 3 is done implicitly in $G_{\mathcal{M}}$ if $h = b$, and s is redundant in other cases because h can be mapped to s through H.

The result of these changes is shown in Algorithm 5. It is straightforward to verify that this algorithm is equivalent to PUCT applied to $\mathcal{M}$. Each observation-terminated history, h, corresponds to a PUCT "decision node", z, and each action-terminated history, ha, corresponds to a PUCT "chance node", w. In other words, the observations have no meaning in the tree other than making up the histories, which are effectively just keys or aliases for the state nodes.

Since PUCT is guaranteed by Theorem 1 of Auger, Couetoux, and Teytaud [3] to converge to the optimal value function of $\mathcal{M}$ exponentially surely, POMCP-DPW is guaranteed to converge to the QMDP value exponentially surely, and the theorem is proven.

$\square$

Remark 1. *One may object that multiple histories may map to the same state through H, and thus the history nodes in a modified POMCP-DPW tree are not equivalent to state nodes in the PUCT tree. In fact, the PUCT algorithm does not check to see if a state has previously been generated by the model, so it may also contain multiple decision nodes z that correspond to the same state. Though this is not explicitly stated by the authors, it is clear from the algorithm description, and the proof still holds.*

Algorithm 4 Modified POMCP-DPW

```
procedure Plan($b$)
    for $i \in 1:n$ do
        $s \leftarrow$ sample from $b$
        Simulate($s, b, d_{\max}$)
    return $\underset{a}{\operatorname{argmax}}\, Q(ba)$
procedure ActionProgWiden($h$)
    if $\lfloor N(h)^{\alpha_{a,d}} \rfloor > \lfloor (N(h)-1)^{\alpha_{a,d}} \rfloor$ then
        $a \leftarrow$ NextAction($h$)
        $C(h) \leftarrow C(h) \cup \{a\}$
    return $\underset{a \in C(h)}{\operatorname{argmax}}\, Q(ha) + \sqrt{\frac{N(h)^{e_d}}{N(ha)}}$
procedure Simulate($s$, $h$, $d$)
    if $d = 0$ then
        return 0
    $a \leftarrow$ ActionProgWiden($h$)
    if $\lfloor N(ha)^{\alpha_{o,d}} \rfloor > \lfloor (N(ha)-1)^{\alpha_{o,d}} \rfloor$ then
        $s', o, r \leftarrow G(s, a)$
        $C(ha) \leftarrow C(ha) \cup \{o\}$
        $M(hao) \leftarrow M(hao) + 1$
        append $s'$ to $B(hao)$
    else
        $o \leftarrow \underset{o \in C(ha)}{\operatorname{argmin}}\, N(hao)/M(hao)$
        $s' \leftarrow$ select $s' \in B(hao)$ w.p. $\frac{1}{|B(hao)|}$
        $r \leftarrow R(s, a, s')$
    $total \leftarrow r + \gamma$Simulate($s', hao, d-1$)
    $N(h) \leftarrow N(h) + 1$
    $N(ha) \leftarrow N(ha) + 1$
    $Q(ha) \leftarrow Q(ha) + \frac{total - Q(ha)}{N(ha)}$
    return $total$
```

Algorithm 5 Modified POMCP-DPW on a continuous observation space

```
procedure PLAN($b$)
    for $i \in 1 : n$ do
        SIMULATE($(b), d_{\max}$)
    return $\underset{a}{\operatorname{argmax}}\, Q(ha)$
procedure ACTIONPROGWIDEN($h$)
    if $\lfloor N(h)^{\alpha_{a,d}} \rfloor > \lfloor (N(h)-1)^{\alpha_{a,d}} \rfloor$ then
        $a \leftarrow$ NEXTACTION($h$)
        $C(h) \leftarrow C(h) \cup \{a\}$
    return $\underset{a \in C(h)}{\operatorname{argmax}}\, Q(ha) + \sqrt{\frac{N(h)^{e_d}}{N(ha)}}$
procedure SIMULATE($h$, $d$)
    if $d = 0$ then
        return 0
    $a \leftarrow$ ACTIONPROGWIDEN($h, d$)
    if $\lfloor N(ha)^{\alpha_{o,d}} \rfloor > \lfloor (N(ha)-1)^{\alpha_{o,d}} \rfloor$ then
        $\cdot, o, \cdot \leftarrow G(H(h), a)$
        $H(hao), r \leftarrow G_{\mathcal{M}}(H(h), a)$
        $C(ha) \leftarrow C(ha) \cup \{o\}$
    else
        $o \leftarrow \underset{o \in C(ha)}{\operatorname{argmin}}\, N(hao)$
        $r \leftarrow R(H(h), a, H(hao))$
    $total \leftarrow r + \gamma$SIMULATE($hao, d-1$)
    $N(h) \leftarrow N(h) + 1$
    $N(ha) \leftarrow N(ha) + 1$
    $Q(ha) \leftarrow Q(ha) + \frac{total - Q(ha)}{N(ha)}$
    return $total$
```

Bibliography

[1] Ali-Akbar Agha-Mohammadi, Suman Chakravorty, and Nancy M Amato. "FIRM: Feedback controller-based information-state roadmap - a framework for motion planning under uncertainty". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2011.

[2] Mauricio Araya et al. "A POMDP Extension with Belief-dependent Rewards". In: *Advances in Neural Information Processing Systems (NIPS)*. 2010. URL: `http://papers.nips.cc/paper/3971-a-pomdp-extension-with-belief-dependent-rewards.pdf`.

[3] David Auger, Adrien Couetoux, and Olivier Teytaud. "Continuous upper confidence trees with polynomial exploration–consistency". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2013, pp. 194–209.

[4] Ralph Bach, Chris Farrell, and Heinz Erzberger. *An Algorithm for Level-Aircraft Conflict Resolution*. Tech. rep. CR-2009-214573. NASA, 2009.

[5] Haoyu Bai, David Hsu, and Wee Sun Lee. "Integrated perception and planning in the continuous space: A POMDP approach". In: *International Journal of Robotics Research* 33.9 (2014), pp. 1288–1302.

[6] Haoyu Bai et al. "Intention-Aware Online POMDP Planning for Autonomous Driving in a Crowd". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 454–460. ISBN: 9781479969234. DOI: `10.1109/ICRA.2015.7139219`.

[7] Haoyu Bai et al. "Unmanned Aircraft Collision Avoidance using Continuous-State POMDPs". In: *Robotics: Science and Systems*. 2012.

[8] D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2005.

[9] Justin A. Boyan and Andrew W. Moore. "Generalization in Reinforcement Learning: Safely Approximating the Value Function". In: *Advances in Neural Information Processing Systems*. 1995, pp. 369–376.

[10] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[11] Sebastian Brechtel, Tobias Gindele, and Rudiger Dillmann. "Solving Continuous POMDPs: Value Iteration with Incremental Learning of an Efficient Space Representation". In: *International Conference on Machine Learning (ICML)*. 2013.

[12] Sebastian Brechtel, Tobias Gindele, and Rüdiger Dillmann. "Solving Continuous POMDPs: Value Iteration with Incremental Learning of an Efficient Space Representation". In: *International Conference on Machine Learning (ICML)*. 2013, pp. 370–378.

[13] C. B. Browne et al. "A Survey of Monte Carlo Tree Search Methods". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), pp. 1–43. DOI: 10.1109/TCIAIG.2012.2186810.

[14] Cameron B. Browne et al. "A survey of Monte Carlo tree search methods". In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1 (2012), pp. 1–43.

[15] Adam Bry and Nicholas Roy. "Rapidly-exploring random belief trees for motion planning under uncertainty". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2011, pp. 723–730.

[16] Adrien Couëtoux et al. "Continuous upper confidence trees". In: *Learning and Intelligent Optimization*. Springer, 2011, pp. 433–445.

[17] A. Couëtoux et al. “Continuous Upper Confidence Trees”. In: *Learning and Intelligent Optimization*. Rome, Italy, 2011.

[18] Scott Davies. “Multidimensional Triangulation and Interpolation for Reinforcement Learning”. In: *Advances in Neural Information Processing Systems*. 1996, pp. 1005–1011.

[19] Louis Dressel and Mykel Kochenderfer. “Efficient Decision-Theoretic Target Localization”. In: *International Conference on Automated Planning and Scheduling (ICAPS)*. 2017. URL: `https://www.aaai.org/ocs/index.php/ICAPS/ICAPS17/paper/view/15761/15090`.

[20] Katherine Driggs-Campbell and Ruzena Bajcsy. “Identifying Modes of Intent from Driver Behaviors in Dynamic Environments”. In: *IEEE International Conference on Intelligent Transportation Systems (ITSC)*. 2015.

[21] J. Foster and F. B. Richards. “The Gibbs Phenomenon for Piecewise-Linear Approximation”. In: *The American Mathematical Monthly* 98.1 (1991), pp. 47–49.

[22] Carlos E. Garcia, David M. Prett, and Manfred Morari. “Model Predictive Control: Theory and Practice – a Survey”. In: *Automatica* 25.3 (1989), pp. 335–348.

[23] T. Gindele, S. Brechtel, and R. Dillmann. “Learning Driver Behavior Models from Traffic Observations for Decision Making and Planning”. In: *IEEE Intelligent Transportation Systems Magazine* 7.1 (2015), pp. 69–79. DOI: `10.1109/MITS.2014.2357038`.

[24] Alex Goldhoorn et al. “Continuous real time POMCP to find-and-follow people by a humanoid service robot”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2014.

[25] George Hagen, Ricky Butler, and Jeffrey Maddalon. “Stratway: A Modular Approach to Strategic Conflict Resolution”. In: *AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*. 2011.

[26] Heber Herencia-Zapana, Jean-Baptiste Jeannin, and César Muñoz. "Formal Verification of Safety Buffers for Sate-based Conflict Detection and Resolution". In: *International Congress of the Aeronautical Sciences.* 2010.

[27] Jesse Hoey and Pascal Poupart. "Solving POMDPs with continuous or large discrete observation spaces". In: *International Joint Conference on Artificial Intelligence (IJCAI).* 2005, pp. 1332–1338.

[28] Jessica E. Holland, Mykel J. Kochenderfer, and Wesley A. Olson. "Optimizing the Next Generation Collision Avoidance System for Safe, Suitable, and Acceptable Operational Performance". In: *Air Traffic Control Quarterly* 21.3 (2013), pp. 275–297.

[29] Vadim Indelman, Luca Carlone, and Frank Dellaert. "Planning in the continuous domain: A generalized belief space approach for autonomous navigation in unknown environments". In: *International Journal of Robotics Research* 34.7 (2015), pp. 849–882.

[30] L. P. Kaelbling, M. L. Littman, and A.R. Cassandra. "Planning and acting in partially observable stochastic domains". In: *Artificial Intelligence* 101 (1998), pp. 99–134.

[31] Leslie Pack Kaelbling and Tomás Lozano-Pérez. *Finding Aircraft Collision-Avoidance Strategies using Policy Search Methods.* Tech. rep. TR-2009-043. MIT-CSAIL, 2009.

[32] Arne Kesting, Martin Treiber, and Dirk Helbing. "Agents for Traffic Simulation". In: *Multi Agent Systems: Simulation and Applications.* Ed. by Adelinde M. Uhrmacher and Danny Weyns. CRC Press, 2009. Chap. 11, pp. 325–356.

[33] Arne Kesting, Martin Treiber, and Dirk Helbing. "General Lane-changing Model MOBIL for Car-following Models". In: *Transportation Research Record* 1999 (2007), pp. 86–94.

[34] Mykel J. Kochenderfer. *Decision Making Under Uncertainty: Theory and Application.* MIT Press, 2015.

[35] Mykel J. Kochenderfer and James P. Chryssanthacopoulos. *Robust Airborne Collision Avoidance through Dynamic Programming*. Tech. rep. ATC-371. MIT Lincoln Laboratory, 2011.

[36] Mykel J. Kochenderfer, James P. Chryssanthacopoulos, and Peter P. Radecki. "Robustness of Optimized Collision Avoidance Logic to Modeling Errors". In: *Digital Avionics Systems Conference*. 2010. ISBN: 9781424466160. DOI: `10.1109/DASC.2010.5655381`.

[37] Mykel J. Kochenderfer et al. "Airspace Encounter Models for Estimating Collision Risk". In: *AIAA Journal of Guidance, Control, and Dynamics* 33.2 (2010), pp. 487–499.

[38] Mykel J. Kochenderfer et al. *Model-Based Optimization of Airborne Collision Avoidance Logic*. Tech. rep. ATC-360. MIT Lincoln Laboratory, 2010.

[39] James K. Kuchar and Lee C. Yang. "A review of conflict detection and resolution modeling methods". In: *IEEE Transactions on Intelligent Transportation Systems* 1.4 (2000), pp. 179–189.

[40] Hanna Kurniawati, David Hsu, and Wee Sun Lee. "SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces." In: *Robotics: Science and Systems*. Zurich, Switzerland., 2008.

[41] Hanna Kurniawati and Vinay Yadav. "An online POMDP solver for uncertainty planning in dynamic environment". In: *Robotics Research*. Springer, 2016, pp. 611–629.

[42] Chi-Pang Lam et al. "Improving Human-In-The-Loop Decision Making In Multi-Mode Driver Assistance Systems Using Hidden Mode Stochastic Hybrid Systems". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015.

[43] Sergey Levine and Vladlen Koltun. "Continuous Inverse Optimal Control with Locally Optimal Examples". In: *International Conference on Machine Learning (ICML)*. 2012, pp. 41–48.

[44] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. "Learning policies for partially observable environments: Scaling up". In: *International Conference on Machine Learning (ICML)*. 1995.

[45] Shie Mannor, Reuven Rubinstein, and Yohai Gat. "The Cross Entropy Method for Fast Policy Search". In: *International Conference on Machine Learning (ICML)*. 2003, pp. 512–519.

[46] Nik A Melchior and Reid Simmons. "Particle RRT for path planning with uncertainty". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2007.

[47] Philippe Morere, Roman Marchant, and Fabio Ramos. "Bayesian Optimisation for solving Continuous State-Action-Observation POMDPs". In: *Advances in Neural Information Processing Systems (NIPS)*. 2016.

[48] Anthony Narkawicz, César Muñoz, and Gilles Dowek. "Provably Correct Conflict Prevention Bands Algorithms". In: *Science of Computer Programming* 77.10–11 (2012), pp. 1039–1057. ISSN: 01676423. DOI: `10.1016/j.scico.2011.07.002`.

[49] Keith Naughton. *Humans Are Slamming Into Driverless Cars and Exposing a Key Flaw*. 2015. URL: `http://bloom.bg/1Qw8fjB`.

[50] Hao Yi Ong and Mykel J. Kochenderfer. "Short-Term Conflict Resolution for Unmanned Aircraft Traffic Management". In: *Digital Avionics Systems Conference*. 2015.

[51] Andreas Pas. "Simulation Based Planning for Partially Observable Markov Decision Processes with Continuous Observation Spaces". MA thesis. Maastricht University, 2012.

[52] Robert Platt Jr. et al. "Belief space planning assuming maximum likelihood observations". In: *Robotics: Science and Systems*. 2010.

[53] Samuel Prentice and Nicholas Roy. "The belief roadmap: Efficient planning in belief space by factoring the covariance". In: *International Journal of Robotics Research* 28.11-12 (2009), pp. 1448–1465.

[54] Erick J. Rodrǵuez-Seda. "Decentralized Trajectory Tracking with Collision Avoidance Control for Teams of Unmanned Vehicles with Constant Speed". In: *American Control Conference*. 2014.

[55] Stéphane Ross et al. "Online planning algorithms for POMDPs". In: *Journal of Artificial Intelligence Research* 32 (2008), pp. 663–704.

[56] Dorsa Sadigh et al. "Information Gathering Actions over Human Internal State". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016.

[57] Dorsa Sadigh et al. "Planning for Autonomous Cars that Leverage Effects on Human Actions". In: *Robotics: Science and Systems*. 2016.

[58] D. Sadigh et al. "Data-Driven Probabilistic Modeling and Verification of Human Driver Behavior". In: *AAAI Spring Symposium on Formal Verification and Modeling in Human-Machine Systems*. 2014.

[59] Brandon Schoettle and Michael Sivak. *A Preliminary Analysis of Real-World Crashes Involving Self-Driving Vehicles*. Tech. rep. UMTRI-2015-34. University of Michigan Transportation Research Institute, 2015.

[60] Konstantin M. Seiler, Hanna Kurniawati, and Surya P. N. Singh. "An online and approximate solver for POMDPs with continuous action space". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 2290–2297.

[61] David Silver and Joel Veness. "Monte-Carlo Planning in Large POMDPs". In: *Advances in Neural Information Processing Systems (NIPS)*. 2010.

[62] *Small Unmanned Aircraft Systems*. 14 C.F.R. §107. 2018.

[63] Adhiraj Somani et al. "DESPOT: Online POMDP planning with regularization". In: *Advances in Neural Information Processing Systems (NIPS)*. 2013, pp. 1772–1780.

[64] Zachary N. Sunberg, Christopher J. Ho, and J. Kochenderfer Mykel. "The Value of Inferring the Internal State of Traffic Participants for Autonomous Freeway Driving". In: *American Control Conference (ACC)*. 2017.

[65] S. Temizer et al. “Collision Avoidance for Unmanned Aircraft using Markov Decision Processes”. In: *AIAA Guidance, Navigation, and Control Conference.* 2010.

[66] Sebastian Thrun. “Monte Carlo POMDPs.” In: *Advances in Neural Information Processing Systems (NIPS).* Vol. 12. 1999, pp. 1064–1070.

[67] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics.* MIT Press, 2005.

[68] C. Tomlin, G. J. Pappas, and S. S. Sastry. “Conflict resolution for air traffic management: A study in multiagent hybrid systems”. In: *IEEE Transactions on Automatic Control* 43.4 (1998), pp. 509–21.

[69] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. “Congested traffic states in empirical observations and microscopic simulations”. In: *Physical Review E* 62.2 (2 2000), pp. 1805–1824.

[70] Jur Van Den Berg, Sachin Patil, and Ron Alterovitz. “Motion planning under uncertainty using iterative local optimization in belief space”. In: *International Journal of Robotics Research* 31.11 (2012), pp. 1263–1278.

[71] Tim A. Wheeler, Philipp Robbel, and Mykel J. Kochenderfer. “A Probabilistic Framework for Microscopic Traffic Propagation”. In: *IEEE International Conference on Intelligent Transportation Systems (ITSC).* Las Palmas de Gran Canaria, Spain, 2015. DOI: `10.1109/ITSC.2015.52`.