SAFE AND SCALABLE PLANNING UNDER UNCERTAINTY FOR AUTONOMOUS DRIVING

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

Maxime Bouton
December 2019

Maxime Bouton

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 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.

(Dorsa Sadigh)

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)

Approved for the Stanford University Committee on Graduate Studies

Abstract

Autonomous driving has the potential to significantly improve safety on the roads. Although major progress has been made in recent years to deploy automated driving technologies, many situations handled on a daily basis by human drivers remain challenging for autonomous vehicles, such as navigating urban environments. They must reach their goal safely and efficiently while considering a multitude of traffic participants with rapidly changing behavior. Hand-engineering strategies to navigate such environments requires anticipating all possible situations and finding a suitable behavior for each, which places a large burden on the designer and is unlikely to scale to complicated situations. In addition, autonomous vehicles rely on on-board perception systems that give noisy estimates of the location and velocity of others on the road and are sensitive to occlusions. Autonomously navigating urban environments requires algorithms that reason about interactions with and between traffic participants with limited information.

This thesis addresses the problem of automatically generating decision making strategies for autonomous vehicles in urban environments. Previous approaches relied on planning with respect to a mathematical model of the environment but have many limitations. A partially observable Markov decision processes (POMDPs) is a standard model for sequential decision making problems in dynamic, uncertain scenarios with imperfect sensor measurements. This thesis demonstrates a generic representation of driving scenarios as POMDPs, considering sensor occlusions and interactions between road users.

A key contribution of this thesis is a methodology to scale POMDP approaches to complex environments involving a large number of traffic participants. To reduce the computational cost of considering multiple traffic participants, a decomposition method leverages the strategies of interacting with a subset of road users. Decomposition methods can approximate the solutions to large sequential decision making problems at the expense of sacrificing optimality. This thesis introduces a new algorithm using deep reinforcement learning to bridge the gap with the optimal solution.

Establishing trust in the generated decision strategies is also necessary to the deployment of autonomous vehicles. Methods to constrain a policy trained using reinforcement learning are introduced and combined with the proposed decomposition techniques. By relying on simple models, these techniques allows learn policies with safety constraints. To address model uncertainty, a new methodology for computing probabilistic safety guarantees in partially observable domains is introduced. It is shown that the new method is more flexible and more scalable than previous work.

The algorithmic contributions present in this thesis are applied to a variety of driving scenarios. Each algorithm is evaluated in simulation and compared to previous work. It is shown that the POMDP formulation in combination with powerful solving method provide a flexible framework for planning under uncertainty for autonomous driving.

Acknowledgments

This work would not have been possible without the support of ... I would like to thank ...

Contents

List of Tables

List of Figures

List of Algorithms

1 Introduction

In recent years, major strides have been made in enabling autonomous driving technologies. However, scenarios such as urban intersections or merging in dense traffic remain challenging for autonomous vehicles. This thesis presents methods for generating safe, scalable, and efficient decision strategies for autonomously navigating complex driving scenarios. In this chapter, we first describe the main technologies and challenges behind autonomous driving before discussing the problem of tactical decision making. The remaining sections present the contributions and an overview of the thesis.

1.1 Autonomous Driving

Introducing automated vehicle technology on the road has the potential to greatly improve safety. Road fatilities represent about 1.35 millions of death each year over the world and is the leading cause of death for children and young adults [1]. Moreover, it is estimated that 94 % of serious crashes are caused by human error [2]. By removing the human from the loop, autonomous driving aims to drastically reduce road fatalities. In addition to the safety impact, it is expected that autonomy would improve efficiency and provide better access to mobility.

Autonomous vehicles are often classified into different levels of autonomy ranging from zero to five [3]. A vehicle with no autonomy is categorized as level 0, whereras a vehicle with complete autonomy that does not need human supervision in any environments is categorized as level 5. In this thesis, we consider automated vehicles of level 3 or more. At this level, the system is able to handle all aspects of the driving task within a specific scenario such as an unprotected left turn at an intersection.

Throughout this thesis, we refer to the automated vehicle taking decisions as *the ego vehicle*. Other traffic participants are assumed to be pedestrians or human driven vehicles.

The *autonomy stack* (architecture) of an autonomous vehicle is often divided into perception, planning, and control [4] as illustrated in Figure 1.1.

Perception The perception module is responsible for sensing and mapping the environment. Sensors often involve a combination of cameras, radars, and lidars. The sensor outputs are processed using sensor fusion and state estimation techniques to give a representation of the environment that can be interpreted by a planning algorithm. The resulting representation contains the road topology as well as estimates of the shape, position, and speed of other traffic participants. This information is often noisy, with false positives and false negatives. Moreover, sensing devices are hindered by occlusions resulting from physical obstacles like buildings or vehicles.

Planning The planning module processes the information given by the perception system as well as the user specifications (e.g. destination) to generate a desired trajectory. A trajectory consists of waypoints as well as speed and acceleration profiles over time.

Control The control module is responsible for commanding the actuators of the vehicle. It takes into account the dynamics and the physical state of the autonomous vehicle to follow the trajectory given by the planning module as closely as possible.

Lead by Original Equipment Manufacturers (OEMs, common term to refer to the major automotive companies) and technology companies, major progress have been made in enabling the deployment of autonomous driving. However, their application is still limited in scope due to several challenges, ranging from engineering the autonomy stack to designing regulations. As self-driving cars are

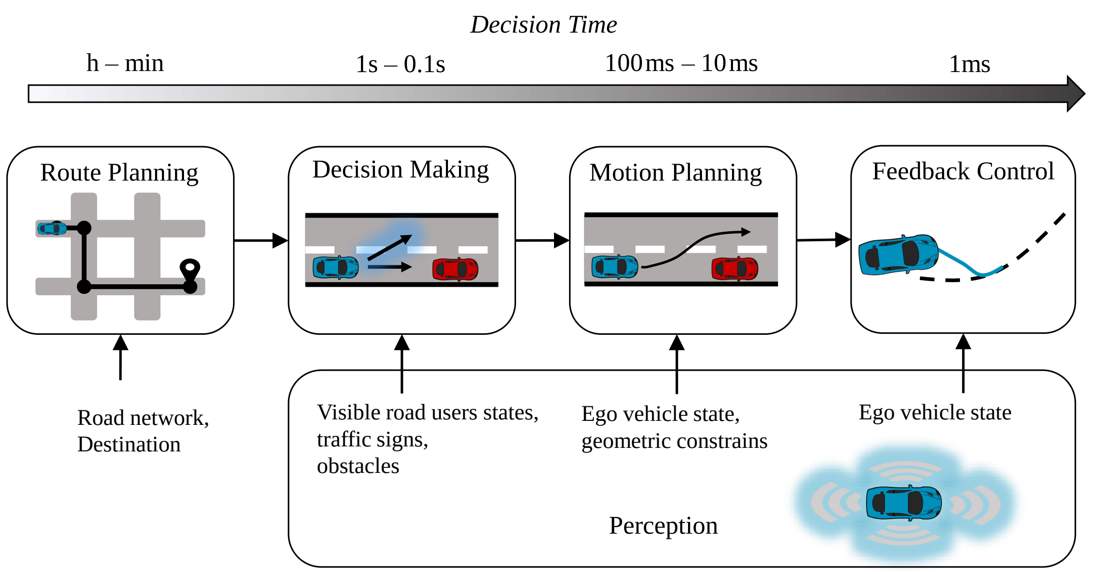


Figure 1.1: A common system architecture for autonomous vehicles.

introduced, they must comply with traffic laws and regulations. Ethical issues must be considered when designing autonomous vehicles, especially when implementing decision making algorithms [5]. Other challenges that autonomous vehicles are facing involve safety validation. Safety validation is the problem of ensuring trust in autonomous systems. It often consists of a combination of on-road testing and simulation. Those operations are very costly as many miles are required to show statistical confidence in the safety metric [6]. One contribution of this thesis involves guaranteeing a minimum probability of success when maneuvering in uncertain environments. In addition, some scenarios such as unprotected left-turns or merging, handled on a daily basis by human drivers remain challenging for autonomous vehicles [7]. This thesis focuses on this last aspect by proposing new techniques to generate decision strategies in crowded environments with limited information.

1.2 *Tactical Decision Making*

The problem of tactical decision making is part of the planning task. The planning task can itself be hierarchically decomposed into three parts that operate at a different time scales [8]:

Route Planning: Given the user destination, route planning involves using information about the road network to find a path from the starting location to the final destination. The planning horizon can range from several minutes to several hours depending on the distance to the destination.

Tactical Decision Making: This module is responsible for prescribing the behavior of the vehicle. Each decision requires between 0.1 s and 1 s depending on the task and the techniques used.

Motion Planning: Given a desired behavior, it considers the obstacles and the road layout to generate a safe and dynamically feasible trajectory for the vehicle.

This thesis focuses on the problem of designing high level decision strategies that describe the behavior of a vehicle. We consider two environments that are currently challenging for autonomous vehicles: urban scenarios (Figure 1.2) and merging in dense traffic (Figure 1.3).

Urban scenarios are complex because they involve variable numbers of traffic participants including cars, pedestrians, bicycles, busses, and trucks, all of which may behave differently over time. In addition, those road users are moving in a spatially restricted area and rely on interactions between each other to navigate through traffic. Autonomously navigating such an environment requires understanding those interactions between traffic participants and anticipating rapid and stochastic changes in their behaviors. Automated vehicles must do so by relying on on-board sensors with limited perception capability. Sensors cannot directly measure the intention of drivers or pedestrians. Moreover, the presence of buildings and parked cars in urban areas occludes the field of view of the sensors. Those scenarios require decision making algorithms that reasons about the potential location of traffic participants, along with their motion over time. An example of such a scenario is illustrated in Figure 1.2. An autonomous vehicle (in blue) must achieve an unprotected left turn while avoiding other traffic participants (in red). Buildings on the side of the road may occlude the field of view of the vehicle.

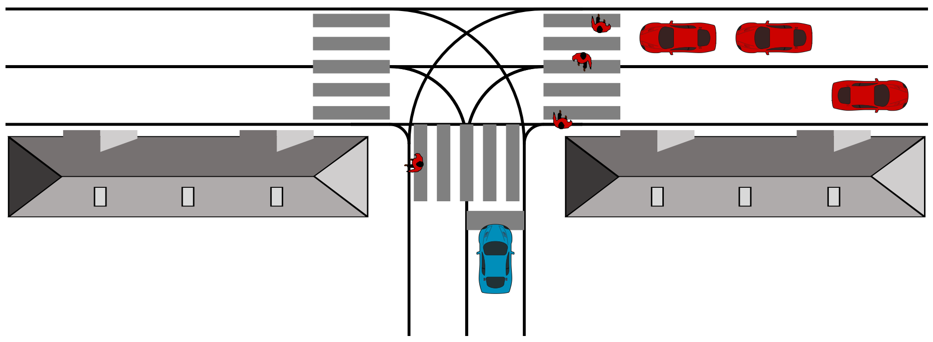


Figure 1.2: Example of an urban scenario with cars, pedestrians and occluded areas. The autonomous vehicle in blue must perform a left-turn while avoiding other traffic participants.

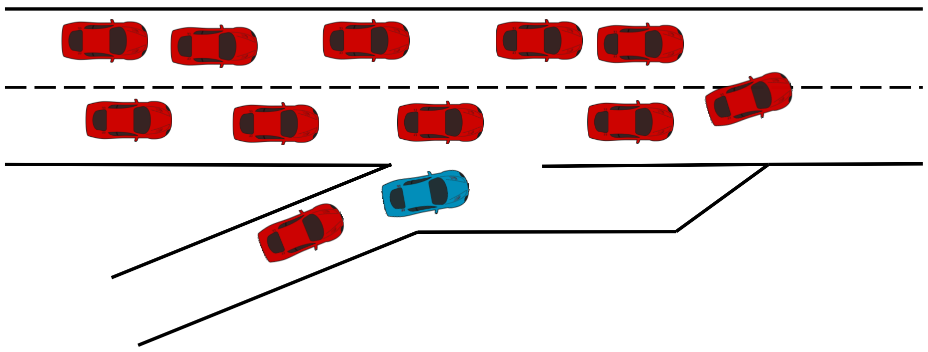


Figure 1.3: Example of a merging scenario with dense traffic. The autonomous vehicle in blue must change lane within a limited distance.

The other scenarios of interest in this thesis are dense merging situations. The autonomous vehicle has a limited distance to merge into incoming traffic. Initially, there is not a no sufficient gap for the vehicle to merge safely. Instead, it must negociate with other drivers and create a gap in the traffic. Such a scenario is illustrated in Figure 1.3. The ego vehicle (in blue) must merge on the two lane road while avoiding the red cars.

Common approaches for designing decision making strategies can be categorized as follows:

Rule-based methods involve hand-engineering hierarchical state machines that attempt to provide explicit strategies for all possible situations. These state machines are useful for solving simple driving problems, but rely on the designer to anticipate how to best handle different situations in advance. In complex scenarios, it places a large burden on the designer, and the inability to consider every edge case often lead to very conservative designs. Hierarchical state machines were used in the DARPA urban challenge [9] and almost lead to an accident at an intersection [10]. Finite-state machines are also not well suited to account for uncertainty on driver intention or sensor noise.

Learning-based methods such as imitation learning involves learning a policy from a human driver by performing supervised learning on some driving data [11]–[13]. Some of these approaches attempt to directly map sensor readings (e.g., raw pixels from a camera [11]) to driving commands. These approaches rely on a large amount of data which are expensive to collect. In addition, they are unlikely to perform better than the human driver used for training.

Planning with respect to a mathematical model of the environment can help reduce the burden on the designer for developing robust decision strategies. The environment is modeled as a stochastic process and a strategy is generated by solving an optimization problem of maximizing a reward function [14], [15]. Finding computationally tractable planning frameworks that take into account uncertainty is often challenging.

Those approaches have many limitations. This thesis presents contributions to planning techniques in partially observable environments by providing a general formulation of the problem and computationally efficient methods to synthesize driving strategies. By modeling driving scenarios as partially observable Markov decision processes, uncertainty about other drivers intentions, and perception limitations can be integrated within the planning framework.

1.3 Approach

Driving is a sequential decision making problem under uncertainty. Decisions such as passing a slow driver or finding the right time to cross an intersection are made with only limited information about the environment. In this chapter, we give an overview of the approach used to generate driving strategies and evaluate their performance.

Mathematically, this problem can be formulated as a partially observable Markov decision process (POMDP). A formal definition of the POMDP framework is given in Chapter 2. In a POMDP, the environment is assumed to evolve according to a known stochastic transition model. The *state* of the environment at a given time contains all the information about the environment. In autonomous driving, it corresponds to the physical state of all the traffic participants (*e.g* position, velocity, heading) as well as their internal state (*e.g.* behavior characteristics, intention). Typically, the internal state is not measurable by the sensors on the autonomous vehicles. Moreover, the perception system might only provide noisy estimates of the positions and is affected by occlusions.

To cope with this lack of information, the decision agent must maintain a *belief state* reflecting its knowledge about the environment. The belief state is often represented by a probability distribution over the possible true states of the world. The ego vehicle updates its belief state at every time step using information given by the perception system. The belief update is performed using state estimation algorithms such as a discrete Bayesian filter or a particle filter.

The decision making loop is summarized in Figure 1.4. At a given time step, the ego vehicle receives an observation, updates its belief, and takes an action. The environment then transitions to a new state. In Figure 1.4 the belief is illustrated by semi-transparent cars and pedestrians that hypothesized occluded traffic participants. Similarly, the ego vehicle maintains different hypotheses on the intention of the detected pedestrian.

In the sequential decision making literature, a decision strategy is called a *policy* and outputs an action to be executed. In a POMDP, a policy maps a given belief

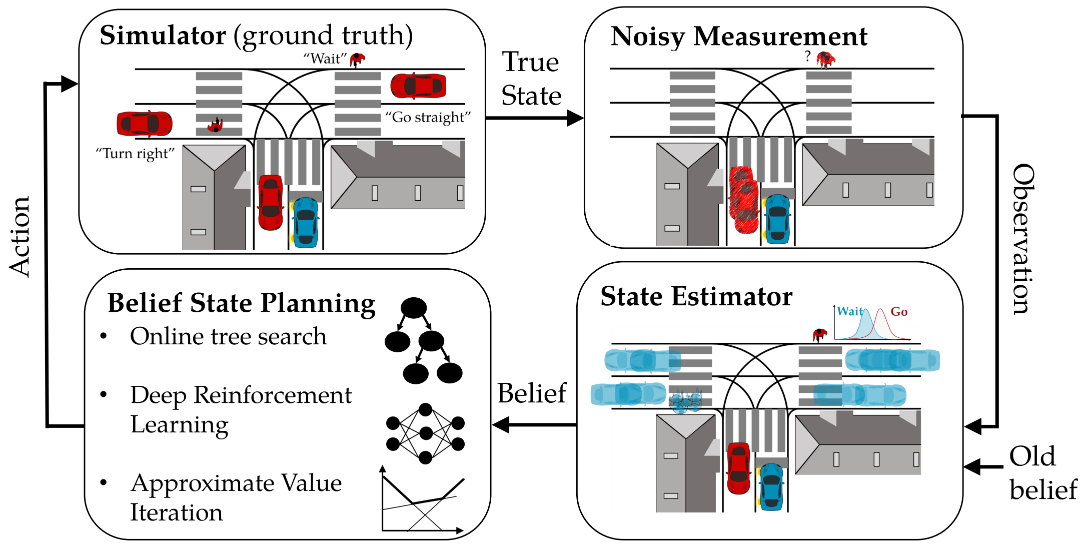


Figure 1.4: Illustration of the decision making loop in a partially observable environment.

state to an action. Given a belief state, the decision strategy outputs the optimal action. This thesis focuses on the problem of computing such a strategy. Finding the best strategy to navigate an environment requires exploring the space of all possible belief states, which is intractable. Instead, we rely on approximation methods. This thesis will discuss three different approaches for computing these strategies: Monte-Carlo tree search [16], approximate value iteration with point-based methods [17], and deep reinforcement learning [18].

Once a decision making strategy has been generated, it is important to evaluate its performance in different driving conditions. To do so, we simulate scenarios with different speeds, positions, and behaviors for other traffic participants and gather some statistics about the performance of the decision strategy. In particular, two main objectives are *safety* and *efficiency*. We collect both the number of collisions that occured in simulations and the time the automated vehicle took to perform the desired maneuver. These values, aggregated over many simulations, allow us to assess the performance of the driving strategy and compare it with other approaches such as rule-based sytems. Given the multiobjective nature of the problem, we use the concept of Pareto domination to assert that a strategy is more suitable than an other. We say that a policy is dominating if it outperforms other policies in both safety and efficiency.

The algorithms presented in this thesis are able to generate driving strategies that outperforms rule-based methods in complex driving scenarios. In addition, a method to enforce a given safety level on a policy is proposed.

1.4 Contributions

This thesis first provide a generic formulation of driving scenarios as partially observable Markov decision processes. The formulation takes into account unobserved internal state of other drivers as well as physical obstacles that can occlude the field of view of the ego vehicle.

A major drawback of the POMDP formulation is the curse of dimensionality. As the number of traffic participants in the environment the number of possible states increase exponentially. To address this challenge we investigate three different solving techniques: online tree search methods, deep reinforcement learning, and offline methods with utility decomposition. Those algorithms are applied to solve a POMDP formulation in three scenarios of interests: unsignalized intersection, occluded urban environments, mergin scenarios.

We first show the capabilities of online tree search and deep reinforcement learning policies before presenting an efficient decomposition method. To bypass the computational cost induced by considering multiple traffic participants, utility fusion leverages the solution of avoiding a single road user to generalize to avoiding many.

The proposed decomposition method approximates the optimal policy. This thesis introduces a new algorithm, deep corrections, that uses deep reinforcement learning to learn a corrective term bridging the gap between the approximated policy and the optimal solution.

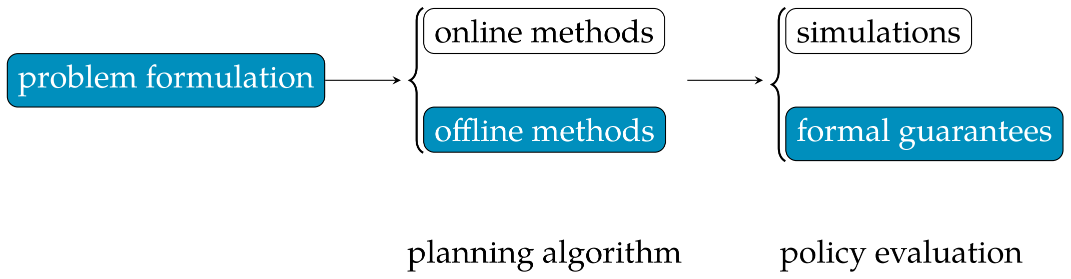


Figure 1.5: Graphical representation of the contributions of this thesis to the field of planning under uncertainty for autonomous driving. Main contributions are highlighted in blue.

Establishing trust in the generated decision strategy is one of the main challenges of autonomous decision making. Model checking can be used to verify with great confidence the performance of a policy. This thesis presents a method using probabilistic model checking for enforcing safety constraints on a decision strategy generated using reinforcement learning.

Finally, this work extends the state-of-art in model checking in partially observable domains. A flexible methodology allows to use approximate POMDP solvers for generating policies with probabilistic guarantees on their performance in POMDPs. Such formal guarantees helps to decide whether to deploy a policy and informs on how much trust can be given to the actions of the agent even in environment with missing information. They can be used concurrently to other evaluation techniques such as simulations and Pareto analysis.

Figure 1.5 illustrates the main contribution of this thesis to the field of decision making under uncertainty for autonomous vehicles.

1.5 Outline

The rest of the thesis is organized as follows.

Chapter 2 describes the formulation of decision making problems for autonomous driving as partially observable Markov decision processes. It first provides background information on POMDPs and different solving techniques: online tree search, point-based methods, deep reinforcement learning. It then highlights the main modeling assumptions about the sensing and actuation capacities of the vehicle as well as assumptions on the inherent properties of driving scenarios.

Chapter 3 explains how to apply online tree search methods to an unsignalized intersection scenario. It shows the ability of belief state planning techniques to anticipate unobserved driver intentions (yield or take way). In addition, we demonstrate the limitations of this approach as the traffic density increases.

Chapter 4 presents an application of deep reinforcement learning (RL) methods to another scenario: merging in dense traffic. It demonstrates that neural representations of the value function allows to scale to large scenarios and can to some extent infer the intention of other drivers. A novel approach to perform reinforcement learning in the belief space is presented.

Although previous chapters demonstrate promising results, the techniques still suffer from the curse of dimensionality: computational limitations for the tree search, and difficult learning for RL. Chapter 5 demonstrates a decomposition method that leverage the solution of avoiding one road users to avoid multiple traffic participants. This technique is applied to two scenarios with sensor occlusions that are formulated using the framework presented in Chapter 2. This chapter shows how to efficiently approximate solutions to large decision making problems using utility fusion.

Chapter 6 introduces the deep corrections algorithms as a method to improve the approximated policy given by the utility decomposition. It is demonstrated on an occluded crosswalk scenario as well as in a multi-agent setting. The multi-agent scenario differs from autonomous driving application but serves the purpose of showing the generality of the algorithm.

Chapter 7 addresses the problem of enforcing safety constraints on the decision strategies. It presents a technique using probabilistic model checking to constrain an RL agent during learning and execution. This technique is applied to an urban navigation environment with sensor occlusions, multiple cars and pedestrians.

Chapter 8 introduces algorithmic techniques to solve model checking problems in POMDPs. Instead of specifying the objective using a reward function, it demonstrates how to use linear temporal logic (LTL) formula. It is shown that this method is more scalable than previous algorithms for computing the probability of satisfying an LTL formula in a POMDP. This technique is applied to compute the maximum probability of success in the occluded crosswalk scenario.

Finally, Chapter 9 summarizes the contributions of the thesis and discusses the remaining challenges in planning under uncertainty for autonomous vehicles.

2 Modeling Autonomous Driving Problems

In order to derive algorithms for generating decision strategies, we must provide a mathematical formulation of the problem. In this chapter, we describe how to model tactical decision making problems for autonomous driving as partially observable Markov decision processes (POMDPs). We first present the formal definitions and background information on POMDPs and then introduce a general formulation of tactical decision making problems for autonomous driving. Casting the problem as a POMDP allows us to formulate an optimization problem over the space of all possible strategies.

2.1 Markov Decision Processes

A Markov decision process (MDP) is a mathematical framework for modeling discrete time sequential decision making problems. It involves an agent taking a serie of decisions in an environment evolving over time according to a stochastic process. The *state* of the environment summarizes the information necessary to fully describe the agent and the environment at a given time. That is, transitioning to a next state, only depends on the previous state and the action the agent took. This property is referred to as the *Markov assumption*.

An MDP is formally defined by a tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$:

- A state space $\mathcal{S}$ represents the ensemble of all the possible states of the environment. In general, there is no particular restriction in the state space definition, it can be finite or continuous.

- An action space $\mathcal{A}$ represents the possible actions that the agent can take. The action space can be either finite or discrete. In this thesis we will mostly discuss problems with discrete actions.
- A transition model T describes the evolution of the environment over time. It is a function such that given a state $s \in \mathcal{S}$, an action $a \in \mathcal{A}$ and a next state $s' \in \mathcal{S}$, the probability of the environment transitionning from s to s' is given by $T(s, a, s') = \Pr(s' \mid s, a)$.
- A reward function R represents a scalar reward or penalty that the agent receives for taking an action in a given state: $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. The reward function is designed to reflect the objective that the agent should maximize. Positive rewards reinforce the corresponding behavior while penalties (negative rewards) discourage it.
- A discount factor γ is a scalar in the interval $[0, 1)$, that attributes more or less values to future reward.

The agent must take a sequence of action, that will maximize a *utility function* defined as follows:

$$U(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s] \tag{2.1}$$

The optimal utility function, U^* is unique and satisfies the Bellman equation:

$$U^*(s) = \max_a [R(s, a) + \gamma \sum_{s'} T(s, a, s') U^*(s')] \tag{2.2}$$

A *policy* is a mapping from state to action. We can define the *utility function* associated to a given policy. From the Bellman equation one can also deduce a state-action value function $Q(s, a)$ satisfying $U^*(s) = \max_a Q(s, a)$. The Q function satisfies the Bellman equation without the max operator. Given a Q function, a policy can be extracted as follows: $\pi(s) = \arg\max_a Q(s, a)$.

The discount factor will influence the result of this optimization problem.

2.2 *Partially Observable Markov Decision Processes*

In a lot of problems, the state of the environment is not directly accessible to the agent. A partially observable Markov decision process (POMDP) is an extension to the MDP framework that can model state uncertainty. It is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, O, R, \gamma)$. The state space, action space, transition model, reward, and discount factor, have the same definition as in an MDP. A POMDP has two additional elements modeling the agent's ability to observe the state.

- An observation space $\mathcal{O}$ represents all the possible observation that the agent can receive. It can be finite or continuous.
- An observation model O describes the probability of observing an observation $o \in \mathcal{O}$ in a given state s' after having taken action a: $O(o, s', a) = \Pr(o \mid s', a)$.

In a POMDP, events happen in the following order: from a state s, the agent takes an action a, the environment transitions to a state s' according to the transition model T and the agent receives an observation o related to s' and a according to the observation model O.

Since the state is not observable, policies are no longer described by mapping from states to actions in a POMDP. Instead, the agent must reason about the action, observation history. Often, this history can be summarized in a sufficient statistic referred to as a *belief* (or belief state). A belief is a probability distribution over states: $b : \mathcal{S} \to [0, 1]$ and $\sum_{s \in \mathcal{S}} b(s) = 1$. Note that the last condition is described using an integral if the state space is continuous.

The agent updates its belief at every time step given a new observation by using Bayesian filtering. The following equation describes how to update a belief state in problems with discrete states:

$$b'(s') \propto \Pr(o \mid s', a) \sum_{s} \Pr(s' \mid s, a) b(s) \quad \forall s' \tag{2.3}$$

where b is the previous belief, a the action the agent just took, o the current observation and b' the new updated belief.

Policies are now described as mappings from beliefs to actions. We define $U^{\pi}(b)$, a belief state utility function associated to a policy π, as the accumulated reward obtained by the agent following π from the belief state b. Most of the POMDP planning algorithms are designed to approximate the optimal utility function $U^{*}(b)$. Given a belief state utility function U, the associated policy can be extracted using a one-step look ahead as follows:

$$\pi(b) = \max_{a} R(b,a) + \gamma \sum_{o \in \mathcal{O}} \Pr(o|b,a)U(b') \tag{2.4}$$

where b' is computed using Equation (2.3), and $R(b,a) = \sum_{s\in\mathcal{S}} b(s)R(s,a)$ is the expected reward in belief b. Similarly as in the MDP case, we can define a belief-action utility function, $Q(b,a)$ satisfying $U^{*}(b) = \max_a Q(b,a)$.

The belief space is a continuous space of dimensionality equal to the number of states in a POMDP. Computing policies maximizing a utility function in a POMDP requires exploring the belief space and is hence much more chalenging than for MDPs. In fact it has been shown that this problem is undecidable [19]. Fortunately, approximation methods have been developed to derive efficient policies in POMDPs [20]. Some of those methods are discussed in the following sections. Chapter 5 and Chapter 6 demonstrate a technique to scale POMDP solving strategies by exploiting a structure inherent to driving scenarios.

2.3 *Offline Methods*

The first category of POMDP planning algorithm is offline methods. Offline planners compute an approximately optimal strategy over the entire state space, prior to execution.

They rely on computing the value function over the belief space. Given a value function, the associated policy can be obtained using a one step look-ahead.

2.3.1 *Value Iteration*

In an MDP, the value function can be computed in time polynomial in the number of states and action. It consists of iteratively applying the Bellman operator as follows:

$$U^{(n+1)}(s) = \max_a [R(s,a) + \sum_{s'} T(s,a,s') U^{(n)}(s')] \; \forall s \tag{2.5}$$

where n corresponds to the n-th application of the operator. As n goes to infinity, $U^{(n)}$ converges to the unique optimal value function U^*, independently of its initial value $U^{(0)}$.

In a POMDP, one could apply the value iteration algorithm in the belief space instead of the state space as follows:

$$U^{(n+1)}(s) = \max_a [R(b,a) + \sum_{o \in \mathcal{O}} \Pr(o \mid a, b) U^{(n)}(b')] \tag{2.6}$$

which is similar to applying the one-step lookahead (Equation (2.4)) recursively. This algorithm was introduced by Sondik [21] but is generally intractable for infinite horizon problems. The first challenge is the *curse of dimensionality*, the value function must cover the continuous belief simplex. The second challenge is the *curse of horizon* which describes the fact that the space of possible plans grows exponentially with the horizon.

2.3.2 *QMDP*

The QMDP approach is an approximation technique introduced by Littman, Cassandra, and Kaelbling [22] that solves the problem under the assumption that the state becomes fully observable after one time step. Practically, it consists of solving the state utility function, $U_{\text{MDP}}(s)$ of a POMDP using Equation (2.5) assuming full observability. In order to incoroporate state uncertainty, the belief action utility function is approximated using a weighted average of the MDP utility function by the belief state:

$$U^*(b) \approx \sum_{s \in \mathcal{S}} b(s) U_{\text{MDP}}(s) \tag{2.7}$$

It has been shown that the resulting utility function is an upper bound on the true solution and that this solution tends to have difficulty solving problems where information gathering is important [20], [23]. The QMDP algorithm has the same complexity as value iteration and scales polynomially with the size of the state space and action space of the POMDP. It is a very computationally affordable approximation technique for solving POMDPs.

2.3.3 *Point-Based Methods*

Although the QMDP approximation is computationally appealing, it might be a poor approximation of the value function. Another approach for solving POMDP offline consists of evaluating the value at a set of carefully chosen belief points. Those algorithms are called point-based methods. Since the major source of intractability is the size of the belief space, by reducing the computation to a finite set of belief points, these methods can scale to POMDP domains with tens of thousands of states.

Point-based methods represents the belief state utility function with a finite set Γ of α-vectors associated to specific belief points. α-vectors are $|\mathcal{S}|$-dimensional vectors defining a linear function over the belief space. Given a set of alpha vectors $\Gamma = \{\alpha_1, \ldots, \alpha_n\}$, the value function is defined as follows:

$$U(b) = \max_{\alpha \in \Gamma} \alpha^\top b \tag{2.8}$$

Point-based value iteration (PBVI) algorithms are a family of POMDP solvers that involves applying a Bellman backup to a set of alpha vectors in order to approximate the optimal value function. Shani, Pineau, and Kaplow survey various PBVI methods. In this thesis, we use SARSOP [17], which has shown state-of-the-art performance in terms of scalability. PBVI algorithms sample the belief space and compute an alpha vector associated to each belief point to approximate the value function at that point. SARSOP differs from other PBVI algorithms by relying on a tree search to explore the belief space. It maintains an upper and lower bound on the value function, which are used to guide the search close to optimal trajectories. The algorithm is given an initial belief point and only explores relevant regions of the belief space. That is, regions that can be reached from the initial belief point under optimality conditions.

Note that all the methods discussed in this section assumes the POMDP has finite states, actions, and observation. As autonomous driving problems are continuous by nature (because it involves physical quantities such as positions and velocities), solving them using offline methods require discretizing the state space. As a consequence it limits their scalability to high dimensional state spaces. In the next section, we discuss techniques that can, to some extent, handle continuous spaces.

2.4 *Online Methods*

Online methods plan from the current belief state up to a certain horizon. As a consequence, online planning algorithms consider only the states reachable from the current belief and at each time step the solver computes the (approximately) optimal action. The best action is typically recomputed after each interaction with the environment.

A popular online algorithm is partially observable Monte Carlo Planning (POMCP), which relies on sampling from a generative model [16]. POMCP is an extension of the Upper Confidence Tree (UCT) algorithm [25] with partially observable state variables. The algorithm takes as input a belief state. From this belief state, it builds a tree where each node represents a history h, which is a sequence of actions and observations. Note that the history can be summarized by a belief state. Nodes are sampled according to a given POMDP model. The construction of the tree involves iterating through the following three steps many times:

Expansion: If the node is not in the tree, we explore the outcome of all the possible actions and initialize $N(h, a)$ and $Q(h, a)$, which are the number of times we visited the node h taking action a and the associated value function.

Rollout: We simulate up to a desired depth according to a rollout policy.

Search: If the sampled state is already in the tree, we choose the action that maximizes $Q(h,a) + c\sqrt{\frac{N(h)}{N(h,a)}}$, where $N(h)$ is the number of times the history was visited and $N(h,a)$ is the number of times the action and history was visited. The parameter c controls the balance between exploration and exploitation.

After each iteration, the information is then propagated up to the root node. The POMCP algorithm converges to the optimal policy as the number of tree queries increases.

One of the drawbacks of POMCP is that it cannot handle continuous state spaces. When sampling a continuous variable from the initial belief, the probability of visiting the same state twice is infinitesimally small, resulting in a very wide tree with a depth of one. This issue can be addressed using progressive widening [26]. Progressive widening involves defining when to explore new states in the tree. It is controlled by two parameters α and k. The selection criteria is as follows:

- Compute $k' = kN(h,a)^{\alpha}$.
- If k' is greater than the number of children of the node (h,a), then we sample a new state. Otherwise, we choose a state that has already been visited.

The branching factor of the search tree can be affected by tuning k and α. When the transition model is very noisy, one typically wants a large branching factor (which can be achieved by increasing α, for example).

2.5 *Modeling Driving Scenarios*

Autonomous driving can be described as a sequential decision making problem where the autonomous vehicle receives a partial observation of the environment and must take an action to maximize a high level objective. In this section we introduce a general model of driving scenarios formulated as POMDPs.

2.5.1 State Space

To model autonomous driving scenarios, we suggest an entity based representation of the state. An entity can be a pedestrian, a vehicle or a stationary obstacle such as a parked truck or a building. Including stationary obstacles allows the formulation to generalize to a variety of occluded environments. Other information about the environment like road topology is not part of the state but is still assumed to be known by the agent. This information is embedded in the formulation of the transition model and action space by restraining the motion of the different traffic participants according to the traffic rules and the geometry of the roads.

The state of the environment is defined by the following set of variables:

- $P_e^t = (x_e^t, y_e^t, \theta_e^t, v_e^t, a_e^t, l_e, w_e)$ describes the physical state of the ego car at time t.
- $P_i^t = (x_i^t, y_i^t, \theta_i^t, v_i^t, a_i^t, l_i, w_i, k_i)$ for $i = 1 \dots n$, describes the physical state of the n other traffic participants (cars or pedestrians) at time t.
- β_i^t describes the internal state of the other traffic participants, characterizing their behavior at time t.

For each entity i, x_i^t and y_i^t represent the position of an entity, θ_i^t its heading, v_i^t its longitudinal velocity, a_i^t its longitudinal acceleration, l_i and w_i the length and width of the entity, and k_i is a variable indicating whether i is a pedestrian, a car, or a stationary obstacle. Figure 2.1 illustrates the different variables representing the physical state of a vehicle.

2.5.2 Action Space

In tactical decision making task, the action space is high level. We assume that a motion planner and control system are present on the vehicle and that it is their responsibility to ensure that those actions are executable.

The action spaces for the three scenarios of interest in this thesis are defined as follows:

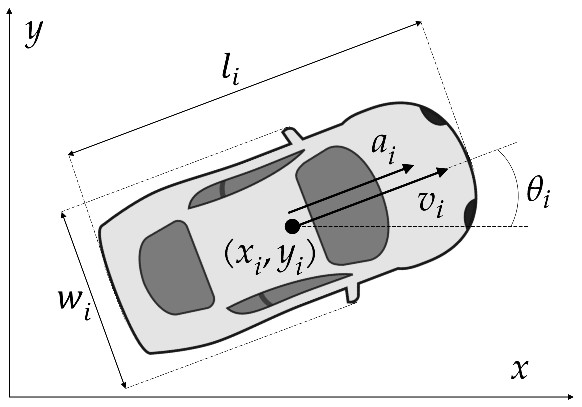


Figure 2.1: Illustration of the different variable representing the physical state of a vehicle i in a Cartesian frame.

Unprotected left-turn The ego vehicle is given a path to follow and only controls its longitudinal motion along this path. The action space consists of four discrete acceleration levels: $\{-4\,\mathrm{m\,s^{-2}}, -2\,\mathrm{m\,s^{-2}}, 0\,\mathrm{m\,s^{-2}}, 2\,\mathrm{m\,s^{-2}}\}$

Occluded crosswalk In the occluded crosswalk the ego vehicle also controls its longitudinal motion only, actions are the same as for the unprotected left-turn.

Dense merging In the dense merging scenario the ego vehicle must control both its longitudinal motion and lateral motion to achieve a lane change. The action space is the cross product of a set of longitudinal actions and a set of lateral actions. The longitudinal actions are the same as for the other two scenarios. The lateral actions correspond to changing lane to the left, to the right, or staying in its current lane. More details on how those actions are executed are provided in Chapter 4.

change to reflect other merging scenario

.

2.5.3 *Transition model*

The transition model dictates how the state of the environment evolves over time. In our driving scenarios we can decouple this evolution into two parts: a dynamics model and a behavior model. The dynamics model describes how to update the physical state of a single entity given an actuator command such as an acceleration or heading input. It is usually given by a simplified model of the physics of the vehicle. An example is the discrete time point mass dynamics model. Given a longitudinal acceleration input a and desired heading θ, the physical state of the vehicle is updated as follows:

$$x_i^{t+1} = x_i^t + v_i^t \cos(\theta_i^t)\Delta t + \frac{1}{2} a \cos(\theta_i^t)\Delta t^2 \tag{2.9}$$

$$y_i^{t+1} = y_i^t + v_i^t \sin(\theta_i^t)\Delta t + \frac{1}{2} a \sin(\theta_i^t)\Delta t^2 \tag{2.10}$$

$$v_i^{t+1} = v_i^t + a\Delta t \tag{2.11}$$

$$\theta_i^{t+1} = \theta \tag{2.12}$$

where Δt is the time step. It is common to add an additive Gaussian noise to these update equations to model uncertainty about the physical model [27]. The addition of a noise term makes the dynamics model stochastic. This model is consistent with the Markov assumption as only information about the current state is needed to compute the next state. We will note T_d the transition function associated to the dynamics.

The behavior model describes how the actuator command is computed based on the state of the environment. In this work we model it as a distribution over possible actions given the current state of the environment. We note it T_β.

As a consequence, we can write the transition model of a single entity i as follows:

$$T_i(s_i', s) = \prod_{a_i \in \mathcal{A}} T_d(s_i' \mid s_i, a_i) T_b(a_i \mid s, \beta_i) \tag{2.13}$$

This equation allows to update the state of one entity conditioned over the full state

s of the environment, and β_i the behavior of the entity. The behavior model comes from external model. It could be learned from data, use a physics based model (*e.g.* constant velocity with noise) or a parametric rule-based model. In this thesis we mostly use the latter, other entities behavior are modeled using simple parametrics models such as the Intelligent Driver Model (IDM) [28] with added stochasticity.

To update the full state of the POMDP, we can take the product over all the individual transition functions:

$$T(s, a, s') = T_d(s'_e \mid s_e, a) \prod_{i=1}^{n} T_i(s'_i, s) \tag{2.14}$$

The ego vehicle does not have a behavior model as it is the controlled agents, its action is a. Figure 2.2 illustrates the transition model using a dynamic Bayesian network. The edge connecting P_e^t to P_e^{t+1} correspond to the term $T_d(s'_e \mid s_e, a)$ in the factored transition function. The other transition edges ending in P_1^{t+1} represent the term $T_1(s'_1, s)$.

2.5.4 *Observation model*

The observation model (or sensor model) represents the ability of the ego vehicle to observe its surroundings. This model is highly dependent on the perception system on board of the vehicle. In this thesis the perception system is modeled by a simple set of assumptions.

First, we consider that the internal state of other traffic participants cannot be sensed by the ego vehicle. This is generally a reasonable assumption since we do not have access to the state of mind of other drivers of pedestrians. Considerations of features like turn signals or eye contact could help relaxing this assumption but are out of the scope of this work.

We assume that the ego vehicle can perfectly sense its own physical state. In addition, the perception system can give measurements of the positions, velocities, heading, and dimensions of other traffic participants or stationary obstacles. Those measurements are not perfect and are subject to limitations like noise, false negative

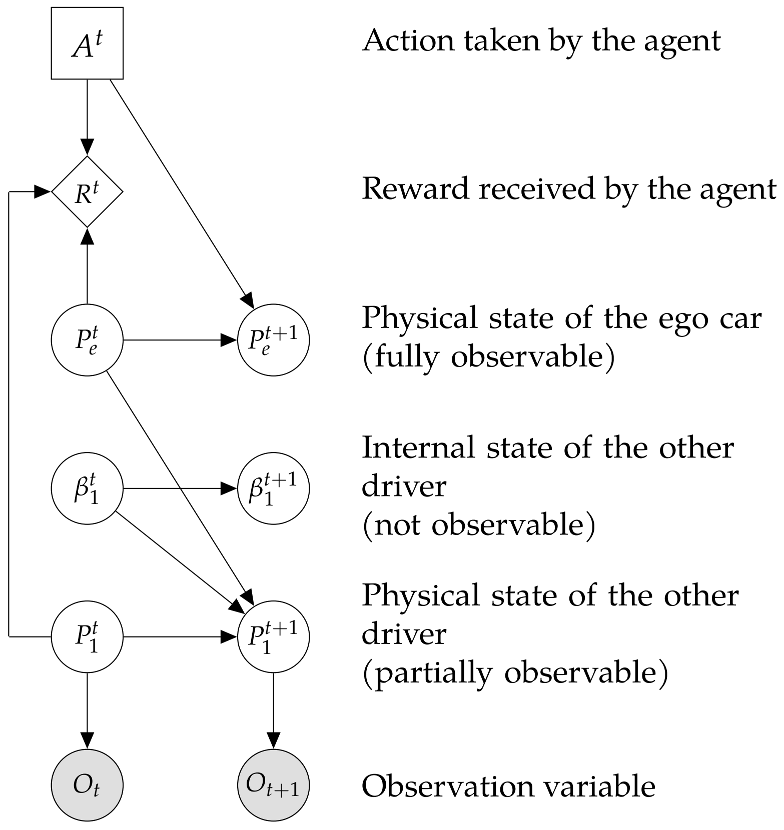

A^t
Action taken by the agent
R^t
Reward received by the agent
P_e^t
P_e^{t+1}
Physical state of the ego car (fully observable)
β_1^t
β_1^{t+1}
Internal state of the other driver (not observable)
P_1^t
P_1^{t+1}
Physical state of the other driver (partially observable)
O_t
O_{t+1}
Observation variable

Figure 2.2: Structure of an autonomous driving POMDP represented as a dynamic Bayesian network. This diagram only shows one other driver to not clutter the figure. However, the problem formulation considers several other traffic participants.

and false positive, and occlusions.

2.5.5 *Reward function*

Before designing the reward function, it is important to clearly define the objective that we wish our agent to achieve. In autonomous driving we mostly focus on the following objectives: safety, efficiency, comfort, and legality.

In this work, the legal aspect will not be considered when designing the reward function. In each decision making problem, the ego vehicle is constrained to a given path and only controls its longitudinal actions. Moreover, its speed is capped to the maximum authorized speed limit.

Safety, efficiency, and comfort can be encoded mathematically within the reward function. Safety is taken into account by attributing a large penalty when the ego vehicle collides with another traffic participants. Efficiency can be encoded by having a bonus for reaching a certain position in the environment or as a penalty for deviating from a desired velocity. Comfort can be modeled by penalizing jerk or accelerations. These different objectives are then combined using a weighted sum as follows:

$$R(s,a) = \lambda_{\text{safety}} R_{\text{safety}}(s,a) + \lambda_{\text{efficiency}} R_{\text{efficiency}} + \lambda_{\text{comfort}} R_{\text{comfort}} \tag{2.15}$$

where λ_{safety}, $\lambda_{\text{efficiency}}$, λ_{comfort} are scalar values that can be tuned to obtain different behaviors. We can replace the functions R_x by a mathematical expression encoding the corresponding objective, for example:

$$\begin{split} R(s,a) = \lambda_{\text{safety}} \mathbb{1}_{\text{collision}}[s] + \lambda_{\text{efficiency}} |v_{\text{ego}} - v_{\text{desired}}| + \\ \lambda_{\text{goal}} \mathbb{1}_{\text{reach goal}}[s] + \lambda_{\text{comfort}} ||a||^2 \end{split} \tag{2.16}$$

where we have two terms for efficiency, one for maintaining a desired speed and one for reaching a goal. Typically, λ_{goal} is positive, and all the other weights would be negative. The choice of those weights is often the results of an iterative process. One would fix a set of weight, evaluate the corresponding strategy, and based on the evaluation metrics, decide to change the relative weights of one objective over another.

In general, designing reward functions can be very challenging and can lead to a value alignment problem where the agent does not behave as expected [29]. A more rigorous treatment of encoding ethical values within the implementation of decision making algorithm can be done using methodologies like value sensitive design [30].

2.6 Discussion

This chapter demonstrated a generic way to model tactical decision making problems under uncertainty for autonomous driving. By casting the problem as a partially observable Markov decision process, we can capture the full complexity of the environment. The transition model accounts for uncertain behaviors and dynamics using stochastic models. The observation model captures uncertainty in the state by accounting for both sensing limitations and uncertainty about human drivers intentions which we model using a latent internal state. Finally, the reward function allows to account for different objectives that will be reflected in the resulting policy according to some tunable weights.

The model from this chapter do not constrain the number of entities present neither limit the variables to be discrete. The most general formulation assumes an infinite possible number of vehicles, continuous physical states, and potentially a high dimensional internal state. However in practice, POMDP solving algorithms can only handle so much complexity in the model. We will see in the next chapters how to practically adapt this model such that current state-of-the-art POMDP planning algorithms can be applied to common autonomous driving scenarios.

3 Online Planning at Intersections

In this chapter we present an application of online POMDP solver to autonomously navigating urban intersection. The ego vehicle must achieve a left turn at an intersection and estimates whether drivers on the main road are going to brake or not. We will show that the POMDP approach allows to balance between conflicting objectives such as safety and efficiency and outperforms rule-based approaches. This chapter is based on research conducted with Akansel Cosgun and Kikuo Fujimura [31].

3.1 Related Work

Several approaches have been employed to address intersection crossing. One approach involves hand-engineering hierarchical state machines that attempt providing explicit strategies for all possible situations. These state machines are useful for solving simple driving problems, but rely heavily on the designer to anticipate how to best handle different situations in advance. hierarchical state machines were used in the DARPA urban challenge [9] and almost lead to an accident at an intersection [10].

Learning-based methods can help reduce the burden on the designer for developing robust decision strategies. One type of learning approach known as behavioral cloning involves learning a policy from a human driver [11]–[13]. These approaches rely on a large amount of data and are unlikely to perform better than the human driver used for training.

Other works rely on planning algorithms using a POMDP formulation [14], [15], [32], [33]. Most of them relied on a grid based representation of the state space which can be hard to scale to larger scenarios. Moreover they only considered one other traffic participant. Later approaches using deep reinforcement learning have also been used to solve intersection navigation problems [34]. Although the resulting policy learns efficient behavior and is fast to execute online, the neural representation can often be unreliable or difficult to generalize to discrepancies between training environment and real world. This technique will be further studied in the next chapter. Combining deep reinforcement learning policies and tree search method would be a promising direction of future work.

In this work we propose an online decision making algorithm to cross an urban intersection. We rely on the POMCP algorithm [16] augmented with progressive widening [26] to solve the POMDP in the continuous space and consider many other vehicles. By modeling the problem as a POMDP, the autonomous system can dynamically change its decision to adapt to the behavior of other agents. Vehicles are at an unsignalized T-junction with traffic flowing in both directions, as illustrated in Figure 3.1. The autonomous vehicle is initially stopped at the intersection and tries to turn left or right. The nominal path is assumed to be generated by a high level route planner. Our algorithm computes the acceleration profile along this path. In this chpater, we consider noisy position and velocity measurements. Sensor occlusions are addressed in the next chapter. Finally, the internal states of the vehicle are not observed, they are estimated using an interacting multiple model (IMM) filter [27].

A similar approach has been studied by **hubmann2017** [35], where they use another tree search method, Adaptive Belief Tree (ABT) [36], to solve an intersection navigation scenario. They also consider uncertainty about other drivers behavior but do not consider perception noise. The ABT solver internally rely on particle filter to update the belief over the state of the environment. Regardless of the technicalities of the tree search method used, they draw similar conclusion as in our work. In particular, it is shown that the POMDP formulation is well suited to address these driving scenarios.

In the next section, we discuss the detail of the POMDP model of the intersection scenario. In Section 3.2.3, we explain how to estimate driver behavior using an IMM

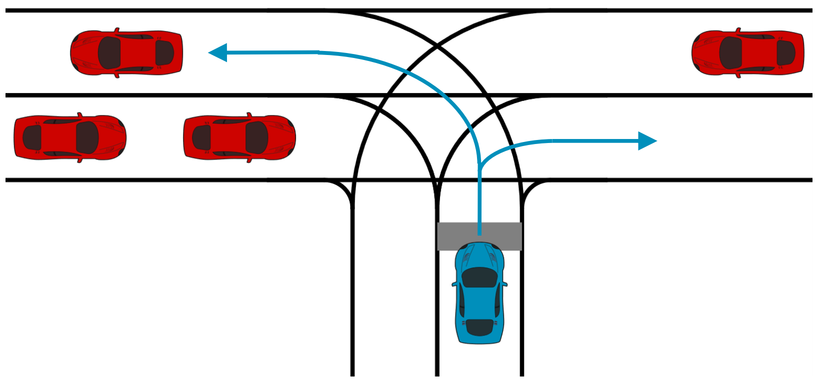


Figure 3.1: Example of an intersection navigation scenario. The objective is to turn left or right safely. The paths (blue trajectories) are given and the autonomous vehicle (in blue) must decide on the acceleration to apply.

filter. Finally we present empirical results and demonstrate that our online POMDP planner outperforms rule based policies in both safety and efficiency.

3.2 *Intersection Navigation Model*

To formulate this intersection navigation problem as a POMDP we follow closely the model discussed in Chapter 2.

3.2.1 *State space and transition model*

The physical state of every vehicle is the same as described in Section 2.5.1. Positions are defined with respect to a fixed Cartesian frame with origin at the center of the intersection. The ego vehicle can take four actions which consist of discrete acceleration levels (Section 2.5.2). When applying an action, the ego vehicle follows a point-mass dynamics model and is constrained to a given path, either left turn or right turn.

The internal state of other driver is modeled by an indicator variable taking one of two values. Each value corresponds to one of two different transition models: constant velocity (CV) and constant acceleration (CA) [27]. These models describe various behavior including yielding to the ego vehicle, maintaining speed or accelerating. The physical state transition function follows linear Gaussian dynamics:

$$Pr(P_i^{t+1} \mid P_i^t) = \mathcal{N}(P_i^{t+1} \mid \mathbf{T}P_i^t, \mathbf{Q}) \tag{3.1}$$

where **T** is the state transition matrix and **Q** the process noise. These matrices are different for each kinematic model and follow the equations of Bar-Shalom, Li, and Kirubarajan [27]. The Gaussian dynamics provide a convenient representation of the transition as it is describing continuous variables with a minimal amount of information so that it is computationally cheap to query the transition model. The internal state β_i^t can take one of two values, which dictates which of the CV or CA model driver i is following. At each time step β_i^t can change according to a switching probability matrix p where p_{ij} is the probability from switchin to model j from model i.

3.2.2 *Observation model*

The ego vehicle is assumed to have perfect knowledge of its own state. Position, heading and velocities of other drivers are observable while their acceleration and their internal state are not. In addition, we assume that position and velocity measurement are noisy and follow a Gaussian distribution:

$$\Pr(O_i^t \mid P_i^t) = \mathcal{N}(O_i^t \mid \mathbf{H}P_i^t, \mathbf{R}) \tag{3.2}$$

where O_i^t is a four dimensional vector containing the measured position, heading, and velocity of vehicle i at time t. **H** is the observation model matrix and **R** is the observation noise matrix. We assume that the measurements are independent (**R** diagonal) and characterized by the stand deviation σ_p for the position measurement and σ_v for the velocity measurement.

3.2.3 *Estimating drivers behavior*

From our state space formulation and the assumption on the observation model we can factor the belief state in three sets of variable:

- The ego car physical state, which is assumed to be perfectly known
- A distribution over the other drivers physical state. Given the assumption of a Gaussian measurement model and dynamics, we assume that the physical states follow a Gaussian distribution $\mathcal{N}(\hat{P}_i^t, \hat{\Sigma}_i^t)$ where $\hat{P}_i^t$ and $\hat{\Sigma}_i^t$ are the estimated mean and covariance of the physical state of vehicle i at time t.
- A distribution over the two possible kinematic models, $\{\mu_{1i}^t, \mu_{2i}^t\}$ representing the probability of vehicle i following the CV model and the probability of car i following the CA model respectively.

The use of a finite set of Gaussian dynamics to model driver behavior allows us to apply a powerful filtering algorithm to estimate both the true physical state of other vehicle and their behavior. This algorithm is the Interacting Multiple Model (IMM) filter. IMM has been used in many tracking applications including pedestrian intention prediction [37], [38], as well as lane changing detection in autonomous driving [39].

The IMM mixes two Kalman filters for both kinematic models CV and CA and update both the state estimation and the probability distribution on the model at each time, given an observation. The IMM algorithm consists of three steps: mizing, filtering, and combining. The first step computes an estimate of the state with respect to the two transition models and from these estimates computes two mized inputs (a linear combination of both). The two mixed inputs are then filtered using a standard Kalman filter [27].

In the POMDP context, we used the IMM as the belief updater. It takes as input a belief state and an observation and returns the updated belief state. The IMM plays a role analogous to Equation (2.3) for this specific formulation with two possible Gaussian dynamics.

3.2.4 Reward

To fully specify our POMDP model we are left with desiging a reward function. In this scenario we reward the agent for reaching a final position in the given path. In addition the agent receives a small penalty for each action and a large penalty for collision. The numerical values of each bonus or penalty are tuned in simulation.

The POMDP model of this problem is fully specified and can be used by a planning algorithm. Since we consider multiple vehicles, the dimensionality of state space is high. Only online planners are suited to solve this problem. POMCP [16] augmented with progressive widening is a good candidate method for this problem.

3.3 Experiments

The performance of our online decision making algorithm is evaluated in simulation and compared to a rule-based policy. We demonstrate the ability of POMDP formulations to trade between different objective through the reward function.

3.3.1 Experimental setup

We used the SUMO simulator [40] for our experiments. Simulated vehicles in SUMO relies on the Intelligent Drive Model (IDM) [28] to navigate, as well as a couple rule based policies that takes into account interactions between drivers. Hence our transition model (relying on Gaussian dynamics) is different than the model used in the test environment. The motivation for this mismatch is to assess the robustness of our algorithm to model discrepancies that would occur in real-world applications.

The simulator outputs the exact position, velocity, and orientation of the vehicles. To simulate the perception of the autonomous vehicle, we added white noise to the position and velocity measurements. The simulation parameters are given in Table 3.1. The traffic density is expressed as the probability of a vehicle going through the intersection every second. The noise parameters correspond to the standard deviations σ_p and σ_v.

Table 3.1: Parameters of the simulation environment for the intersection scenario.

Traffic density	0.2
Position sensor noise	0.1 m
Velocity sensor noise	$0.1\,\mathrm{m\,s^{-1}}$
Maximum Speed	$13.88\,\mathrm{m\,s^{-1}}$

Table 3.2: POMCP solver parameters used in the experiment.

Depth	15
Exploration constant	20.0
Tree queries	2,000
Rollout policy	TTC Policy
PW α	0.2
PW k	4.0

The decision making loop follows a process similar as Figure 1.4. The agent start with a prior belief b_t. From this belief, an approximately optimal action is given by the POMCP algorithm. Then, a simulation step in SUMO is run and the states of all the vehicles are updated. After this step, the ego vehicle observes the environment and updates its belief to b_{t+1} using the IMM algorithm. The decision, measurement, and belief update are made every 0.25 s. The parameters of the online solver are reported in Table 3.2.

To evaluate the perofrmance of algorithm we used the following metrics: average number of collisions, average time to cross the intersection, success rate, average time when a car is braking, and average time when a car is stopped. A success is defined when the ego vehicle reaches the end of the given path without crashing and in less than a minute. The first three metrics account for safety and efficiency. The braking time and the waiting time capture the impact of the ego car on the current traffic.

3.3.2 *Baseline policy*

We defined a simple rule based policy to serve as a baseline. It uses a threshold on the time to collision (TTC) to decide when to cross. Consider an imaginary

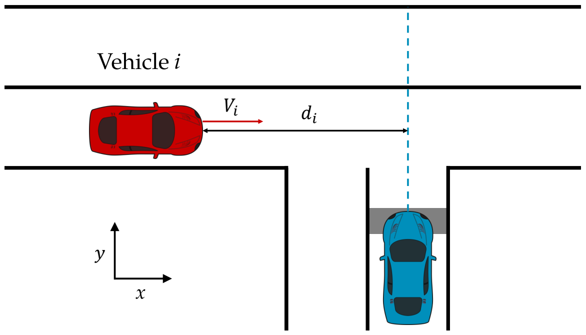


Figure 3.2: Illustration of the distance used to compute the TTC between a vehicle and the ego vehicle in the intersection scenario

line starting from the ego car aligned with the vertical axis. The TTC with another vehicle i in the intersection is the time it takes for the vehicle to reach that line. For vehicle i, it is estimated by dividing d_i by V_i, where d_i is the distance indicated in Figure 3.2 and V_i is the velocity of vehicle i relative to the ego vehicle. This estimate of the time to collision assumes that vehicle i follows a constant velocity model. If the TTC exceeds a threshold for two consecutive time steps of 0.1 s, the vehicl starts the crossing phase. The crossing phase follows the IDM. The redundant check for the TTC is designed to make the decision more robust to poor estimate of the TTC.

3.3.3 *Results*

We first analyzed the influence of the TTC threshold on the different metrics and chose the threshold that results in zero collisions over a thousand simulations. Figure 3.3 shows how four metrics vary with respect to the choice of the TTC threshold. Due to measurement noise, we can see that even for high thresholds there are still small fluctuations in the collision rate. The chosen threshold for comparison with the POMCP policy is 4.5 s. This threshold guarantees success for both the right and the left turn scenarios under the traffic condition described in Table 3.1, without being overly conservative.

The reward function in the POMDP formulation can be used to tune the behavior of the agent in favor of one objective over another. By increasing the cost for each

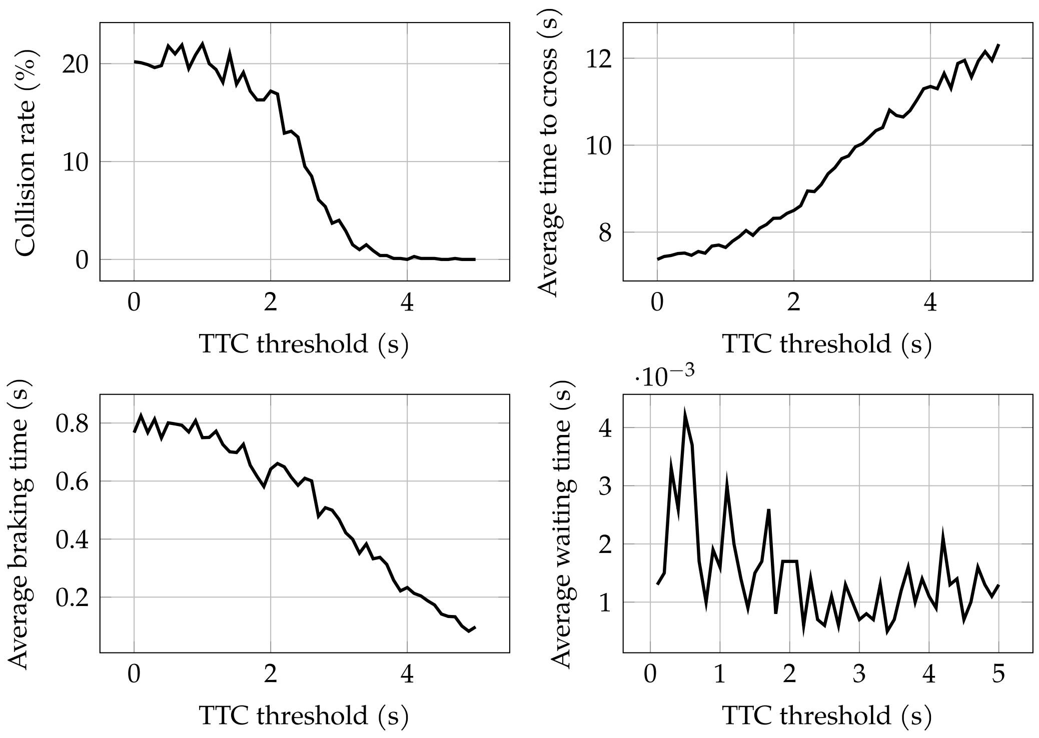


Figure 3.3: Four metrics with different TTC thresholds for right-turns with a traffic density of 0.2.

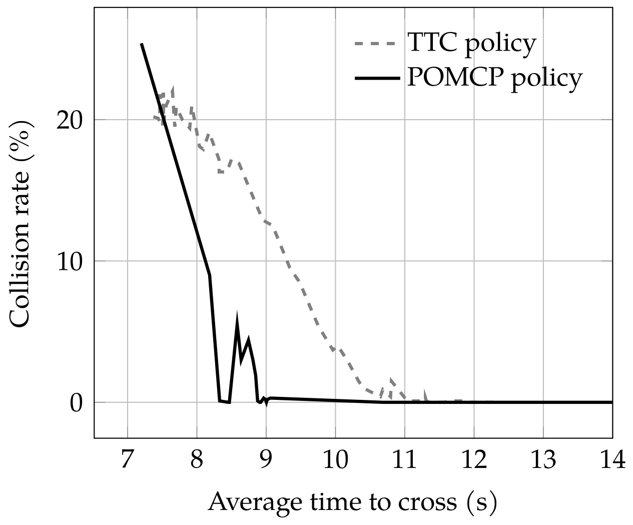


Figure 3.4: Trade-off between collision rate and time to cross as key parameters are varied for the policies in the right-turn scenario with a traffic density of 0.2.

action, the accumulated reward will be highly dependent on the time it takes to complete the maneuver and incentivize the agent to cross as fast as possible at the expense of a greater risk of collision. By tuning the reward function we can balance these two objectives. Figure 3.4 illustrates the trade-off between safety and efficiency for the right-turn scenario. We compare the performance of the POMCP policy with different action costs against the TTC policy with various thresholds. The POMCP policy clearly dominates the rule-based policy with respect to both objectives.

From the previous behavior tuning experiment we selected a conservative set of parameters for the POMCP policy (Table 3.3) to compare against the TTC policy with a threshold of 4.5 s. The penalties chosen for each action differ in order to favor forward motion. We also compared a gainst a random policy to asses the general difficulty of the scenarios. Results are summarized in Table 3.4.

For the right-turn scenario, both the POMCP and TTC policies achieve 100% success rate and the POMCP policy outperforms the TTC policy in average time to cross the intersection. The waiting time is higher for the POMCP policy but it still does not exceed 10 ms on average. The results show that both the POMCP policy and the TTC policy are safe for the right-turn scenario, even under some measurement

Table 3.3: Reward function parameters for the fastest safe POMCP policy.

Collision penalty	$-2{,}000.0$
Acceleration penalty	-4.98
Maintaining speed penalty	-4.99
Moderate braking penalty	-5.0
Strong braking penalty	-5.02
Crossing reward	100.0

Table 3.4: Performance of TTC and POMCP policies on left-turn and right-turn scenarios.

Policy	Time to cross (s)	Braking time (s)	Waiting time (s)	Collision rate (%)	Success rate (%)
Right Turn					
POMCP	10.65	0.37	0.031	0.0	100.0
TTC	10.73	0.12	0.001	0.0	100.0
Random	55.60	37.98	20.40	9.80	0.40
Left Turn					
POMCP	10.37	0.38	0.13	0.2	99.0
TTC	10.77	0.19	0.001	0.0	100.0
Random	34.11	17.05	10.29	75.70	0.10

noise. Moreover, the POMCP policy reaches the goal faster than the TTC policy but will cause the other users to brake and wait more often.

For the left-turn scenario, the TTC policy achieved zero collisions and 100% success rate, wheras the POMCP policy still has some collision (0.2%) and time-outs, leading to a success rate of 99%. The braking time and waiting time are also higher. For left turns, similar conclusions can be drawn on the time to cross, the braking time, and the waiting time. However, the POMCP policy still has some collisions. One explanation is the difference between the generative model and the simulator model. In order to assess the discrepancies between the two, we measured the error in the position prediction as a function of the planning horizon. We found an average error of 2.15 m when predicting the motion of the other cars ten steps ahead (2.5 s). This difference in the predicted position prevents the algorithm from predicting some rapid maneuver changes. A more sophisticated generative model, combined with a good internal state estimator could improve performance, and we believe zero collision rate could be achieved for left turns.

In order to assess the scalability of the two policies, we analyzed the evolution of the metrics with an increasing traffic density for the right-turn scenario. Both policies achieved a collision rate of 0%, but they were subject to time-outs. Figure 3.5 shows that until a density of 0.7, the POMCP policy has a success rate at least as high as the TTC policy. For every tested traffic density, the POMCP policy takes on average less time to cross the intersection with a maximum difference of 6.12 s for a traffic density of 0.5. The POMCP policy manages to cross the intersection more often and faster than the TTC policy up to a certain traffic density at the expense of somewhat greater disruption to the traffic. The choice of the reward function penalizing the actions makes the ego car more eager to enter the intersection and is not penalizing for making the other drivers brake or wait. The problem formulation can explain the higher braking time and waiting time for the POMCP policy.

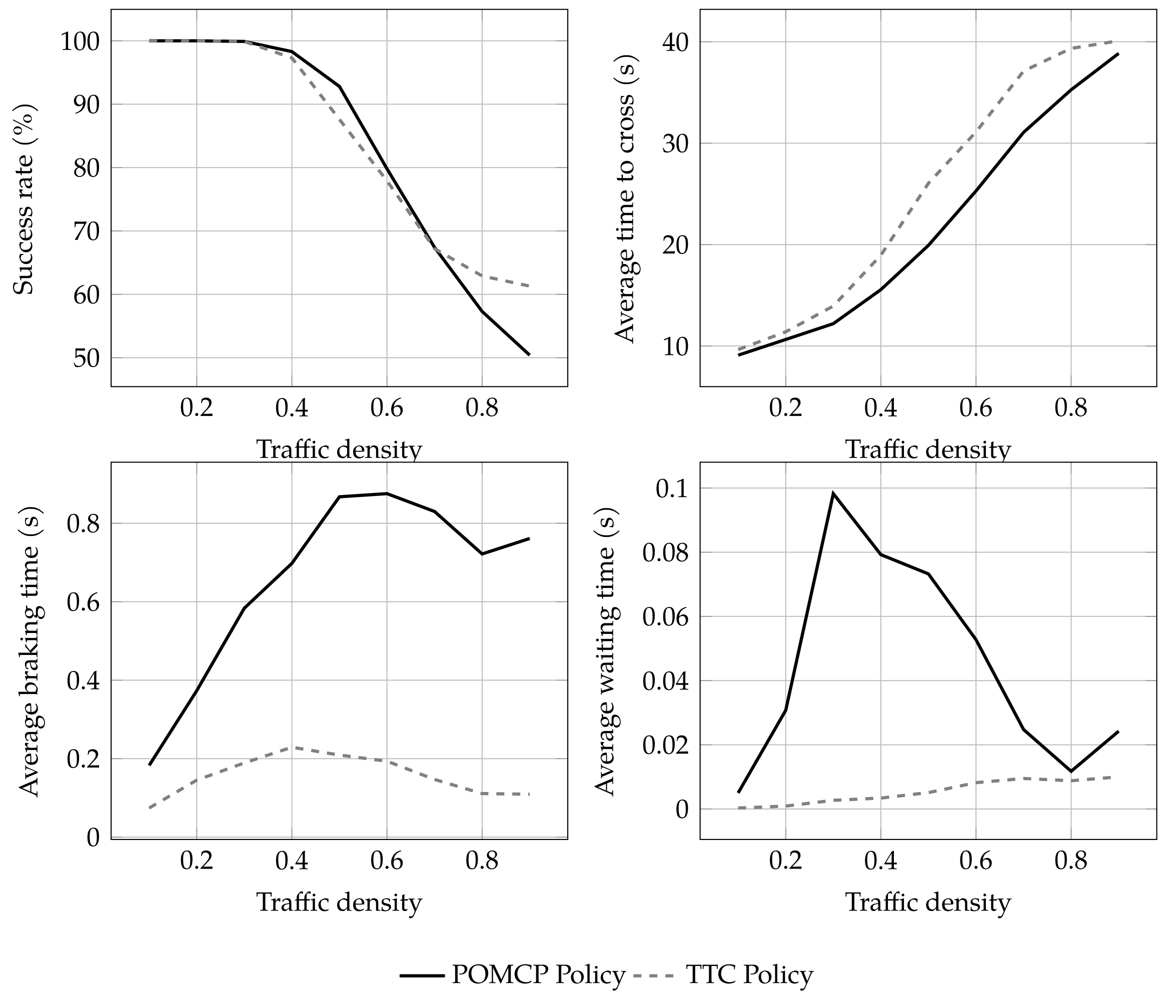


Figure 3.5: Metrics for varying traffic density for both the TTC and the POMCP policies in the right-turn scenario.

3.4 Discussion

In this chapter we demonstrated how to model an intersection navigation problem using the POMDP framework presented in Chapter 2. An online tree search method is used to solve the problem in the continuous space. Empirical results showed that the approach performs better than a rule based policy. Even by assuming a simple Gaussian transition model, our decision making algorithm is robust enough to handle discrepancies between the assumed model and the simulation model used for testing. We showed that the reward function can be used to balance different conflicting objectives and choose an appropriate behavior for the autonomous vehicle.

The scalability study showed some of the limitations of this approach. As the traffic becomes denser, the success rate drops significantly. One reason is that it is not possible to consider more than four other vehicle at a time in the algorithm without exceeding the computation budget of 0.25 s. Another drawback of this formulation is that it does not account for other sensor limiations such as sensor occlusions. The next chapter extends the POMDP formulation to scenarios involving both cars, pedestrians, and occluded areas. The scalability issue will be addressed in Chapter 5 by building upon the formulation from Section 5.2.

4 Learning to Merge in Dense Traffic

In this chapter, we analyze the ability of an RL agent to benefit from interaction between traffic participants in dense merging scenarios. We show that deep reinforcement learning policies can capture interaction patterns when trained in a variety of different scenarios, even if information about the driver behavior is not available. This approach is then combined with a belief updater that explicitly maintains a probability distribution over the driver cooperation levels. Simulation results shows that an RL agent using belief states as input yields better performance than standard RL techniques as well as an online planning solver. In addition, we propose a simple rule-based behavior parameterized by a cooperation level to model the reaction of vehicles on the main lane to the merging vehicle. The work presented in this chapter results from a collaboration with Alireza Nakhaei and Kikuo Fujimura [41].

4.1 Related Work

Merging into very dense traffic situation is a challenging task for autonomous vehicles. Some decision making algorithms are overly conservative and sometimes fail to create a gap in the traffic. Performing a merge maneuver requires reasoning about the reaction of traffic participants to the merging vehicle in a short time. When no gap is present in the traffic, the autonomous vehicle must be proactive in distringuishing driers that are willing to slow down and yield to the merging vehicle.

It has been shown that planning algorithms must consider interaction and joint collision avoidance models to avoid deadlock situations also known as the freezing robot problem [42]. Previous research addressed the issue of interaction-aware planning by combining a probabilistic interaction model with an online planner [35], [43]–[45]. These approaches are very similar to the one presented in Chapter 3. Online planners (such as POMCP) simulate the environment up to a certain horizon and take actions that maximize th expected reward of the corresponding simulated trajectories. As demonstrated in the previous chapter, these methods can scale to large environments and continuous state spaces, but they still suffer from the curse of dimensionality. The computational complexity associated with dense traffic scenarios limits the planning to short time horizons [35], [46] or limit the number of vehicles considered [31], [44], [45].

The performance of the resulting policies is greatly influenced by the underlying model used to represent the environment [47]. Sunberg, Ho, and Kochenderfer showed that a significant improvement in performance can be gained when the planning algorithm has access to information about the driver internal state in lane changing scenarios. Previous work address the problem of modeling interactions between traffic participants using data-driven approaches, probabilistic models, inverse reinforcement learning, rule-based methods, or game theoretic frameworks [43]–[45], [48], [49].

Inverse reinforcement learning techniques and game theoretic frameworks are generally too computationally expensive to be used in an online planning algorithm considering more than two traffic participants [45], [49]. Schmerling, Leung, Vollprecht, *et al.* demonstrated a data-driven approach to learn the interaction model on a traffic weaving scenario involving two agents [44]. They benefit from parallelization to use this model efficiently for online planning. Such an approach is promising but is not suitable for dense traffic scenarios where more than two traffic participants are interacting.

Instead of relying on online planning methods, we propose to learn an efficient navigation strategy offline, using reinforcement learning (RL). RL provides a flexibility in the choice of the interaction model. RL has been applied to a variety of driving scenarios such as lane changing [50], or intersection navigation [34], [51].

The scenario of interest is illustrated in Figure 4.1. Vehicles on the main lane

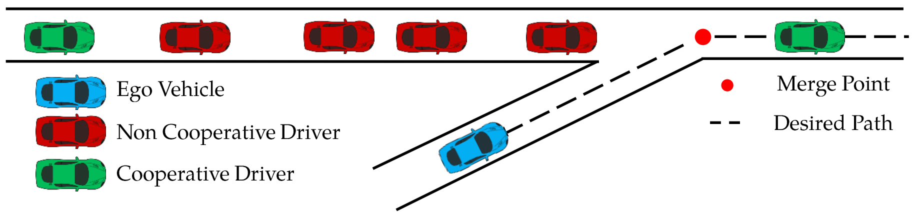


Figure 4.1: Example of a merging scenario in dense traffic. Drivers on the main road have different reactions to the ego vehicle's motion. Their behavior continuously spans from cooperative drivers that yield to the merging vehicle (green cars) or non cooperative drivers that ignore it (red cars).

have priority over the merging car (in blue). We focus on dense traffic situation where cars drive slowly (around $5\,\mathrm{m\,s^{-1}}$) and very close to each other (the gaps can be below $2\,\mathrm{m}$). The ego vehicle on the merging ramp must merge into traffic at a given merge point. When the gap between vehicles is not sufficient, the merging vehicle can only merge if vehicles on the main lane agree to yield.

4.2 *Deep Q-Learning*

We first introduce the solving technique used in this chapter. It is an offline methods that differs from traditional POMDP solvers.

Reinforcement Learning (RL) algorithms provide techniques to solve sequential decision making problems modeled as MDPs with unknown transition and reward models. *Deep Reinforcement Learning* refers to reinforcement learning algorithms that rely on neural representations of the value function, the policy, or the dynamics of the environment. Recent advances in deep RL lead to algorithms that scales to continuous and high dimensional domains [18].

In this thesis, we consider reinforcement learning algorithms as offline methods to solve POMDPs. Instead of being unknown, the transition model and reward function are available through a simulation environment that the agent can interact with. RL generally assumes that the environment is modeled by an MDP and not a POMDP. A common approximation is to approximate the state by the observation or by a finite history of observation. In this thesis we consider a history of k observations (we will use $k = 4$) to approximate the state of the environment: $s_t = (o_{t-k}, o_{t-k+1}, \dots, o_t)$. Such approximation, referred to as k-Markov approximation, assumes that by considering the state as a history of observations, the environment still respects the Markov property in spite of the partial observability [18], [52].

In MDPs with discrete state and action spaces, the value function can often be represented by a table and computed using dynamic programming. When the state space is very large or continuous, it is impossible to explicitly represent the value associated to every possible state. Instead, the value function is represented by a parametric model such as a neural network. We denote $Q(s, a; \theta)$ a state action value function parameterized by θ, referred to as a deep Q-network (DQN). This technique approximates the value of every possible state with a limited number of parameters θ. Note that, s is in practice an observation or sequence of observation. We keep the notation s to be consistent with the RL literature.

Computing the parameters of the network is formulated as an optimization problem. An objective function to minimize, based on Equation (2.2), can be expressed as follows:

$$J(\theta) = \mathbb{E}_{s'}[(s + \gamma \max_{a'} Q(s', a'; \theta_-) - Q(s, a, ; \theta))^2] \tag{4.1}$$

The parameter θ_- defines a fixed target network [18]. The loss function is computed and minimized using sampled experiences. An experience consists of an interaction with the environment over one time step where the agent in state s takes action a, transition to state s', and receives reward r. The action a is selected using a ϵ-greedy strategy [52]. If Q satisfies the Bellman equation, then the loss should be zero. By performing stochastic gradient descent on the object J, we obtain the following update rule given an experience (s, a, r, s'):

$$\theta \leftarrow \theta + \alpha(r + \gamma \max_{a'} Q(s', a'; \theta_-) - Q(s, a; \theta)) \nabla_\theta Q(s, a; \theta) \tag{4.2}$$

where α is the learning rate, a hyperparameter of the algorithm controlling the magnitude of the update. Computing the gradient of Q with respect to θ can be done efficiently using back-propagation when Q is represented by a neural network [53].

In general, Equation (4.2) can lead to unstable updates and the hyperparameters of the algorithm are difficult to tune. Fortunately, several innovations to improve netwok training have been proposed in the literature. The original Q-learning algorithm uses a single Q-network but fixing the parameters in a target network for several opimization steps has been shown to help the convergence of the algorithm [18]. Other innovations involve double DQN [54], dueling network architecture [55], and prioritized experience replay [56]. Our implementation of the deep Q learning algorithms uses those three innovations, and is available at `https://github.com/JuliaPOMDP/DeepQLearning.jl`, it relies on the POMDPs.jl [57] and Flux.jl [58] libraries.

Although this algorithm can handle high-dimensional state spaces, it requires many experience samples to converge (on the order of millions for Atari games [18], [55]). Experience can sometimes be generated using simulators (it is the case in this thesis), but for applications where simulation is expensive or nonexistent, the deep Q-learning algorithm may be impractical. When applying reinforcement learning to accomplish a difficult task, a possibility is to decompose it into simpler subtasks for which you can afford to run algorithms such as deep Q-learning. The policy for the complex task can then be obtained using the utility fusion technique presented in Chapter 5.

4.3 Modeling Merging Scenario

We model the merging scenario as a POMDP according to the framework described in Chapter 2 with the following definitions of the states, observations, actions, reward and transition model.

4.3.1 States

The state consists of the physical state of all the vehicles as described in Chapter 2. Given the configuration of the scenario, position is only represented by one dimension corresponding to the longitudinal position of the vehicles along their lane (merge lane for the ego vehicle and main lane for all the others). The internal state of other driver is represented by a single stationary scalar variable. The signification of this variable will be explained in the later section on driver model.

Each vehicle follow a one dimensional point mass dynamics controlled by a longitudinal acceleration input. Other drivers in the environment follows an interactive driver model described in the next section. In addition, when a vehicle passes the end of the main lane, it is spawned at the beginning of the lane again. This technique allows to maintain a dense traffic in simulation.

4.3.2 *Observations*

The ego vehicle has limited sensing capabilities. It can only sense the vehicles within a certain range, and cannot measure internal states of other vehicles. In this section we do not consider sensor noise. Instead, we focus on the partial observability introduced by the internal state governing the behavior of other drivers. To simplify the observation space, we restrict the observation to the longitudinal position and velocity of four neighbor cars: the front neighbor of the ego vehicle, the vehicle right behind the merge point, the rear neighbor and front neighbor of the projection of the ego vehicle on the main lane. To project the ego vehicle on the main lane, we place it at the same distance to the merge point than its actual position on the merge lane. The ego vehicle can observe the longitudinal position and velocity of these four vehicles perfectly if they are in the field of view. The longitudinal position, speed, and acceleration of the ego car are observed. In addition, we compare the cases where the internal state is directly observable by the ego vehicle or not leading to two or three dimensions per neighbor cars. The total dimension of the observation space is 15 when the internal states are observed and 12 otherwise. We will refer to those two cases as RL with full observation (FO), that can observe the internal state, and standard RL that only observes positions and velocities. Figure 4.2 illustrates the vehicles observed by the ego car. In addition, it can observe its own state (three dimensions). This observation vector is used as input to our RL agent.

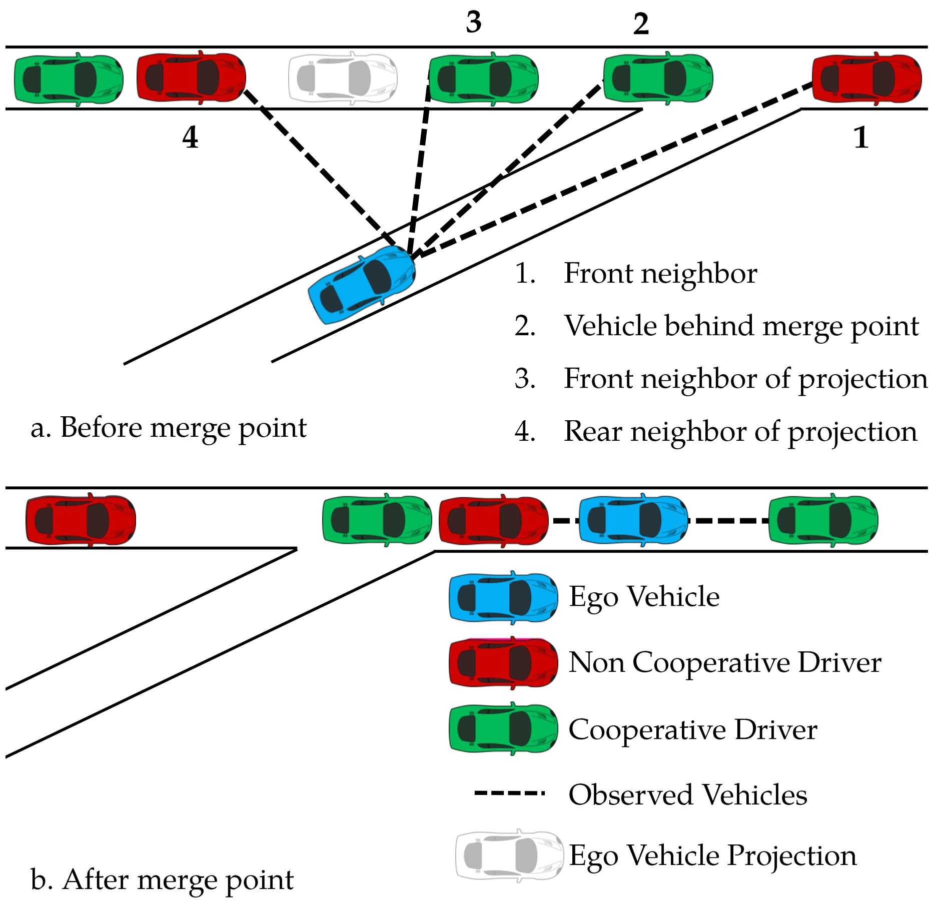


Figure 4.2: Illustration of the vehicles observed by the ego vehicle. The observation vector (or feature vector) contains information on the position and velocity of the observed vehicles. When less than four vehicles are observed, redundant information is present in the feature vector to preserve its dimension. In addition we analyzed cases where the cooperation level of each observed cars is part of the feature vector.

4.3.3 Actions

The ego vehicle controls its motion by applying a change in acceleration. At each time step, the acceleration is updated as follows:

$$a_t = a_{t+1} + \Delta a \tag{4.3}$$

where Δa is the action chosen by the agent in the set:

$$\{-1\,\mathrm{m\,s^{-2}}, -0.5\,\mathrm{m\,s^{-2}}, 0\,\mathrm{m\,s^{-2}}, 0.5\,\mathrm{m\,s^{-2}}, 1\,\mathrm{m\,s^{-2}}\}.$$

The agent can also apply a hard braking action and releasing action which instantaneously set the acceleration to $-4\,\mathrm{m\,s^{-2}}$ and $0\,\mathrm{m\,s^{-2}}$ respectively. Hence, the action space is discrete with 7 possible actions at each time step. The design of the action space is inspired by the work of Hu, Nakhaei, Tomizuka, *et al.* [59].

4.3.4 Reward

The reward function is designed such that the optimal policy maximizes safety and efficiency. The agent receives a penalty of −1 for colliding with other traffic participants and receives a bonuses of 1 for reaching a goal positions defined 50 m after the merge point. The time minimization is incentivized by the discount factor, the sooner the goal is reached, the less the bonus is discounted. We used a discount factor of 0.95.

4.4 Driver Model

To model the behavior of drivers on the main lane, we propose an extension of the Intelligent Driver Model (IDM) [28]. Our model controls the longitudinal acceleration of the vehicle on the main lane while taking into account merging vehicles. In addition to the IDM parameters, we introduce a cooperation level $c \in [0, 1]$ which is a scalar parameter controlling the reaction to the merging vehicle state. $c = 1$ represents a driver who slows down to yield to the merging vehicle if she predicts that the merging vehicle will arrive ahead of time. $c = 0$ represents a driver who completely ignores the merging vehicle until it traverses the merge point and follows standard IDM. We will refer to this model as Cooperative Intelligent Driver Model (C-IDM).

C-IDM relies on estimating the time to reach the merge point (TTM) for the car on the main lane (TTM_a) and the car on the merge lane (TTM_b) to decide whether the merging vehicles should be considered or not. Once the time to merge for both vehicles is estimated, three cases are considered:

- If $TTM_b < c \times TTM_a$, the vehicle on the main lane follows IDM by considering the projection of the merging vehicle on the main lane as its front car.
- If $TTM_b \geq c \times TTM_a$, the vehicle on the main lane follows standard IDM.
- In the absence of a merging vehicle or far from the merging vehicle, the C-IDM driver follows the standard IDM.

Although this model might not represent precisely how human drivers behave, it provides us with a broad range of behaviors by adjusting the cooperation level. In this work we used a simple constant velocity model to estimate the time to merge. A more sophisticated prediction model can be used to have more realistic estimates of the TTM. Given the cooperation level, the driver has a deterministic behavior. Figure 4.3 illustrates the situation where a cooperative vehicle takes into account the merging vehicle.

4.5 *Inferring the Cooperation Level*

Since the cooperation level cannot be directly measured, the ego car maintains a belief on the cooperation level of the observed drivers. At each time step, the belief is updated using a recursive Bayesian filter given the current observation of the environment.

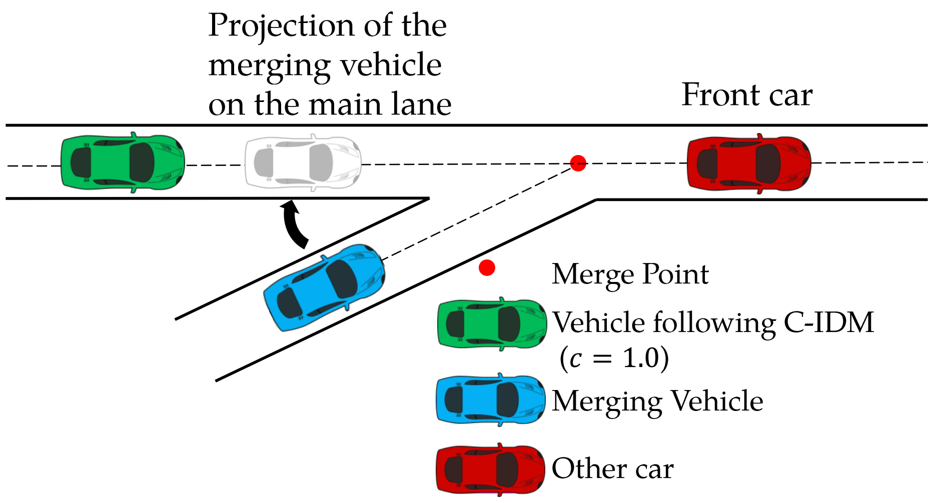

Projection of the merging vehicle on the main lane
Front car
Merge Point
Vehicle following C-IDM ($c = 1.0$)
Merging Vehicle
Other car

Figure 4.3: Illustration of the C-IDM model where a cooperative vehicle (in green) considers the merging vehicle as its front car instead of the red vehicle which is the front neighbor used by standard IDM.

In this problem, the position and velocity of other drivers is assumed to be fully observable whereas the cooperation level is not always observed. This assumption of mixed observability can help us reduce the computational cost of the belief update. This assumption of mixed observability can help us reduce the computational cost of the belief update. The agent only maintains a distribution over the cooperation level of observed drivers instead of estimating the full state of the environment. Our simple belief updater, is acting as if the cooperation level was binary although it can take any value between 0 and 1. The belief at time t is composed of the fully observable part of the state, o_t, and θ_i for $i = 1 \ldots n$, where n is the number of observed drivers. θ_t^i represents the probability that vehicle i has a cooperation level of 1. At time $t+1$, the ego vehicle observes o_{t+1} and updates its belief on the cooperation level of vehicle i as follows:

$$\theta_{t+1}^i = \frac{\Pr(o_{t+1} \mid o_t, c_i = 1)\theta_t^i}{\Pr(o_{t+1} \mid o_t, c_i = 1)\theta_t^i + \Pr(o_{t+1} \mid o_t, c_i = 0)(1 - \theta_t^i)} \tag{4.4}$$

Equation (4.4) can be computed by simulating forward the previous state with the two possible hypotheses: $c_i = 1$ and $c_i = 0$ and comparing the outcome with the current observation. The probability of transitioning from o_t to o_{t+1} given the cooperation level, is computed by propagating the state forward using the proposed transition model and assuming a Gaussian distribution centered around the predicted value with a standard deviation of 1 m for the positions and 1 m s^{-1} for velocities. Without this addition of noise in the transition model, the belief would converge after one observation to $\theta^i = 0$ or $\theta^i = 1$ since the two models would be perfectly distinguishable.

The focus of this work is not to develop an efficient driver state predictor but rather discuss how this information can be used by an RL agent. Future work might consider more complex filtering techniques such as multi-hypothesis filters, interacting multiple models, or data-driven approaches to estimate the driver cooperation level from observation [60]. An additional extension is to consider continuous values of the cooperation level since it is supported by the C-IDM model.

4.6 Training Procedure

This section highlights some important aspects of our implementation: the belief state RL training procedure, and the distribution of scenarios used during training. Further details can be found in our code base: `https://github.com/sisl/AutonomousMerging.jl`.

4.6.1 Belief state reinforcement learning

Standard reinforcement learning techniques assume that the underlying environment is an MDP. Latent states such as driver behavior characteristics are not explicitly inferred during training. Although memoryless policies can be efficient, reasoning about latent states can often lead to a significant improvement [47]. In the merging scenario, knowing which drivers are more cooperative can help the agent take better decisions. It is expected that the ego vehicle will only try to merge in front of cooperative drivers. In order to learn such a behavior through reinforcement learning, we propose to use the belief state as input to the reinforcement learning policy. The resulting algorithm is very similar to the standard DQN algorithm applied to the belief MDP. A transition in the belief state MDP can be described as follows:

- At time step t, the agent has a belief b_{t-1} and receives an observation o_t
- The new belief b_t is computed using Equation (4.4)
- The agent takes action $a_t = \arg\max_a Q(b_t, a)$

The rest of the algorithm is identical to standard DQN.

The input to the network is a vector of dimension 15:

$$b_t = [o_t, \theta_t^1, \ldots, \theta_t^n]$$

where n is the number of observed vehicle, and o_t is the fully observable part of the state (information on position and velocity), and θ_t^i is the probability of driver i being cooperative. By feeding in the probability on the internal state, the agent can reason in the belief space rather than in the observation space. The resulting Q network is combined with a belief updater at test time to result in a policy robust to partial observability.

4.6.2 Curriculum learning

We used a curriculum learning approach to train the agent by gradually increasing the traffic density. When training an RL agent in dense traffic directly, the policy converges to a suboptimal solution which is to stay still in the merge lane and does not leverage the cooperativeness of other drivers. Such a policy avoids collisions but fails at achieving the maneuver. To encourage exploratory actions, we first train an agent in an environment with sparser traffic (5 to 12 cars). An alternative to curriculum learning is to design the reward function to incentivize the learning agent to move forward at each time step. However, a more complex reward function often requires a lot of parameter tuning. We found the curriculum learning approach more practical in this work.

Table 4.1: Deep Q-learning parameters

Neural network architecture	2 dense layers of 64 and 32 nodes
Training steps	3×10^6
Activation functions	Rectified linear units
Replay buffer size	400 k experience samples
Target network update frequency	5 k episodes
Discount factor	0.95
Optimizer	Adam
Learning rate	1×10^{-4}
Prioritized replay [56]	$\alpha = 0.7$, $\beta = 1 \times 10^{-3}$
Exploration strategy	ϵ greedy
Exploration fraction	0.5
Final ϵ	0.01

The parameters of the DQN algorithm are summarized in Table 4.1. It was implemented using the Flux.jl library [58]. Training one policy took around 40 minutes on three million examples.

4.6.3 *Initial state distribution*

To populate the initial state of the merging scenarios, we used a two step procedure. The first step consists of sampling the number of vehicles present from a desired range. We considered two ranges corresponding to different traffic conditions:

- Mixed traffic: between 5 and 12 cars on the main lane. The agent will experience both sparse traffic scenarios and dense traffic scenarios.

- Dense traffic: between 10 and 14 cars on the main lane.

The main lane has a length of 150 m and the vehicles have a length of 4 m. In dense traffic situations, the gap between vehicles varies from less than 2 m to larger distances. Once the number of cars is decided, vehicles are randomly positioned on the main lane. The initial velocity of each vehicle is drawn from a Gaussian distribution of mean $5\,\mathrm{m\,s^{-1}}$ and a standard deviation of $1\,\mathrm{m\,s^{-1}}$. Finally, the desired velocity of each vehicle is drawn uniformly from the set: $\{4\,\mathrm{m\,s^{-1}}, 5\,\mathrm{m\,s^{-1}}, 6\,\mathrm{m\,s^{-1}}\}$

and their cooperation level is drawn uniformly in the interval $[0, 1]$. The desired velocity is used to parameterize the IDM part of C-IDM.

The second step of the initialization consists of simulating the initial state for a burn-in time randomly chosen between 10 s and 20 s. During this burn-in time, the ego vehicle is not present and the vehicles on the main lane follow the C-IDM. The burn-in time allows the initial state to converge to a more realistic situation. This approach is loosely inspired by the work of Wheeler, Kochenderfer, and Robbel [61].

Such procedure ensures that the learning agent experiences a variety of situations during training. The design of the training scenarios is critical to the performance of the RL agent and will determine the ability of the policy to generalize. The more variety it experiences during training, the more the policy can generalize.

4.7 *Experiments*

To evaluate the ability of our method to use the cooperativeness of other drivers, we compared the performance of different policies in the merging scenario with dense traffic (between 10 and 14 cars on the main lane). Each policy is evaluated in 1,000 scenarios with initial states sampled as described in Section 4.6.3. We measured the percentage of scenarios that ended in collisions (collision rate), the average time to reach the goal position located 50 m after the merge point, as well as the number of scenarios ending in time-out failure. A time-out failure is declared if the ego vehicle has not reached the goal position after 50 s. Such failure cases are representative of the robot freezing problem [42].

We compare the following methods:

RL without cooperation level information: The first policy consists of using a standard reinforcement learning algorithm which can observe position and velocity of the other vehicle but not their cooperation level. It is referred to as RL.

RL with cooperation level information: This policy is trained using standard reinforcement learning but has access to the cooperation level at training and test time. It is referred to as RL (FO).

Belief state RL: This policy is trained in the belief MDP. The input of the policy is the prediction given by the belief updater. It is referred to as belief RL.

MCTS without cooperation level information: This policy uses the same algorithm as the previous policy but does not have access to information about the cooperation level. Instead, it makes an assumption on the driver cooperation level. We evaluated three different assumptions:

- $c = 1$: assumes drivers are always cooperative.
- $c = 0.5$: assumes a cooperation level of 0.5, drivers on the main lane will react to the merging vehicle only if it will reach the merge point in twice less time. This behavior is a middle ground between non cooperative and cooperative drivers.
- $c = 0$: assumes drivers are never cooperative. It is equivalent to assuming that the drivers on the main lane follows IDM and are blind to the merging vehicle.

MCTS with cooperation level information: This policy is using Monte Carlo tree search. To handle the continuous state space we used double progressive widening [26]. This online method is a fully observable version of the POMCP algorithm described in Section 2.4. We used a computation budget matching the decision frequency of 0.5 s. The exact model of the environment is used to produce the tree, and the root node contains all the information about the cooperation levels. It is referred to as MCTS (FO).

The first three approaches are offline deep reinforcement learning algorithms where as the last two are online planning methods.

An example of behavior learned through RL is illustrated in Figure 4.4. The ego vehicle first slows down as it approaches a dense traffic. Once a cooperative driver is detected ($t = 13$ s) the vehicle slowly merges. At $t = 18$ s the ego vehicle has merged and follows the traffic on the main lane.

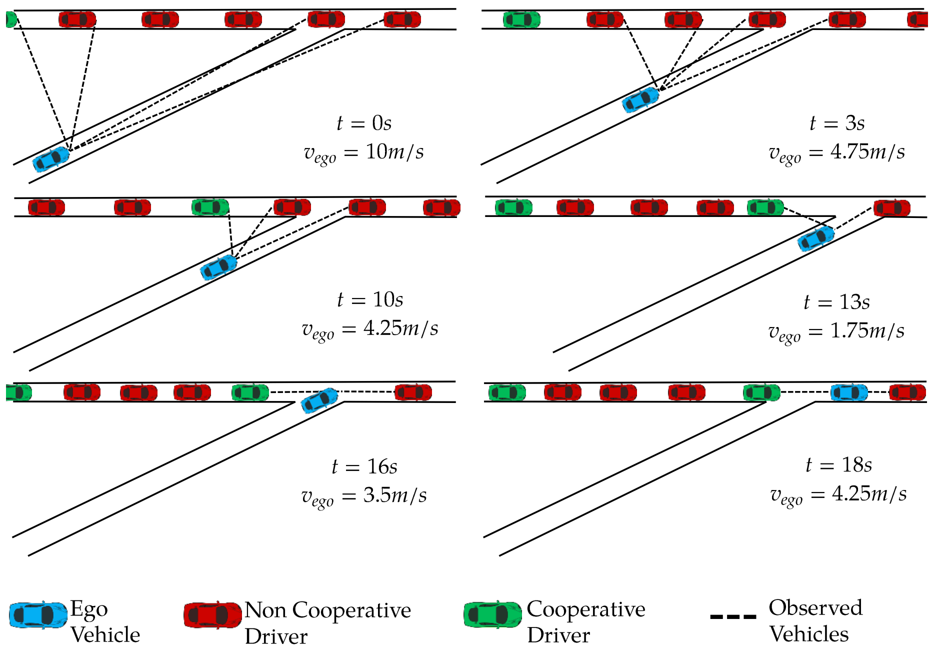


Figure 4.4: Example of a trajectory when executing the reinforcement learning policy in dense traffic. The ego vehicle learns to merge in front of cooperative drivers.

Figure 4.5 illustrates the performance of the evaluate policy on a dense traffic scenario. We can see from the percentage of collisions that MCTS (FO) is the safest policy, followed by the belief RL approach. The three other MCTS policies had a lot of collisions (much larger than 1 %). The RL policy without access to information on the cooperation level had 2 % collisions at test time and the two other RL policies performed similarly with around 0.6 %collisions. Previous works has shown that only relying on deep RL is not sufficient to achieve safety [62], [63] (this aspect is explained in more detailed in Chapter 7). The deployment of those policies would require the addition of a safety mechanism. It is important to notice that even though they did not have access to the full state, the RL and belief RL policy have much better safety performance than the MCTS approaches that did not have access to this information either.

Regarding the number of time steps to cross, we notice that the MCTS policy with $c = 1$ is the most efficient. The policy is biased towards taking more aggressive

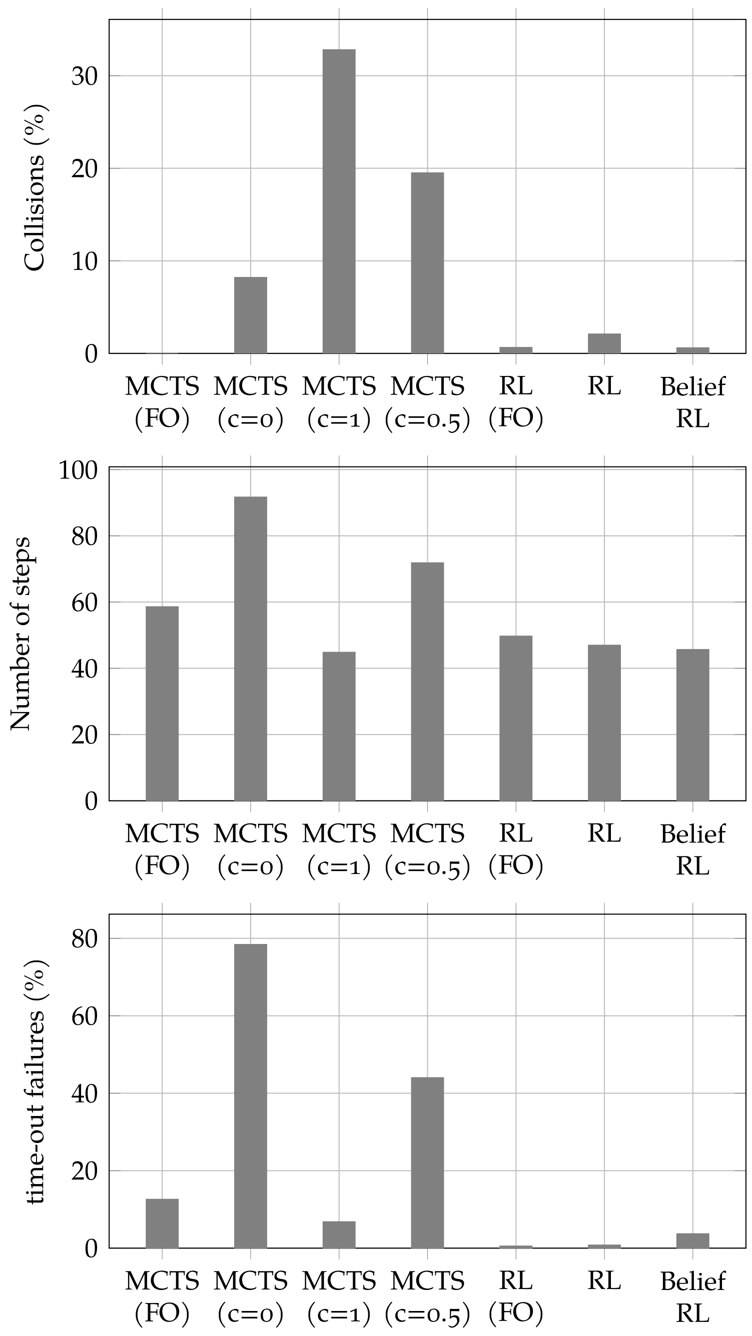


Figure 4.5: Performance of the different policies on a dense traffic scenario. Each policy is evaluated on 1,000 simulations. The policy MCTS (FO) did not result in any collisions. A step in the environment corresponds to 0.5 s. The number of steps corresponds to the average time to reach the goal position.

actions since it is assuming that every driver is cooperative. On the contrary, the MCTS policy assuming that no driver is cooperative ($c = 0$) has a very conservative behavior: it takes the longest time and has the largest number of time-out failures. As expected MCTS ($c = 0.5$) has a performance in between the previous two. The RL policies are more efficient than the MCTS (FO) policy and have an average time to cross close to the most aggressive MCTS policy. In addition, we can see that the RL policies have much fewer time-out failures than the MCTS policies. This last fact illustrates that they were able to successfully infer and leverage the information on the cooperation level.

One can notice that the gap in performance between MCTS (FO) and the other MCTS is much larger than the gap between RL and RL (FO). A possible explanation is that the neural network approximation is able to capture the cooperation level inference task in the hidden layers. However, this implicit state estimation is less efficient than the explicit state estimation provided by combining the belief updater with the RL policy during training and execution. The belief RL policy has a similar safety level and takes, on average, a similar number of steps than the RL policy with perfect observation.

MCTS with full observation still presents a significant number of time-out failures. We believe that relaxing the computation constraint would have lead to better performance. Computation is generally the major bottleneck of online planning algorithms. Our experiments show that the performance can be closely matched using offline trained policies which take a very short time to execute online. However, the MCTS policy with full observation is safer than the deep RL equivalent. A direction for future work is to use the RL policy as a value estimator to guide the search in MCTS.

4.8 Discussion

In this chapter we presented a reinforcement learning approach to the problem of autonomously merging in desne traffic. Our study confirms that an autonomous agen can benefit from planning in the belief space rather than direclty from observation. In this scenario, the belief represents the estimated level of cooperation of other drivers. We presented a belief state RL procedure that explicitly tries to estimate the internal state of other drivers The POMDP formulation allows to reason about interaction with other traffic participants and leads to more efficient policies. We have also shown that an agent trained using deep reinforcement learning can outperform online planning methods when being exposed to a broad range of driver behaviors during training. Finally, we proposed C-IDM, a simple parametric model capturing a variety of cooperative behaviors.

Although our RL agent learned more efficient policies, an online planner may provide greater robustness and easier generalization to other scenarios (no need to retrain). For more complex tasks, the deep reinforcement learning algorithm is unlikely to learn good policies without an extensive reward engineering process [29]. Moreover, it is hard to enforce safety constraints: as we see from the results, even with full knowledge, the RL agent still has some collision. Instead of brute forcing scenarios using expensive tree search or function approximation technique, we propose in the next chapter a decomposition method to efficiently solve driving scenarios. Chapter 6 will then demonstrate how deep reinforcement learning algorithms can be used in conjonction of this decomposition method.

5 Utility Decomposition

In this chapter, we present a scalable method for approximately solving POMDPs for avoiding multiple occluded traffic participants through utility decomposition. We focus on an unsignalized crosswalk and an unsignalized intersection with traffic in both directions. Those scenarios are modeled using the framework proposed in Chapter 2. Both scenarios have areas that are not observable by the car due to an obstacle on the side of the road as illustrated in Figure 5.1. Providing appropriate behaviors requires scalable algorithms that reason about the potential locations of pedestrians and other vehicles along with their motion over time. The proposed decomposition method consists of computing the optimal state action utility for each user in the environment independently of the others and fusing these utilities to approximate the global problem of avoiding multiple users. We show that this technique lead to performant policies. This work has been done in collaboration with Alireza Nakhaei, Kikuo Fujimura, Markus Schratter and Daniel Watzenig [64], [65].

5.1 Related Work

Prior approaches for accounting for sensor occlusions involved resolving the worst case scenario. By specifying an upper bound on the speed of the agents potentially coming from the occluded area, one can provide a speed profile that will guarantee safety [66]. Other techniques to navigate in occluded areas use set-based methods [67]–[69]. Such methods are often well suited to provide safe velocity profiles. However, they rely on over approximations to account for uncertainty. In

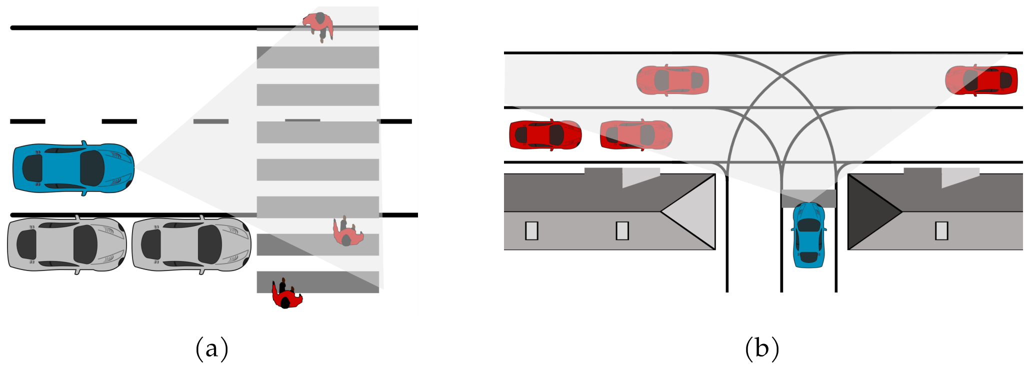


Figure 5.1: Examples of two occluded environments. In (a), the ego vehicle must go through the crosswalk while avoiding pedestrians. In **??** the vehicle must achieve a left turn while avoiding other cars. In both situation, the other traffic participants might not be visible.

cases where the uncertainty is high, a worst-case approach might lead to suboptimal behaviors such as standing still if the car fails to find a feasible strategy to avoid collisions [42]. Another approach is to set a threshold on the level of uncertainty acceptable to make safe decisions [70]. Providing such tailored solutions is unlikely to scale to complex tasks or generalized to different scenarios. Instead, the POMDP formulation provides a principled way to trade off between gathering information and achieving the task.

The objective of this chapter is to demonstrate the application of our POMDP approach to autonomous driving scenarios with sensor occlusions. We first demonstrate how to solve the problem of avoiding a single traffic participant using offline methods. We describe the POMDP Model and analyze two different offline POMDP solvers: QMDP [22] and SARSOP [17]. The use of an offline method will allow us to leverage the decomposition method from **??** and generalize the solution to avoiding multiple traffic participants.

There are many domains where an autonomous agent must interact with multiple entities. In autonomous driving, other entities may include vehicles and pedestrians that must be avoided. Decision making algorithms can have difficulty scaling to such domains because the state space grows exponentially with the number of entities. One way to approximate solutions to problems with multiple entities is to decompose the state space into pairwise interactions and learn the utility function with each entity individually. The solutions to the individual subproblems can be combined online through an arbitrator that maximizes the expected utility [71]. Applications to aircraft collision avoidance with multiple intruders explore summing or using the minimum state-action values [72], [73]. The solution to each subproblem assumes that its individual policy will be followed regardless of other entities, in contrast to the global policy considering all subproblems [74].

Previous approaches to scale decision algorithms using decomposition methods relied on distributed agent architectures [71]. Each agent is responsible for addressing one of the multiple objectives required to achieve a complex task. At each time step, an arbitrator must decide between the differenta ctions recommended by these agents and address possible conflicts. Possible arbitration strategies are command fusion [72], voting [71], lexicographic ordering [75] or utility fusion [71], [74]. Wray, Witwicki, and Zilberstein suggest a decomposition of complex driving scenario into several POMDPs [75]. Each instantiated problem is solved offline. An arbitrator chooses online between the actions given by the different policies. The arbitration process relies on a simple ordering of the actions and does not take into account the utility of the individual problems. It has been shown that utility fusion offers a more principled way of deciding between the individual agents compared to a voting-based approach or command fusion [71].

Utility decomposition methods have been used in many practical applications in adversarial settings. Chryssanthacopoulos and Kochenderfer applied utility decomposition to a collision avoidance problem where an aircraft must avoid multiple other aircraft [72]. A similar approach is proposed for avoiding potential debris in the airspace [76]. In settings where the reward function can be additively decomposed, an arbtrator maximizing the sum of the individual utilies can be optimal under certain conditions [74].

5.2 Avoiding One Road User

In this section we describe how to discretize the POMDP model from Chapter 2. We illustrate the flexibility of the model by applying the same formulation to a pedestrian avoidance scenario as well as an intersection navigation scenario.

5.2.1 Scenario Modeling

All the element of the POMDP tuple needs to be defined: state space, transition model, action space, observation space, observation model, and reward function.

State space and transition model

To solve those scenarios using offline POMDP solvers we first need to discretize the state space and define an action space. The action space has been described in Chapter 2. Since offline methods only scales to tens of thousands of state, we limit our focus to situations with only one other traffic participant to avoid. In addition, we assume that the motion of the pedestrians and other vehicles are constrained along a lane. That is pedestrians can only go straight on the crosswalk, and cars move longitudinally along a pre-determined route. These assumptions allow to reduce the state representation to $(s_{\text{ego}}, v_{\text{ego}}, s_{\text{other}}, v_{\text{other}}, \beta_{\text{other}})$ where s_{ego} is the longitudinal position of the ego vehicle along its path (left-turn), v_{ego} the longitudinal velocity, s_{other} the longitudinal position of the other entity, v_{other} its longitudinal velocity, and β_{other} the internal state used to indicate the route of the vehicle in the intersection (*e.g.* right turn or keep straight). In the crosswalk scenario, the pedestrian can have only one path.

The pedestrian behavior model is to change its velocity by a random amount in the set: $\{-1\,\mathrm{m\,s^{-1}}, 0\,\mathrm{m\,s^{-1}}, 1\,\mathrm{m\,s^{-1}}\}$, and its dynamics model is a deterministic point-mass constant velocity model. The motion model of the pedestrian does not depend on the state of the ego vehicle. It is designed such that it spans a large number of possible trajectories. It is expected that the ego vehicle learns a conservative policy when interacting with this environment. Many of the pedestrian trajectories generated using the proposed motion model would not occur in the real world. A more complex modeling of pedestrian behavior is left as future work. Previous work presents data driven approaches for modeling pedestrian behavior at crosswalk [77].

The velocity of the pedestrian is bounded up to $2\,\mathrm{m\,s^{-1}}$. In the crosswalk scenarios position variables are discretized with a resolution of $1\,\mathrm{m}$ and velocity variables with a resolution $1\,\mathrm{m\,s^{-1}}$. An additional pedestrian pose is added to model the case where the pedestrian is absent from the scene: s_{absent}.

The discretization results in thirty three possible positions and eight velocities for the ego car and eleven positions and three possible velocities for the pedestrian. The combination of all positions and velocities leads to 1.01×10^4 states.

In the intersection scenario, the other vehicle behavior is to change its acceleration by a random amount in the set $\{-0.5\,\mathrm{m\,s^{-2}}, 0\,\mathrm{m\,s^{-2}}, 0.5\,\mathrm{m\,s^{-2}}\}$ and follows its path given by β_{other} according to a constant acceleration dynamics. The position and velocity spaces are again discretized but with a larger resolution of $2\,\mathrm{m}$ and $2\,\mathrm{m\,s^{-1}}$ respectively because greater speeds are involved at intersections. Again, an extra state for modeling an absent car is added. The combination of ego car poses and other car poses and intention leads to 2.15×10^4 states for this scenario.

Observation model

The observation space is similar to the state space except that all states were the other entity is occluded are aggregated in the same observation $(s_{\text{ego}}, v_{\text{ego}}, s_{\text{absent}})$. The observation model can be described as follows:

- An entity in an non-occluded area will always be detected
- If a user is in an occluded area it will not be detected
- The measured position and velocity of a detected user are uniformly distributed around the true state at $\pm 1\,\mathrm{m}$, $\pm 1\,\mathrm{m\,s^{-1}}$ for the crosswalk and $\pm 2\,\mathrm{m}$, $\pm 2\,\mathrm{m\,s^{-1}}$ for the intersection.

To check if an entity is occluded we draw a segment between the front of the ego vehicle and the center of the entity to observe and check if the segment intersect with an obstacle. Intersection between a segment and a convex shaped obstacle in a two dimensional space can be performed efficiently using the separating axis theorem [78].

For these two problems, we assume that the path of the other car, β_{other}, is observable. The focus of this case study is to analyze the performance of the POMDP approach regarding occlusion handling rather than uncertainty in the intentions of other drivers. This aspect has been addressed in previous work [14], [15], [31], [79], [80] and in the previous chapter to some extent.

Reward model

The ego vehicle receives a unit reward for reaching a final position and receives a penalty for collision. The final position is 10 m after the crosswalk and 30 m after the intersection center point for the intersection scenario. The value of the collision penalty is tuned through simulation to balance risk averse behaviors and efficiency.

5.2.2 *Solving for belief state policies*

We solve the discrete state and action POMDP using two offline methods: QMDP [22] and SARSOP [17]. The source code for the POMDP model is available at `https://github.com/sisl/AutomotivePOMDPs.jl`. To analyze the quality of the policy, we visualize it on a two-dimensional slice of the state space. Results are presented for the crosswalk scenario only because the geometry of the problem makes the visualization of the policy easier. A quantitative evaluation of this method and comparison with a baseline for both the intersection and the crosswalk scenario is provided in the next chapter.

Each plot represents the actions taken by the vehicle for any longitudinal position of the ego vehicle and longitudinal position of the pedestrian along the crosswalk. The remaining two state variables are fixed: the ego car velocity is fixed at $5\,\text{m}\,\text{s}^{-1}$ and the pedestrian velocity at $1\,\text{m}\,\text{s}^{-1}$. When using the policy, the ego vehicle only has a noisy estimate of the pedestrian position and velocity. To analyze the robustness of the policy to state uncertainty, we assumed that the belief is represented by a Gaussian distribution centered around the true state. Depending on the performance of the perception system, the standard deviation of the belief may be large.

Figure 5.2 shows the difference in the policy when this standard deviation increases. As the state uncertainty increases, we can see that the QMDP policy changes shape until the red zone spreads vertically to almost every pedestrian positions. This figures shows that regardless of the location of the pedestrian, the vehicle brakes hard when reaching the position 15 m. The QMDP policy becomes less efficient. Interestingly, the SARSOP policy barely changes until the standard deviation exceeds 1 m, which roughly corresponds to the level of uncertainty used in the observation model. When the standard deviation increases to 2 m the red zone starts spreading vertically but in a less drastic way than for QMDP. SARSOP performs planning in the belief space, and is aware that the uncertainty on the state will decrease over time as the agent will receive more observation. As a consequence it can take less conservative action. Using a belief state planner such as SARSOP can result in more robust policies and avoid suboptimal behaviors when uncertainty is high.

We demonstrated how to model scenarios with sensor occlusion using a POMDP. If we limit the scenario to involve only the ego vehicle and one other traffic participant, the state space can be discretized and a policy can be computed using offline methods. We have shown that depending on the algorithm used, the policy can be more robust to uncertainty.

Following the generic model presented in Chapter 2, we could extend the POMDP formulation to avoiding multiple users by extending the state space. Adding an additional participant requires introducing three new variables to the state space (longitudinal position, velocity and internal state). If we keep the same grid resolution, the number of states would grow from 1.01×10^4 to 2×10^5 states for the crosswalk scenario and from 2.15×10^4 to 7×10^6 for the intersection scenario. Both problems would become intractable for SARSOP [17]. Because of the grid based representation, the size o the state space grows exponentially with the number of entity to avoid.

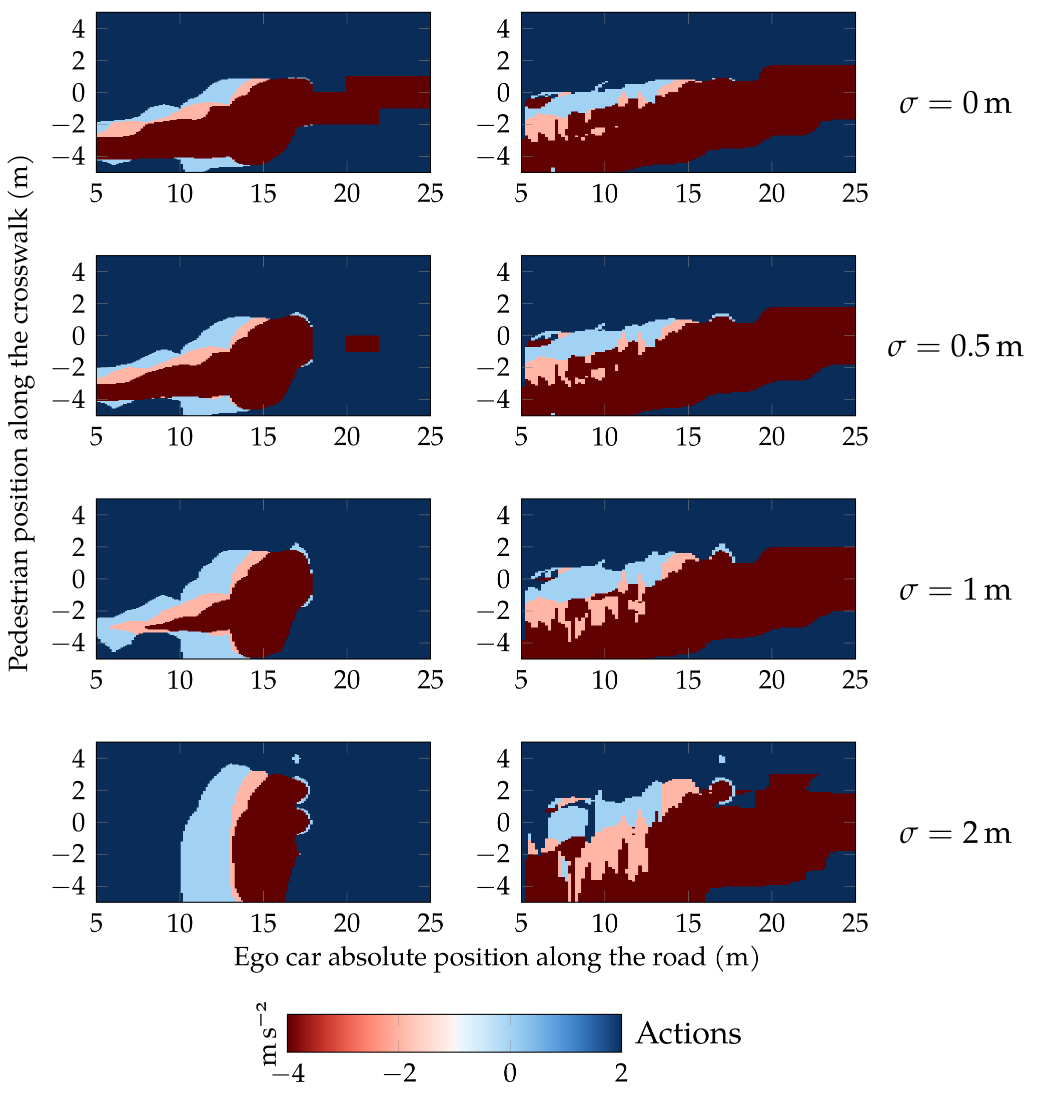


Figure 5.2: Visualization of the policies obtained using QMDP (left) and SARSOP (right) with increasing uncertainty on the state. These plots assume a fixed ego velocity at $5.0\,\mathrm{m\,s^{-1}}$ and fixed pedestrian velocity at $1.0\,\mathrm{m\,s^{-1}}$. The belief is assumed to be a Gaussian distribution over the position of the pedestrian with standard deviation σ.

Online methods presented in Chapter 3 cannot handle more than four traffic participants without exceeding the computation budget. Offline methods presented in this chapter are limited to considering only one other road user. In the next chapter, we present a decomposition method that can help approximate solutions to this problem and scales to many traffic participants.

5.3 *Utility Fusion*

Utility decomposition, sometimes referred to as Q-decomposition or utility fusion, involves combining the state action value functions of simple decision making tasks to solve a more complex problem. We refer to the complex task as the global problem and the subtasks as local problems. It is often the case that the state space of the local problems is a subset of the state space of the global problem. Each local problem i is formulated as a POMDP and is first solved in isolation. The function Q_i represents the optimal belief-action utility function associated to subtask i. In cases like avoiding multiple entities, these local problems correspond to pairwise interactions between the decision agent and one of the entities to act agains. Solving the global task is then achieved by fusing the utilities associated with each local task. More formally, utility fusiom requires defining a function f such that the global belief-action utility function is approximated as follows:

$$Q(b, a) \approx f(Q_i(b_i, a), \ldots, Q_n(b_n, a)) \tag{5.1}$$

where Q is the optimal value function associated with the global task, n the number of subtask and b_i the beliefs associated to the state of each subtask. The global belief is also decomposed, we can assumed that each of the value functions Q_i uses a subset of the information contained in b to solve the simpler subtask. One approach to this decomposition involves decomposing the reward function additively [74]. Each term of the sum is then optimized individually by a sub-agent. To solve the global problem they choos f to be the sum of the individual value functions. The simplicity of the approach makes it a very appealing techniques to find an approximately optimal solution to a complex decision making problem. In many problems involving non-cooperative multiple agents, utility fusion can help to scale the solution.

In autonomous driving scenario, a possible decomposition is to consider each traffic participant individually. For each entity i in the environment, the ego vehicle maintains a belief b_i on its state and computes the optimal belief action utility $Q_i(b_i, a)$ assuming i is the only user to avoid. Q_i can be computed using an offline POMDP solver as demonstrated in the previous chapter. Q_i measures the expected accumulated reward of taking action a and then following the optimal policy for avoiding entity i considered independently of the others. Solving for pairwise interactions rather than the global problem requires exponentially fewer computations since the size of the state space grows exponentially with the number of traffic participants. The gobal utility is computed by summing the individual value function or by taking the minimum over each entities. Summing the utility functions as follows:

$$Q(b, a) \approx \sum_i Q_i(b_i, a) \tag{5.2}$$

implies that all entities to avoid are independent. Equation (5.2) equally weighs the utility of a road user in a safe zone of the environment and a user in a more dangerous zone. Instead, another strategy is to take the minimum as follows:

$$Q(b, a) = \min_i Q_i(b_i, a) \tag{5.3}$$

This approach is more risk adverse. In Equation (5.3), taking the minimum will consider the user with the lowest utility. Given a reward function penalizing for collisions, it will consider the entity that action a is most likely to harm. Equation (5.2) and Equation (5.3) will be referred to as the max-sum and max-min approaches, respecitvely, because the policy π to execute is obtained as follows:

$$\pi(b) = \arg\max_a f(Q_1(b_1, a), \ldots, Q_n(b_n, a)) \tag{5.4}$$

One issue raised by the decomposition method is that it requires knowing the number of users present in the environment. Since the ego vehicle does not know how many traffic participants are in the occluded areas, it keeps one common belief for all invisible entities. All the entities that are absent or in an occluded area are treated as the same in the decomposition method. The underlying motivation for this approach is that the maneuver to avoid several occluded road users should not be different from avoiding a single occluded user.

Finally, it can be deduced from Equation (5.1) that the computational cost of querying the multi-entity policy online grows linearly with the number of agents. In order to compute $Q(b, a)$, one must compute the utilities associated with each detected entities and the utility associated with invisible entities. Querying the belief action utility for a single user policy can be achieved in the order of millisecond if this utility has been computed with QMDP and a tenth of a second for SARSOP. As a consequence, decomposition methods allow to handle a very large number of entities to avoid.

5.4 *Experiments*

In urban driving scenarios, decomposition methods can leverage the optimal decision strategy of avoiding a single entity to generate a policy that can avoids multiple traffic participants. The previous chapter provides techniques to compute such strategy. The choice of offline solver is very appealing because the query of the utility function online is very fast. As a consequence, the QMDP policy and SARSOP policy computed in the previous chapter are candidate of choice to serve as the sub-task in our utility fusion approach. We evaluate our decomposition method in the same crosswalk and intersection environment as in the previous chapter. All policies are evaluated according to two metrics: average number of collision (safety), average time to reach the goal position (efficiency).

To evaluate the performance of the policies, we compare them in simulation with rule-based policies. For the crosswalk scenario, the policy involves coming to a full stop at the level of the crosswalk (where there is full visibility), check at each time step if a pedestrian is crossing. Once the croswalk is clear, the ego vehicle accelerates.

This strategy corresponds to a wors-case approach where we assume that there will always be a pedestrian right at the edge of the obstacle. A similar strategy is designed for the intersection scenario: stop in the zone of full visibility and then execute a time to collision policy as described in Section 3.3.2. For both scenarios, once the car takes the decision to cross, it keeps accelerating until it reaches the goal position.

In the POMDP formulation, we can tune the reward function to favor one objective over another. Increasing the collision cost will favor risk averse behavior, while decreasing it will lead to a more aggressive driving. To have a fair comparison with the baseline, we first evaluate the POMDP policies from QMDP and SARSOP alone with different values for the collision cost in the reward function. We analyzed the choice of the function used for fusing the individual utilities. Once suitable reward parameters and a suitable fusion function are found, we run 1,000 Monte Carlo simulations and measures the number of collisions and time to cross.

To assess the robustness of our strategy, the environment used for evaluation is different (higher fidelity) from the model used in the POMDP formulation. The source code for the simulation environment is available at `https://github.com/sisl/AutomotivePOMDPs.jl`. In the evaluation environment, a flow of pedestrians or cars is generated and controlled by a probability of appearance (set to 0.01) at every time step. This value lead to three pedestrians or cars present in the environment per scenario in average. The pedestrian are following a constant speed of $1\,\mathrm{m\,s^{-1}}$, and the cars are following the intelligent driver model [28]. At each simulation step, the ego car takes a measurement of the scene. It is assumed that all the non-occluded entities can be identified and tracked independently. The simulated sensor gives a noisy measurement of the position and velocity of other users with Gaussian noise specified in Table 5.1. The ego vehicle can access its own physical state exactly.

As shown in Figure 5.3, computing the action scales linearly with the number of traffic participants observed. We can notice that the belief update operation scales linearly as well and is almost fifty times more expensive than computing the action. Updating the belief requires applying Equation (2.3) for every traffic participants. For SARSOP, since the policy leads to a larger number of α-vectors, computing the

Table 5.1: Simulation parameters

Position sensor standard deviation	0.5 m
Velocity sensor standard deviation	0.5 m s^{-1}
Cars maximum speed	8 m s^{-1}
Pedestrians maximum speed	2 m s^{-1}
Simulation time step	0.1 s
Decision frequency	0.5 s
Belief update frequency	0.1 s

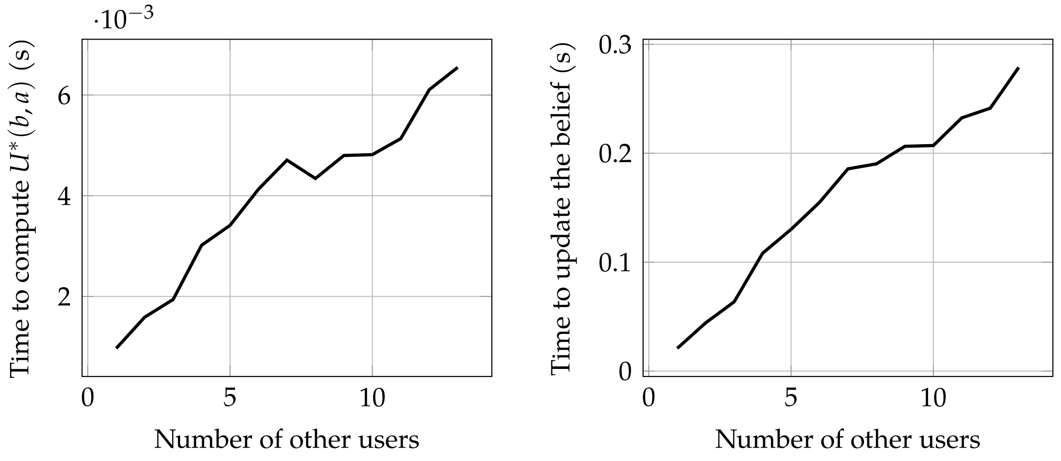
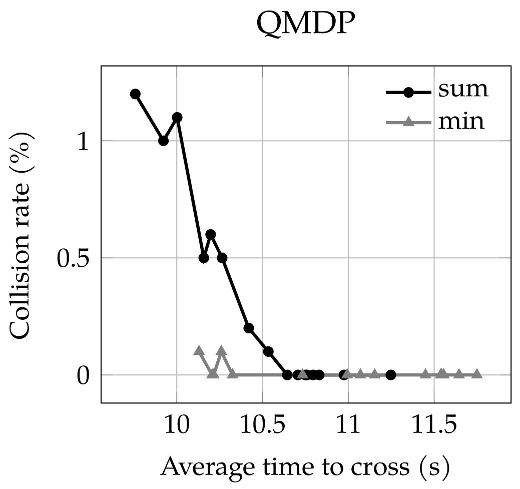


Figure 5.3: The computation time required to compute the belief utility function (QMDP) and to update the belief both increase linearly with the number of traffic participants detected.

action takes about the same time as the belief update.

Figure 5.4 shows the trade-off between safety and efficiency on the crosswalk scenario. For both POMDP planners, as we increase the cost of colliding with another user, the policy becomes safer (fewer collision) but less eficient (longer time to cross). In addition, the simulations were repeated for the two proposed fusion functions. We can see on both graphs that using the minimum yields to safer policies. Moreover, there is a region of Pareto domination of the policy using minimum over the policy using summation. For SARSOP, all the points are dominated, whereas for QMDP there is a region where both functions lead to a safe policy. From this figure, we can select conservative policies that lead to a safe behavior for each planner.



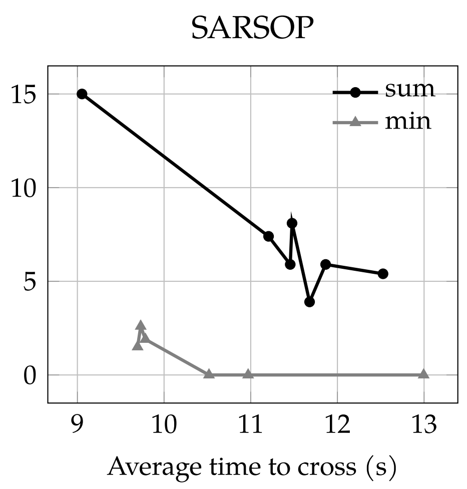


Figure 5.4: Evolution of the performance of QMDP (left) and SARSOP (right) as the collision cost is changed for the crosswalk scenario. The two fusion functions sum and min are being analyzed.

The associated collision costs are -1.5 for QMDP and -30 for SARSOP. Similar experiments for the intersection scenario lead to a collision cost of -1.6 for QMDP and -17 for SARSOP. To produce the results in Table 5.2 we chose the minimum function for fusing the utilities.

The baseline policies are tuned using a similar procedure as in Chapter 3. The two parameters to tune are the threshold on the time to collision and the verification time. For the crosswalk scenario, checking that the time to collision is below 10 s during 10 decision steps leads to the safest strategy. For the intersection scenario, a TTC threshold of 6 s and a verification time of 8 decision steps leads to the safest strategy. Table 5.2 summarizes the performance of the baseline policies. They are also compared to a random policy to assess the difficulty of the scenario.

For the crosswalk scenario, the results show that the two POMDP policies outperform the baseline on the two metrics. The rule-based baselines did not lead to a completely safe strategy. Handling measurement uncertainty by increasing the verification time is not as robust as a Bayesian state estimation approach. It is often challenging to account for state uncertainty in rule-based methods without being overly conservative. Because we prioritized safety, the rule-based policy is in

Table 5.2: Performance of the different policies for the occluded scenarios with multiple traffic participants.

	Collision rate (%)	Time to cross (s)
Crosswalk		
Random	54.45 ± 2.56	12.03 ± 10.66
Baseline	0.1 ± 0.04	18.58 ± 5.39
QMDP	0.0 ± 0.0	10.61 ± 3.76
SARSOP	0.0 ± 0.0	10.51 ± 4.44
Intersection		
Random	28.00 ± 2.27	14.68 ± 6.80
Baseline	0.1 ± 0.04	13.46 ± 3.04
QMDP	0.0 ± 0.0	6.20 ± 2.108
SARSOP	0.7 ± 0.83	4.38 ± 0.1342

average 8 s slower than the POMDP approach. This experiment confirms that the POMDP framework is better suited to trade off between conflicting objectives.

For the intersection scenario, the SARSOP policy presented more collisions than the baseline and QMDP. The solver was given a maximum of twelve hours of computation time but converged to an aggressive policy. The single user intersection problem is at the limit of what state space size can be handled by SARSOP [17]. Further reward engineering or parameter tuning could certainly be done to achieve better performances. In contrast, the QMDP policy is less efficient but it is safer than SARSOP. This policy is still safer and more efficient than the baseline policy. The few collisions for the baselines are due to cars arriving in a critical zone right after the ego car has made the decision to cross. In contrast with the POMDP approach, the ego car is not able to change its decision dynamically. This qualitative result highlights the difficulty of engineering a good safety criterion such that all future decisions are safe. The low gaw gap between the performance of QMDP and SARSOP implies that this problem does not require information gathering behavior. The design of the action space (longitudinal motion constrained to a path) makes information gathering happen naturally as the agent moves towards the goal.

In addition to providing a safe and dynamic policy, smart behavior emerged from the POMDP planning. If no road users are present in the environment, the baseline policy always comes to a stop and check the TTC for the given number of steps resulting in a suboptimal behavior whereas both POMDP policies slow down without necessarily stopping. Another major difference between the two approaches is how occlusions are handeld. In the manually designed policy, one must hand-code the action to stop when there is no visibility whereas this behavior emerges automatically from the POMDP planning approach indifferently for both scenarios. For these reasons, the proposed approach is more efficient and generalizes more easility than rule-based methods.

5.5 *Discussion*

In this chapter we discussed an approach to scale POMDP techniques to environment with multiple traffic participants using a decomposition method. By leveraging the solution to avoiding a single road user, utility fusion allows to approximate the policy of avoiding multiple users at a very low computational cost compared to solving the full POMDP. We analyzed the performance of two different offline POMDP planners to solve autonomous driving scenarios involving a crosswalk and an intersection. Simulations showed that the proposed method is safer and more efficient than rule-based policies.

The choice of this function can greatly affect the performance of the global policy and has no guarantee of optimality. A disadvantage of this decomposition method is that it requires choosing a fusion function that will play the role of arbitrator between the different utilities considered. In fact, it can be shown that when the individual utility functions are computed, they converge to ''selfish'' estimates [74]. Instead this solution provides a low-fidelity approximation of the optimal value function for little computational cost once the individual utility functions are computed. The next chapter present an efficient learning algorithm to improve this solution. Decomposition is also helpful in problems where several agents must be coordinated. One can consider each agent individually and find a joint action by maximizing the sum of the individual utilities. This setting will be discussed in more depth in the next chapter.

6 Deep Corrections

Although decomposition methods have been shown to perform well empirically, they rely on strong independence assumptions on the local task. Computational benefits are obtained in expense of sacrificing optimality on the global task. In this chapter we demonstrate a technique to mitigate this effect by learning a correction term to the approximate solution of the problem through deep reinforcement learning. We introduce a novel algorithm, *deep corrections* to improve the approximate solution given by decomposition methods. The algorithm is presented in the context of reinforcement learning, where we search policies mapping sequences of observations to actions. The work presented in this chapter have been conducted in collaboration with Kyle D. Julian, Alireza Nakhaei, and Kikuo Fujimura [51].

6.1 Related Work

In the previous chapter we discuss problems where an agent must interact with multiple entities. Decomposition methods can also be applied to problems where multiple agents are controlled. In resource allocation problems [74], [81], all the agents must be coordinated and assigned to some resources to maximize a common objective. We make the distinction between *multiple entities* problems where one agent must react agains non-controlled entities (*e.g.* collision avoidance) and *multi-agent* problems when several agents must cooperate or be coordinated (*e.g.* resource allocation). The general problem of decentralized control of multiple agents in a stochastic, partially observable environments has been shown to be intractable [82].

An alternative approach in leader follower scenarios consists of reducing the problem of controlling the group of agent to controlling a single group leader [83]. Other approaches rely on distributed learning algorithms, such as independent Q-learning, where each agent learns a policy without being aware of the other agents actions [84]. The underlying assumption of independence betweena gents in these algorithms trades off the benfit of collaboration for an easier learning of the task [85]. Van de Pol and Oliehoek used utility decomposition to coordinate traffic lights across a road network [86]. Instead of considering each cooperating agent as an individual, one can formulate the problem through pairwise interactions, as demonstrated on a wildfire surveillance problem [87]. Efficient search strategies might be required to compute the joint action maximizing the sum of the individual utilities [73], [88]. Although maximizing the sum of the individual utilities can be optimal under certain conditions [74], this arbitration method will be suboptimal in many problems. In resource allocation tasks, the individual agents might have selfish policies, so arbratrating selfish policies by maximizing the sum of utilities can lead to a problem known as "the tragedy of the commons" [89]. To address the problem of selecting an appropriate utility fusion function, it has been suggested to learn the structure of the composition using a neural network representation [90], [91].

Another approach to scaling reinforcement learning algorithms leverages the representational power of deep neural networks in modeling the utility function. Although this approach can handle domains such as Atari games and complicated manipulation tasks, it often requires millions of training examples and long training times [18], [92]. To minimize training time, transfer learning can take advantage of prior knowledge. For example, an existing controller or human expert can gather high reward demonstrations to train an autonomous agent [93]. Another transfer learning approach uses a regularizing term to guide the agent toward regions where it has prior knowledge of the appropriate behavior [94]. Multi-fidelity optimization has been combined with model-based reinforcement learning [95], where a learning framework is proposed for efficiently deciding between sampling from an inexpensive low-fidelity simulator and real world data.

6.2 Deep Corrections

The previous chapter demonstrated how to use decomposition method to solve large decision making problems. However, this technique sacrifices the optimality of the solution. We propose a correction method using deep reinforcement learning to refine the approximation of the state action value function of the global problem.

Deep Reinforcement learning methods are likely to scale to the collision avoidance problem described in the previous chapter even without the decomposition method. However, it might require a very large number of experience samples and a tedious hyperparameter tuning procedure. Computing the optimal Q function with the deep Q-learning algorithm requires many experience samples mostly because the agent must go through a random exploration phase. By introducing prior knowledge into the algorithm, the amount of exploration required to converge to a good policy is significantly reduced. In this chapter, we consider that prior knowledge takes the form of a state action value function. No specific representation is needed. The decomposition method described in the previous chapter would be a simple and efficient way to generate prior knowledge. There are may techniques to retrieve a state action value function, such as discretizing a continuous state space and using a standard MDP or POMDP planner or using domain expertise such as phyics-based models. If a policy already exists, a state action value function can be estimated using the Monte Carlo evauation algorithm [52]. Given an approximately optimal value function, the objective is to leverage this prior knowledge to learn the optimal value function with as little compuation as possible.

The method we propose is inspired by multi-fidelity optimization [96]. Consider a setting with two models of different fidelities: an expensive high-fidelity model, f_{hi}, and a low-fidelity model, f_{lo}, providing an inexpensive but less accurate approximation. In multi-fidelity optimization, the goal is to maximize an objective function f approximated by these two surrogate models. Using only the high-fidelity model would be too expensive since many queries are usually needed to optimze f. On the other hand, rleying only on the low-fidelity model might lead to a poor solution.

Instead we could construct a parametric model that approximates f_{hi} but is inexpensie to compute. Such a surrogate model can represent the difference between the high-fidelity and the low-fidelity models [96], [97]:

$$f_{\text{hi}}(x) \approx f_{\text{lo}}(x) + \delta(x) \tag{6.1}$$

where x is the design variable and δ is a surrogate correction using a limited number of samples from f_{hi}. Commonly, δ is modeled using Gaussian processes or parametric models. In this work, we focused on additive surrogate correction, but a multiplicative correction or a weighted combination of both would be a valid approach as well [96].

Reinforcement learning can be reformulated as a multi-fidelity optimization problem. Let Q_{lo} be our low-fidelity approximation of the state action value function obtained from prior knowledge or from a decomposition method. The high-fidelity model that we want to obtain is the optimal state action value function. The surrogate correction for reinforcement learning can be described as follows:

$$Q^*(s, a) \approx Q_{\text{lo}}(s, a) + \delta(s, a; \theta) \tag{6.2}$$

In regular multi-fidelity optimization problems, samples from f_{hi} are used to fit the correction model. In reinforcement learning, Q^* is unknown, so a temporal difference approach is used to derive a learning rule similar to the Q-learning algorithm. We seek to minimize the same loss function described in Equation (4.1). The main difference lies in the representation of the Q funciton, which can be substituted by Equation (6.2). Only the corrective part is parameterized. As a consequence, when taking the gradients with respect to the parameters, the update rule becomes:

$$\theta \leftarrow \theta - \alpha[r + \gamma \max_{a'}(Q_{\text{lo}}(s', a') + \delta(s', a'; \theta_-)) - Q_{\text{lo}}(s, a) - \delta(s, a; \theta)]\nabla_\theta \delta(s, a; \theta) \tag{6.3}$$

Equation (6.3) would be the only difference to the algorithm if another correction method is used. We can generalize the notation to any parametric corrective function

f: $Q(s,a;\theta) \approx f(s,a,Q_{\text{lo}}(s,a);\theta)$. As the DQN algorithm relies on efficient gradient computation, it assumes that the gradient of f with respect to θ can be computed easily.

```
1:  input: $Q_{\text{lo}}$, $G$ a generative model of the environment, an exploration strategy
    ($e.g.$ $\epsilon$ greedy).
2:  hyperparameters: similar as DQN [18], let $T$ be the target network update
    frequency
3:  Initialize $\delta$
4:  for each episode do
5:      initialize state $s$
6:      while $s$ is not a terminal state do
7:          select $a = \begin{cases} \text{random with probability } \epsilon \\ \arg\max_{a'}(Q_{\text{lo}}(s,a') + \delta(s,a';\theta)) \end{cases}$
8:          $s', r \sim G(s,a)$
9:          Store transition $(s,a,r,s')$ in a replay buffer
10:         Sample batch of transitions from the replay buffer
11:         Update $\theta$ using Equation (6.3) for each sampled transitions, $Q_{\text{lo}}$ stays
    unchanged
12:         set $\theta_{-} = \theta$ every $T$ steps
```

Algorithm 6.1: Deep Corrections Algorithm

Instead of learning a Q-function representing the full problem, an additive correction is represented as a neural network. By using this representation of the corrective term, all the innovations (dueling, prioritized replay, double Q-network) used to improve the DQN algorithm can also be used when learning the correction term. As illustrated in algorithm 6.1, the psuedocode of the deep corrections algorithm is very close to the one of DQN with the exception of the representation of the Q function (line 7 and 11). Although the algorithms are similar, learning the correction to a given value function happens to be easier than learning the full value function. In the extreme case where $Q_{\text{lo}} = Q^*$, δ should be a constant function equals to 0 which is easy to represent. In contrast, if Q_{lo} is random, then learning δ is as hard as learning the value function using standard DQN.

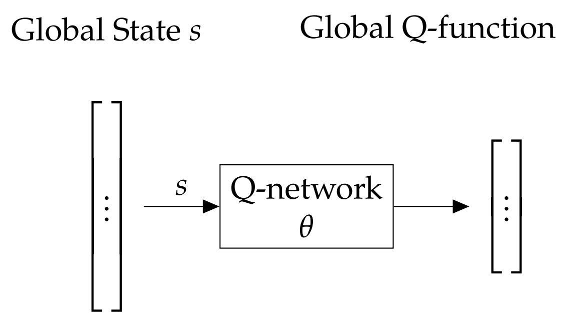
An illustration of th deep correction architecture along wih the decomposition method is presented in Figure 6.1. In order to use this approach jointly with a decomposition method, we can substitute Q_{lo} by the results from utility fusion. For example substituting Equation (5.2) in Equation (6.2) would lead to the following approximation of the global Q function.

$$Q^*(s,a) \approx \sum_i Q_i(s_i,a) + \delta(s,a;\theta) \tag{6.4}$$

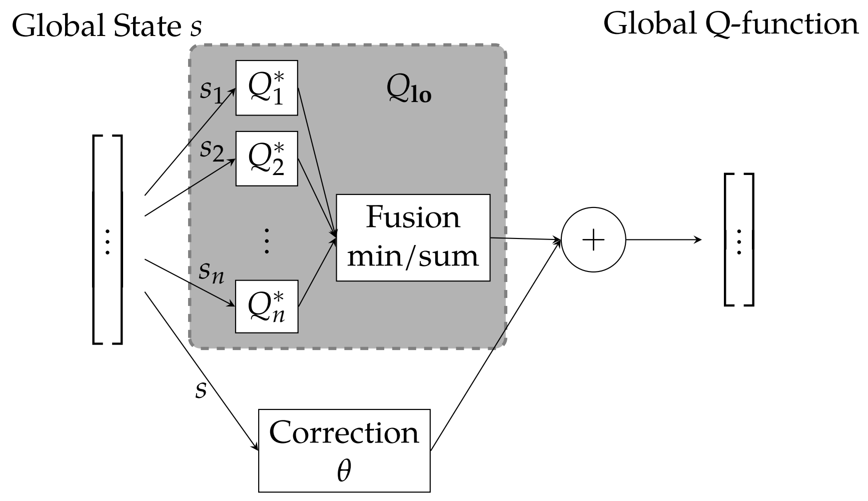
The value functions Q_i come from the solution to the subtasks. In the pedestrian collision avoidance scenario, the value functions are the same, although other settings could use distinct value function to represent Q_i. We can see from the update rule with correction that these functions are constant with repsect to θ. In practice, if the individual Q_i are represented by neural networks, the weights are frozen during the training of the correction term.

One advantage of framing the deep corrections algorithm as a multi-fidelity optimization problem is the flexibility in the form of Q_{lo}. There are no particular constraints on the representation of Q_{lo}. Various approaches can be used to obtain an approximate value funciton for the problem. In this work we exclusively use decomposition methods to proide Q_{lo} as they have been shown to provide good policies according to the results from the previous chapter. Those method often rely on some independence assumptions between the entities [74], [84]. The role of the correction network is to fill the difference between the low fidelity approximation and the true value function of the global problem with multiple entities.

The low-fidelity approximation of the value function is a way to introduce prior knowledge in the algorithm. The fidelity of Q_{lo} corresponds to how close it approximates the optimal value function. Generally, the closer Q_{lo} is from the optimal value fucntion, the easier it is to learn δ. Intuitively, starting with the policy from utility fusion significantly reduces the amount of exploration in DQN to learn a good strategy. On the other hand, if Q_{lo} is a poor approximation, δ has an important contribution to the performance of the final policy. To control the impat of the corrective factor if one has trust in the low-fidelity policy, a regularization term could be added to the loss function. Although we used a neural network representation



(a) Standard Q-network



(b) Decomposition and Deep Corrections

Figure 6.1: Regular DQN architecture (a) and architecture of the deep correction approach used with the decomposition method (b). The global state is decomposed and each sub-state is fed into networks pre-trained on the single entity problem. The output of the correction network is then added to the output of the utility fusion to approximate the global Q-function. More generally, the shaded gray area could be replaced by any existing value function Q_{lo}.

for δ, th mulit-fidelity formulation from Equation (6.2) generalized to any class of parametric functions.

A limitation of our method is that the correction network could potentially get stuck in a local optima. Moreover it could harm the performance during training as it is initialized randomly. Further work must be done to provide constraints on the policy improvement or convergence guarantees. We do not provide theoretical guanratees on the convergence of the deep Q-learning algorithm used to learn the correction term. However we demonstrate in the next sections that despite the lack of guarantees, the deep corrections algorithm still learns good policies.

The last section of the chapter are dedicated to applying the deep corrections approach and the decomposition method to two different scenario. The problems and the deep corrections algorithm were implemented using the POMDPs.jl framework [57] and the Flux.jl deep learning framework [58]. Our implementaion is available at `https://github.com/sisl/DeepCorrections.jl`.

6.3 *Cooperative Multi-Agents Problems*

In cooperative multiple agents settings, the local problems are instances of the global problem with a single agent or a subset of agents. This section presents an application of the deep corrections method to a multi-agent problem. The goal is to coordinate multiple agents to mximize a global objective. We consider a similar fishieries management problem as Russell and Zimdars [74]. This problem belongs to the broader category of multi-agent resource allcoation problems, which has also been applied to server allocations [81]. These two preious works rely on decomposition methods to find approximately optimal policies.

6.3.1 *Model description*

Several fishermen must coordinate to catch a maximum number of fish. If each fisherman follows a selfish strategy, the fish stocks will collapse resulting in a "tragedy of the commons" [89]. The fisheries commissioner, a centralized arbitrator, must

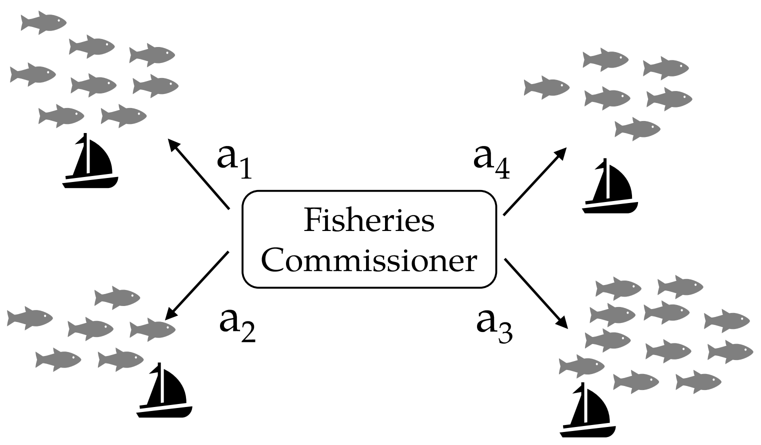


Figure 6.2: Illustration of the fisheries management problem with four boats. The fisheries commissioner must assign the proportion of fish that each fisherman can fish to maximize the aggregate catch over time.

assign a proportion of the local fish population that each fishermen is allowed to catch at each season. The goal of the commissioner is to maximize the global catch. We consider a fishery with n fishermen fishing in local regions. A season consists of two steps: a mating season and a fishing season. During the mating season, the fish population reproduces and is then spread across n regions with equal probability. During the fishing season, each fisherman is assigned a proportion of fish to catch in their region.

The fisheries management problem can be modeled as a multi-agent MDP:

- *State space*: the state consists of the number of fish in each region at time t: $s_t = (f_{1t}, \dots, f_{nt})$. The local task of the fisheries problem has a one dimensional state equals to f_{it} at time t for fisherman i.

- *Action space*: Let $\mathcal{A}_{\text{local}} = \{1, 0.5, 0.3, 0.1\}$ the possible individual actions, corresponding to a proportion of fish to catch in a given region. The global action space consists of the joint action space, $\mathcal{A} = \mathcal{A}_{\text{local}}^n$. Given $a \in \mathcal{A}_{\text{local}}^n$, a_i corresponds to the proportion of fish assigned to fisherman i.

- *Transition model*: During the mating season, the fish population reproduces

Table 6.1: Parameters for the fisheries management problem

number of boats	10
initial population	1.5×10^5
maximum population	3×10^5
minimum population	200
growth rate	0.5
fishing cost ζ	1×10^3
boat efficiency η	0.98
discount factor	0.99

according to the following model:

$$f_{t+1} = f_t \exp(G(1 - \frac{f_t}{f_{\max}}) \tag{6.5}$$

where $f_t = \sum_{i=1}^{n} f_{it}$ is the total fish population at time t, G is the population growth rate, and $f_{\max}$ is the maximum fish population. After the mating season, the fish are uniformly assigned to the different regions. The number of fish, c_{it}, caught by the fisherman i, follows a Poisson distribution of mean $\eta a_{it} f_{it}$ where η is a boat efficiency parameter and a_{it} is the proportion of the local fish population assigned to fisherman i at time t.

- *Reward model*: The individual reward of fisherman i is given by $r_i(c_{it}, a_{it}) = C(c_{it} - \zeta a_{it}^2)$, where ζ is a constant reflecting the cost of fishing and C is a normalization factor equal to $n/f_{\max}$. The term proportional to a_{it}^2 reflects the fact that catching more fish requires more resources such as equipment or crew members and is hence more costly. We used the same reward definition as Russell and Zimdars. The global reward consists of the sum of the individual rewards.

An episode ends after the fishery has survived for 100 seasons or when the fish population goes below a minimum. The numerical values for the parameters of the fisheries problem are summarized in Table 6.1.

6.3.2 Multi-agent policies for the fisheries management problem

We compare the performance of three different policies against simple baselines.

Baselines: The first baseline is a random policy where the commissioner selects a random action for each boat. The other baselines consists of single action policies where each boat is allowed to fish a fixed proportion of fish at every step. The policy with fixed action 0.1 (harvesting a tenth of the fish population) corresponds to a conservative policy where each boat fish the minimum possible number of fish. The policy with the fixed action 1.0 is a greedy policy.

Solving the full problem using the conventional DQN algorithm is intractable. The size of the action space grows exponentially with the number of boats: with 10 boats, there are 4^{10} possible actions. Instead, we use the different decomposition methods presented below.

Max-sum: We first solve a single agent problem of assigning the proportion of fish to catch for one boat in one region. The environment follows the same model as in the multi-agent setting but with an initial population reduced to 15,000. A Q-network is computed using DQN, with a one-dimensional input corresponding to the current amount of fish available. To generate a multi-agent policy, we summed the individual utilities given by the solution to the single-agent problem as follows:

$$a^* = \arg\max_{a_1,\dots,a_n} \sum_i Q_{\text{single}}(s_i, a_i) \tag{6.6}$$

where s_i is the number of fish in region i after the fishing season and a_i the local action of fisherman i.

Decomposed DQN: We learn n different Q-networks in a decomposed manner as suggested by Russell and Zimdars with the difference that each Q-network takes the global state into account and is trained using the global reward signal [74]. At every training step, we perform one parameter update for each Q-network. Each network predicts the value of the global state for each local action. The dimension of the output is $|\mathcal{A}_{\text{local}}|$. The joint action is again given by maximizing the sum of the utilities:

$$a^* = \underset{a_1,\dots,a_n}{\arg\max} \sum_i Q_i(s, a_i; \theta_i) \tag{6.7}$$

Max-sum with correction: Finally, we apply the deep corrections method to improve the max-sum policy. The correction function is represented by a neural network and receives the full state as input but predicts the value of the local action (same input output as for the decomposed DQN). The correction term is trained using a decomposed version of algorithm 6.1 (one correction network per boat). The joint policy is then given by:

$$a^* = \underset{a_1,\dots,a_n}{\arg\max} \sum_i Q_{\text{single}}(s_i, a_i) + \delta_i(s, a_i; \theta_i) \tag{6.8}$$

During the training of the correction terms, the low fidelity approximation given by Q_{single} is not updated. The correction network has access to more information than the original policy.

6.3.3 *Experiments*

In order to fairly compare the three methods, we allocate a fixed training budget of 160k experience samples. For the deep corrections approach, the budget is split as follows: 100k examples are used to learn the single agent Q-network and 60k examples are used to train the correction networks. For the decomposed DQN approach, the entirety of the budget is used to learn the Q-networks. In all three trainings, the same neural network architecture is used: one hidden layer of sixteen nodes with rectified linear units activations. The dimensionality of the input is either one (for the single agent problem) or ten (for the multi-agent problem), and the output of the networks is always of dimension $|\mathcal{A}_{\text{local}}| = 4$. Larger architectures did not improve performance or would require more experience samples to achieve a similar accumulated reward. For the three policies, we used the same hyperparameters, shown in Table 6.2, and an ϵ-greedy policy with a linearly decaying ϵ for exploration.

To evaluate the policies, we measure the average accumulated undiscounted reward across 100 simulations on the fisheries management problem. The results

Table 6.2: Deep Q-learning hyperparameters for the fisheries management problem

Neural network architecture	1 fully connected layer of 16 nodes
Activation functions	Rectified linear units
Replay buffer size	500 k experience samples
Target network update frequency	2 k episodes
Discount factor	0.99
Optimizer	Adam
Learning rate	1×10^{-4}
Prioritized replay [56]	$\alpha = 0.7$, $\beta = 1 \times 10^{-3}$
Exploration fraction	0.2
Final ϵ	0.05

from these simulations are reported in Figure 6.3 for the five baselines policies, the max-sum policy, the decomposed DQN policy and the policy improved with deep corrections.

6.3.4 Results

We can see from Figure 6.3 that the learned policies outperform all the baselines. As expected, the greedy policy, the random policy, and the $a = 0.5$ policy (harvesting half of the population all the time) perform poorly and are victims of "the tragedy of the commons". All other policies allowed the fisheries to survive for a hundred seasons because the stock never ran out. The conservative policy reaches a score of 8.47 and is outperformed by the policy harvesting 0.3 of the fish population which scored 12.47. The max-sum approach gives a significant improvement over the conservative policy with a score of 12.62, but only a minimal improvement compared to the the policy with $a = 0.3$. When learning the corrective term in addition to the conservative policy, the score was improved as it achieves 13.9. The decomposed DQN approach achieved a close performance of 13.7, which is lower than the policy with corrections. In this problem, the gap between the correction method and the DQN approach is not very large, but the correction network converged with 60 k samples only (against 160 k for the decomposed DQN).

These results show that the deep corrections method efficiently improves an

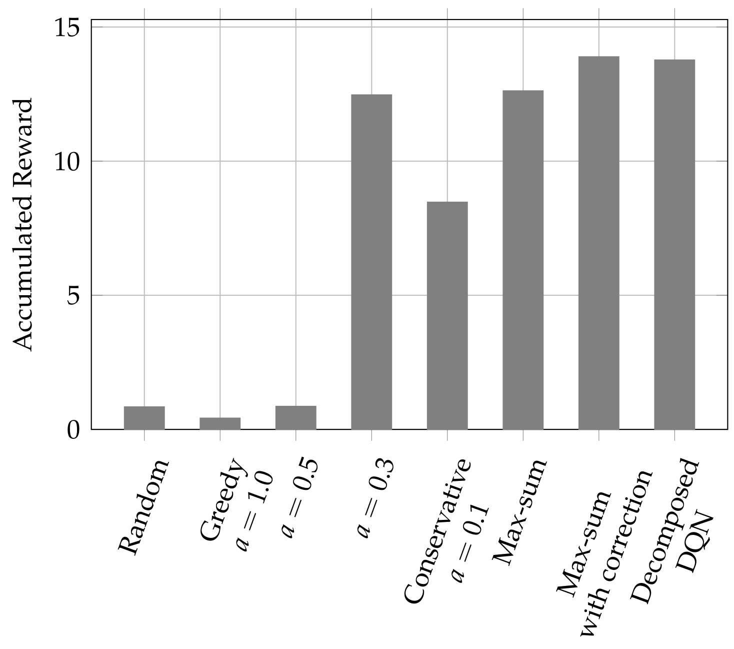


Figure 6.3: Performance of different policies on the fisheries management problem with ten boats.

existing policy and that utility decomposition is not optimal. Moreover, this result demonstrates that the corrected policy achieves better performance than a policy trained to solve the full problem using DQN, or in this case a decomposed DQN approach.

This section was a digression from the main topic of decision making for autonomous vehicles. It serves the purpose of showing the generality of the algorithmic contribution present in this thesis. The next section illustrates the application of the deep correction algorithm to the pedestrian avoidance scenario from the previous chapter.

6.4 Deep Corrections for the Occluded Crosswalk Scenario

This section applies the deep corrections approach to the pedestrian avoidance scenario described in Section 5.2 and Chapter 5. We demonstrate that learning the correction term improves the resulting policy using the decomposition method only.

Table 6.3: Deep Q-learning parameters for the occluded crosswalk problem

Neural network architecture	5 fully connected layers of 32 nodes
Activation functions	Rectified linear units
Replay buffer size	400 k experience samples
Target network update frequency	5 k episodes
Discount factor	0.99
Optimizer	Adam
Learning rate	1×10^{-4}
Prioritized replay [56]	$\alpha = 0.7$, $\beta = 1 \times 10^{-3}$
Exploration fraction	search between 0.0 and 0.9
Final ϵ	search between 0.0 and 0.1

In this case study we compare the performance of three different approaches.

Deep Q-learning: The first approach uses deep Q-learning to solve the full problem. Contrary to the model described in Section 5.2, the state space is not discretized. It consists of the continuous values of the longitudinal position of the pedestrians and the ego vehicle, as well as their velocities. To accomodate with the continuous state space we will approximate the state action value function using a neural network. The hyperparameters used for the training algorithm are reported in Table 6.3. A challenge for the occluded crosswalk scenario is that the number of pedestrians present in the scene is unknown. As a result, there are twenty-two state dimension, with two dimensions for the ego car position and velocity and two dimensions for each pedestrian position and velocity.

In Section 5.2, the input to the value function was a belief state, that was updated at each time step using Equation (2.3). Here we use the k-Markov approximation instead, that is the input to the value function is a history of the last k observations ($k = 4$). The increase of dimensionality of the input can be handled by the neural network approximation of the value function. This approach could be improved by using a recurrent neural network to handle the partial observability, but this is left as future work.

Utility Fusion: In this approach we apply the decomposition method presented in Chapter 5. For each pedestrian i in the environment, the ego vehicle observes a state s_i consisting of its own position and velocit as well as the position and velocity of the pedestrian. We compute a state-action value function $Q_{\text{single}}(s_i, a)$ assuming i is the only user to avoid. The global value function is then approximated using the max-sum and max-min methods presented in the previous chapter.

Policy Correction: Using the max-sum and max-min policies as low-fidelity approximations of the optimal value function, we learn an additive correction term. The correction function is represented by a neural network, which is optimized using algorithm 6.1. The input to the correction term is similar to the one used for the deep Q-learning approach.

In this collision avoidance problem, the number of pedestrians to avoid is not known in advance due to the presence of the obstacle. To address the issue, we capped the number of possible pedestrians to consider to ten. This maximum number of pedestrians was measured empirically and may change with a different probability of appearance for the pedestrians.

6.4.1 Experiments

All proposed solutions methods involve training a Q-network: on the full problem, on the single pedestrian problem, and on the corrective term. In order to fairly compare the three methods, a fixed training budget of one million experience samples was used for all three methods. For the policy correction technique, part of the sample budget was used to train the decomposed policy. The sample budget is divided equally between training a Q-network on the single pedestrian problem and training the corrective term. A hyperparameter search was executed to select the neural network architecture and the target network update frequency. Other parameters were set to common values found in the literature [18], [55], [56].

During training, the agent interacts with the environment following an ϵ-greedy policy. The value of ϵ is scheduled to decrease during training, and we found that the amount of exploration greatly influences the final policy. For the three approaches (training from scratch on the full problem, training on the problem with only one

Table 6.4: Environment Parameters

	Evaluation	Training
Position sensor standard deviation	0.5 m	0.5 m
Velocity sensor standard deviation	$0.5\,\mathrm{m\,s^{-1}}$	$0.5\,\mathrm{m\,s^{-1}}$
Decision frequency	0.5 s	0.5 s
Simulation time step	0.1 s	0.5 s
Pedestrian velocity maximum noise	$0.5\,\mathrm{m\,s^{-1}}$	$1.0\,\mathrm{m\,s^{-1}}$

pedestrian, and training the correction factor), a random search over the exploration schedule was conducted. The exploration fraction represents the fraction of total training time used to linearly decay ϵ from 1.0 to its final value. Each training sample is initialized randomly, the ego vehicle starts at the same position with a velocity uniformly sampled between $6\,\mathrm{m\,s^{-1}}$ and $8\,\mathrm{m\,s^{-1}}$. The initial number of pedestrians and their positions and velocities is also randomly sampled. From this initial state, the state is updated given the action of the agent and the dynamic model described in Section 5.2.

All trained policies are evaluated in an evaluation environment that differs from the training environment. It has a finer time discretization, and the model followed by the pedestrian is less stochastic, as reported in Table 6.4. The weights of the networks are frozen during the evaluation. Each simulation is randomly initialized: random initial velocity for the ego vehicle, random number of pedestrians present with a random position and velocity. Each trained policy is evaluated on the two metrics of interest that are averaged over a thousand simulations: collision rate and time to pass the crosswalk. Since these two objectives are conflicting, Pareto optimality is used to decide if a policy is better than another. Formally, we say that a policy dominates another if it outperforms the other policy in one objective and is no worse in the other objective. From this definition, we can draw the Pareto frontier associated with all the generated policies from the hyperparameter search. The frontiers for the three approaches is represented in Figure 6.4.

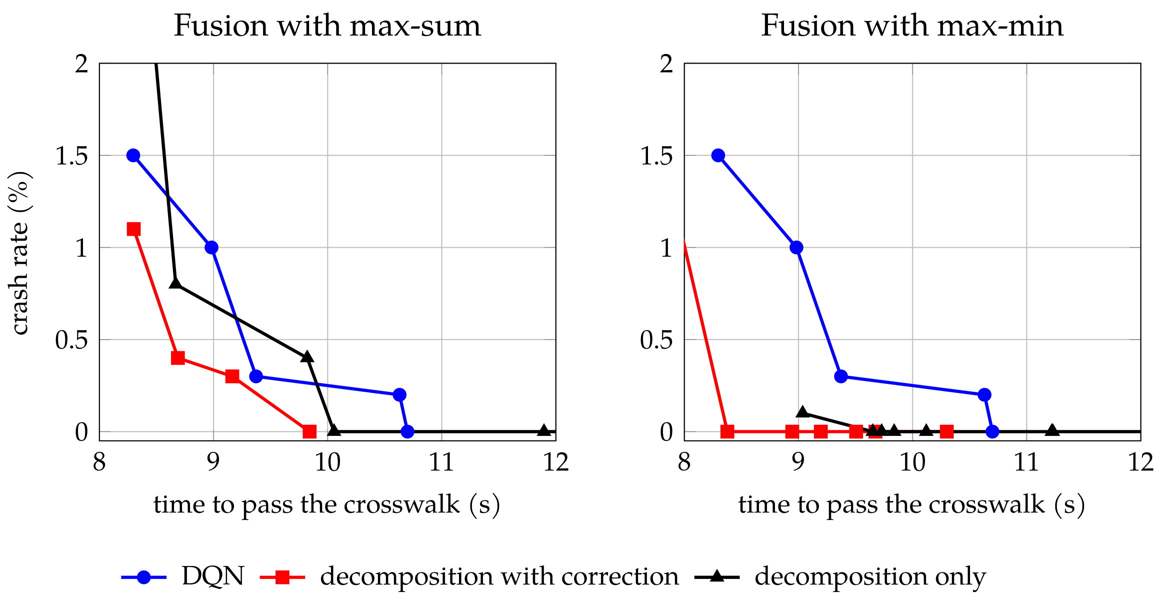


Figure 6.4: Pareto frontiers for the different approaches used to solve the crosswalk problem: DQN, utility decomposition, and utility decomposition with correction. Each point results from evaluating a policy generated by a given set of hyperparameters. The figure is zoomed in the region of optimality (bottom left). The two figures represents the max-sum (left) and the max-min (right) fusion techniques.

6.4.2 Optimality of the policies

The Pareto frontiers in Figure 6.4 show a domination of the policy correction method. If we keep one objective fixed, we can always find a policy computed with the correction method that outperforms the two other methods in the other objective. A reasonable approach would be to fix the safety level at 0 % of collisions over the thousand simulations and use the policy that minimizes the time to pass the crosswalk. Using the max-sum fusion, the fastest safe policy has an average time to cross of 10.06 s. By adding the correction term, the policy achieves an average time to cross of 9.84 s. Using max-min, the policy reaches an average time of 9.65 s, and 8.38 s with the correction. The deep Q-network policy has a time to cross of 10.70 s. For both utility fusion methods, max-sum and max-min, the addition of the corrective factor not only leads to an improvement in the policy but also outperforms the deep Q-network policy. The max-min decomposition method even dominates the Q-network trained on the complex environment. This result illustrates that the choice of a good function for combining the utilities can provide a good policy without training in the multi-pedestrian environment. However, Figure 6.4 shows that the max-sum decomposition without correction does not dominate the Q-network approach. In the scenario of interest, considering the agent with the minimum utilities favors risk averse behavior, resulting in safer policies for a given time to pass the crosswalk.

To gain intuition on the difference between these policies, we can visualize them on a two-dimensional slice of the state space in Figure 6.5. To generate this representation we ran simulations with the ego vehicle driving at a constant speed of $6.0\,\mathrm{m\,s^{-1}}$ (not reacting to the policy), and a single pedestrian fixed at a given position along the crosswalk. Although the policy is being tested against one pedestrian, it is still assuming that there might be multiple pedestrians. Figure 6.5 shows the actions returned by the policy at a given ego car position along the road and a given position of the pedestrian. The pedestrian is located at 25 m along the road and the ego car is moving along a horizontal line that intersects the crosswalk at 0 m. All plots show a red zone just on the left of the position $(25, 0)$ representing a braking

Table 6.5: Best Hyperparameters

Model	Exploration Fraction	Final ϵ
DQN	0.50	0.01
max-min correction	0.00	0.01
max-sum correction	0.20	0.00

behavior when a pedestrian is in the middle of the road. The larger to the left the red area is, the more the car will anticipate the braking. Similarly, all the policies present a vertical red zone before 15 m along the road, indicating that the car will slow down regardless of pedestrian position. This behavior is natural since the presence of the obstacle hides a lot of information. A safe behavior is to slow down until the vehicle has more visibility. These plots can help us visualize areas of the state space where the policy might be suboptimal. For example, the two policies using the decomposition method without the correction present a braking area after the crosswalk. The suboptimal areas after the crosswalk illustrate that all methods used in this paper only approximate the optimal value function. In the area after the crosswalk, braking areas should disappear with more training samples. It is also important to note that the action values in those parts of the state space are very close since it is possible to reach the goal with a probability of 1 in only a couple time steps. These suboptimal parts disappear for the max-min policy with the correction term. In the max-sum case, we can see that the correction relaxed a large part of the braking area to use moderate braking ($-2\,\mathrm{m\,s^{-2}}$) rather than strong braking ($-4\,\mathrm{m\,s^{-2}}$).

6.4.3 *Training Performance*

We looked at the evolution of the policies performance during training. Although they were trained with half the samples, the policies with the correction term converge. Since the policy resulting from the decomposition method already has performances close to the deep Q-network policy, only very little exploration is required to learn the corrective term. The max-min decomposition results in more stable

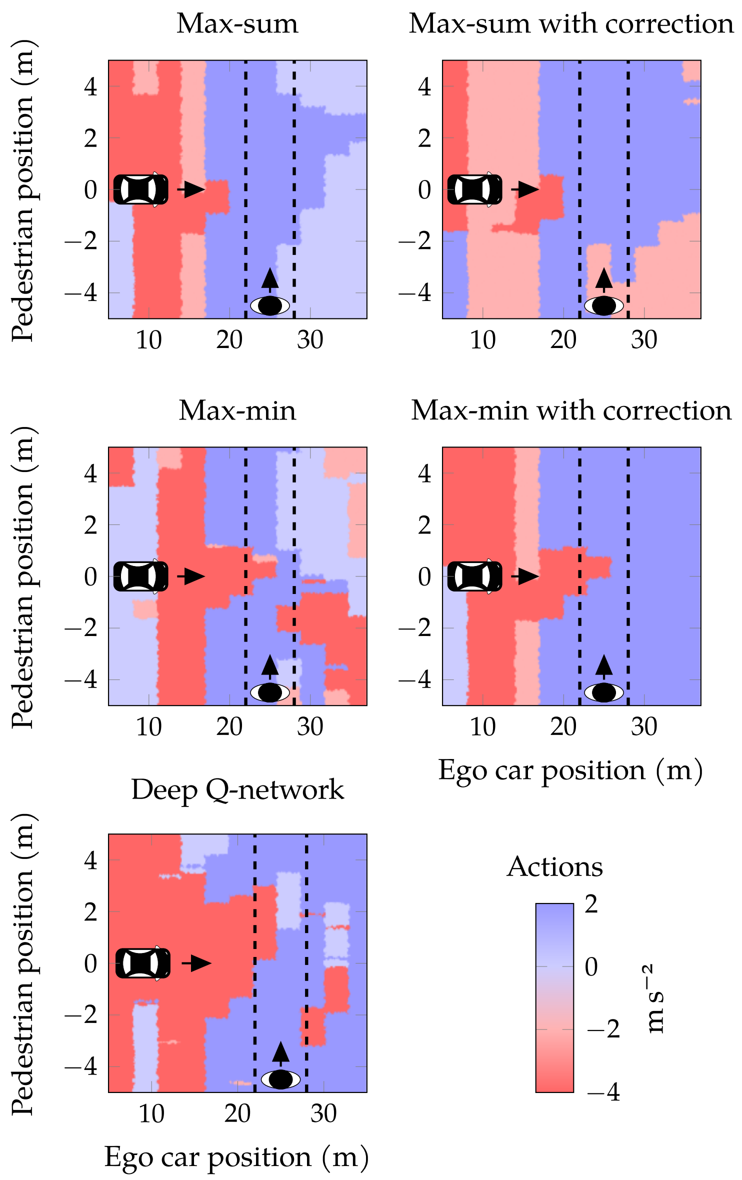


Figure 6.5: Visualization of the policies from the decomposition methods, the policy correction approach and using a deep Q-network. It assumes that the velocity of the car is 6 m s^{-1} and that the pedestrian is not moving. The color at a given point represents the action the ego car would take if it is at a certain position along the road and the pedestrian is at a certain position along the crosswalk.

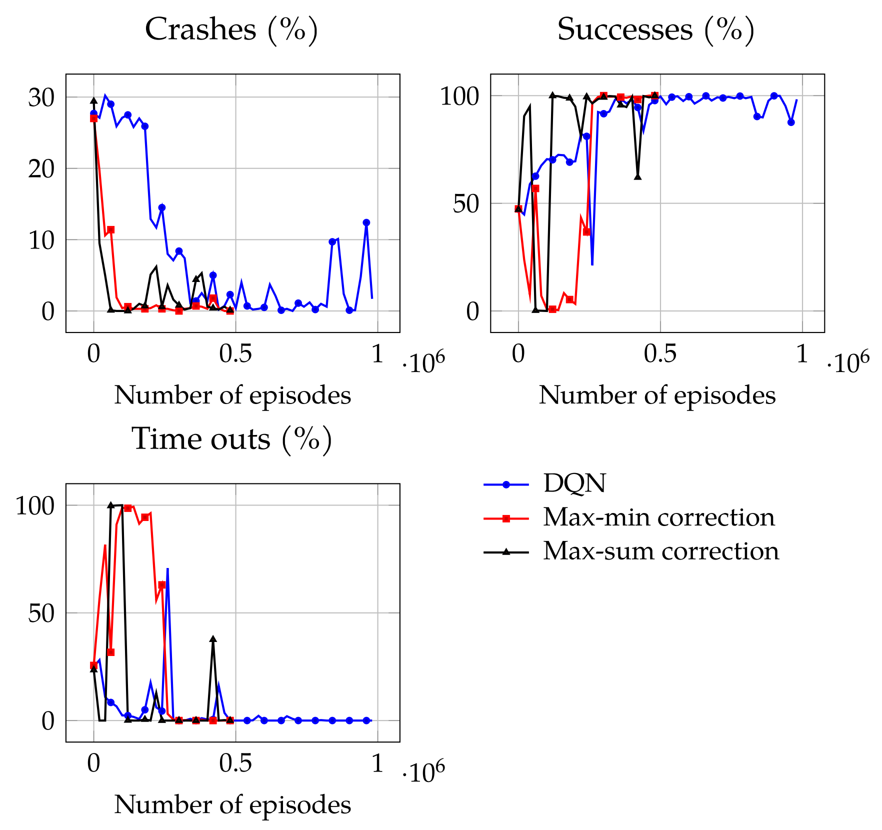


Figure 6.6: Evaluation of the policies performance throughout training. The correction function is being trained on half as many samples as the regular DQN policy but still converges and outperforms the regular DQN policy for both choices of the decomposition method (max-sum or max-min).

training than the other policies, and the three metrics converge after about three hundred thousands episodes. The networks achieving these performances result from the hyperparameter search on the exploration schedule. The corresponding values are reported in Table 6.5. When training the corrective term, the amount of exploration required for good performance is an order of magnitude lower than training DQN on the global problem.

To further analyze the advantages of the deep corrections method for learning efficient policies, we compared the performance of the deep corrections, DQN, and the value decomposition network (VDN) approach proposed by Sunehag, Lever, Gruslys, *et al.*[90]. For the deep correction method, only 50,000 training samples

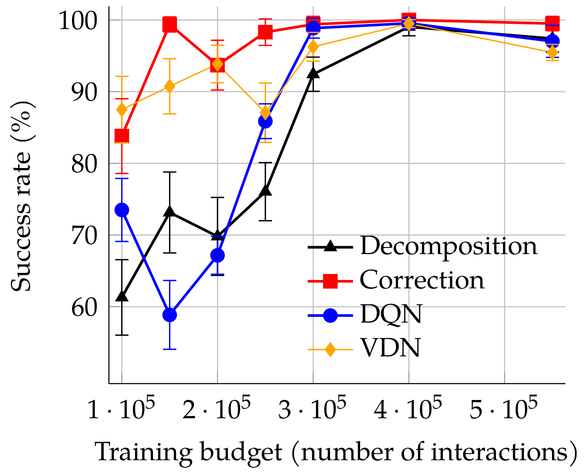


Figure 6.7: Performance of the deep corrections method, standard DQN and the value decomposition network approach [90] for different training budget. Each point is averaged over five trained policies initialized using different random seeds. The bars represent the standard error. The success rate is computed performing 1,000 simulations with randomized initial conditions.

are used to train the correction network while the remaining of the training budget is used to pre-train the policy resulting from the decomposition method. We can see that for low training budget, our approach and VDN significantly outperform standard DQN. Moreover the deep corrections algorithms outperforms VDN in most cases. In contrast with VDN, the deep corrections network does not assume any particular structure and should be more expressive. As the training budget increases, all the methods approach the optimal solution. There is also a smaller standard error when using the correction network than when using any of the other approaches. The source code to reproduce these experiments is available at `https://github.com/MaximeBouton/DeepCorrectionsExperiments.jl`.

6.5 Discussion

Utility fusion methods efficiently find approximate solutions to decision making problems in multiple agents problems or when a single agent interacts with multiple entities. Although computationally efficient, combining the utilities with simple functions, such as max-sum and max-min, leads to a suboptimal solution. To overcome this problem, we presented a novel technique to gear an existing suboptimal policy towards the optimum by learning a correction term. The correction term is represented by a neural network and is learned using deep Q-learning. This method inspired from multi-fidelity optimization can significantly improve a policy computed with the decomposition method. We verified this statement empirically on a fisheries management problem and an occluded crosswalk scenario. Li, Egorov, and Kochenderfer successfully applied the deep correction algorithm for collision avoidance in dense airspace [98]. Adding the corrective factor leads to a much more efficient policy than using the decomposition only. Our method outperformed a deep Q-network policy trained on the full scale problem. We also demonstrated that learning the corrective factor can be done with far fewer training samples than directly learning the value function of the full scale problem.

In the future, more sophisticated multi-fidelity optimization techniques could be used to represent the correction term. Rather than an additive correction, we could try a multiplicative term [96]. Another possibility involves learning the function used for utility fusion itself [90], [91]. A straightforward extension would be to learn a linear weighted combination of the utilities from the single entity problem. Our approach could be used jointly with the VDN technique where the VDN solution serves as a low fidelity approximation that is being corrected. Another direction for future research is to explore the generality of the method to correcting policies coming from potentially different solving techniques such as an offline POMDP solver.

Thus far, the methods presented in this thesis focused on efficient algorithms for generating decision policies in partially observable environments. The objective of the policy was specified using a reward function and by tuning the costs in this function we were able to generate pareto frontiers (Figures 3.4, 5.4 and 6.4). Although this framework is useful to compare policies and reason about the safety efficiency trade-off, tuning the reward function is a tedious procedure. In addition, those approaches do not provide any safety guarantees about the policy. The next chapter propose methods to restrict the search space to policies satisfying a safety criteria. This approach allows to decouple the safety and efficiency objectives.

7 Learning with Safety Constraints

This chapter introduces a method to enforce probabilistic safety constraints on an RL agent. A probabilistic model checker is used to compute the probability of succeeding a maneuver in every state of the environment. The information from the model checker is used to constrain an RL agent to only take reasonably safe actions. This approach allows to reduce the search space to safe policies which helps training and simplifies the reward design. The work presented in this chapter is based on research conducted with Jesper Karlsson, Alireza Nakhaei, Kikuo Fujimura, and Jana Tumova [63].

7.1 Related Work

In previous chapters, we have seen that reinforcement learning can be used as a powerful method to automatically derive suitable decision strategies in uncertain environments. Although RL algorithms can handle complex environment and result in strategy with high expected return, no guarantees can be made about its performance. Moreover, in many applications, the reward function might be difficult to design, causing the resulting behavior to be misaligned with the desired objective.

Learning safe policies using reinforcement learning is challenging. Previous approaches attempted to generate safe policies by tuning a penalty for collision in the reward function (Chapter 3). The reward fucntion can be used to balance conflicting objectives such as safety and efficiency by varying the cost associated with collisions. Metrics, such as time to reach the goal and collision rate, are then used to choose a suitable operating point. Given a reward unction, the policy is empirically evaluated through simulations. Empricial validation is hard because it requires an enormous number of simulations to conver every possible situation that might cause unsafe behavior [6].

Formal methods provide tools to verify a given policy with high confidence but often rely on very strong modeling assumptions. A possible approach involves verifying the policy after the planning or learning phase but requires restarting the planning if an issue is found [99]. Another method is to derive a policy from a boolean requirement rather than a reward function. Wang, Schaul, Hessel, *et al.* propose an online POMDP algorithm to solve such problems [55]. However, due to tractability issues, they are limited to small discretized problems making them unsuitable for practical use im complex environments.

Many approaches to derive safe policies in a reinforcement learning context consist of constraining the policy space to known safe actions [100]–[102]. Mukadam, Cosgun, Nakhaei, *et al.* propose to use ad-hoc knowledge to design a rule-based system that would prevent an RL agent to take unsafe actions [103]. Fulton and Platzer demonstrated a reinforcement learning algorithm to derive provably safe policies has been demonstrated for hybrid systems [100]. The authors propose the use of a model monitor that allows for taking exploratory actions when the execution environment is different from the training or planning environment. This approach was successfully applied to a simple autonomous driving scenario, but is is unclear how it would scale to more complex scenarios and it only handles a specific set of semantics for specifying the objective.

In this chapter, we consider linear temporal logic (LTL) [104] as the framework for specifying the objective. LTL formulas can mathematically express objectives formulated in structured English [105] which makes it an interesting framework for to mathematically express specifications derived by regulators, policy makers or governmental agencies. Existing approaches involving LTL formulas and monitoring RL agent rely on a deterministic abstraction fo the environment [102]. In contrast with previous work, our framework handles probabilistic specifications that accommodate less conservative policies. The resulting policy exhibits safe behavior according to a user-defined confidence level. We discuss its application to urban navigation scenarios with cars and pedestrians.

7.2 Probabilistic Model Checking

In this section we present the first part of our algorithm which consists in identifying at each state the set of actions that satisfy a desired specification. The specification is expressed using an LTL formula and a threshold on the probability of satisfaction. For example, in an autonomous driving scenario, one could specify the formula $\mathsf{G}\neg\texttt{collision}$ with a thresholde of 0.99, which states that the probability that a collision never occurs should be greater than 0.99. The threshold can be interpreted as an acceptable level of risk.

The task of computing the probability of satisfying an LTL formula for every state action pairs of an MDP is known as *quantitative* model checking in opposition to *quantitative* model checking which consists of finding states from which there is a probability of 1 of satisfying the formula.

7.2.1 Linear Temporal Logic

LTL is an extension to propositional logic with temporal operators. An LTL formula is build of atomic propositions according to the following grammar:

$$\phi ::= p \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi \mid \mathsf{G}\phi \mid \mathsf{F}\phi \mid \phi_1 \mathsf{U} \phi_2 \mid \mathsf{X}\phi \tag{7.1}$$

where p is an atomic proposition, ϕ, ϕ_1, and ϕ_2 are LTL formulas, $\neg$ (negation), $\wedge$ (conjonction), $\vee$ (disjunction) are the standard logical operators, and G (globally), F (eventually), U (until), and X (next) are temporal operators [106].

In this thesis we use LTL as a language to specify the objective of the agent. For example, safe-reachability objectives: "avoid state A and reach state B" are specified by the formula $\neg A \mathsf{U} B$. Persistent tasks: "keep visting A" are represented by the formula $\mathsf{GF}A$.

The satisfaction of an LTL formula is evaluated on an infinitely long trajectory in the environment. A labelling function maps each state of the environment to the set of atomic propositions holding in that state. The satisfaction of the formula can be verified by analyzing the sequence of atomic propositions generated by a trajectory. Even if the trajectory is continuous in time, the sequence of atomic propositions needs to be discrete.

When performing model checking, a convenient approach is to label the states of the MDP and express the property we wish to verify in terms of these labels. The labels are atomic proposition that evaluates to true or false at a given state. We augment the standard definition of an MDP presented in Chapter 2 with a labelling function to form a *labelled MDP*: $(\mathcal{S}, \mathcal{A}, T, R, \Pi, L)$. Π is a finite set of atomic propositions and L is a mapping: $L : \mathcal{S} \to 2^{\Pi}$, returning the set of atomic propositions that evaluate to true in a given state. The other elements of the tuple are the same as in the previous definition. The LTL specification is formed using atomic propositions from the same set Π.

7.2.2 *Identifying Acceptable Actions*

We address the general problem of identifying which actions can be taken in a given state, to ensure that the probability of satisfaction of the LTL formula is above a threshold. We compute the probability of satisfying a given LTL property ϕ at a given state pair. The satisfaction of ϕ dependes on the policy followed by the agent. As a consequence, we compute the maximum probability of satisfying ϕ when starting in a state s and taking an action a: $\Pr^{\max}(s, a \models \phi)$ where the maximum is over all the possible policies. Existing model checking algorithms address the problem of computing $\Pr^{\max}(s \models \phi)$ [106]–[108]. Most of these approaches can be adapted to derive the desired state action quantity using the following relation between the two quantities: $\Pr^{\max}(s \models \phi) = \max_a \Pr^{\max}(s, a \models \phi)$.

One can notice the strong analogy with the value function and state action value function in an MDP (Chapter 2). A value iteration algorithm can derived to compute the probability of satisfaction in every state action pair. For any LTL formula, computing these probabilities can be reduced to a reachability problem [106]. A reachability problem consists of computing the maximum probability of reaching a set of states $\mathcal{B}$. The set $\mathcal{B}$ is often expressed in terms of the states labels. The reachability problem is then solved using value iteration as presented in algorithm 7.1. We can see the direct analogy between line 6 of algorithm 7.1 and the Bellman update described in Chapter 2.

```
1: input: a labelled MDP, a set of states $\mathcal{B}$
2: Initialize $P^{(0)}(s,a)$ to 1 if $s \in \mathcal{B}$, 0 otherwise
3: $k \leftarrow 0$
4: repeat
5:     For all state action pair $s$, $a$:
6:     $P^{(k+1)}(s,a) \leftarrow \sum_{s' \in \mathcal{S}} \Pr(s' \mid s,a) \max_a P^{(k)}(s',a)$
7:     $k \leftarrow k+1$
8: until convergence
```

Algorithm 7.1: Value Iteration algorithm to compute the maximum probability of reaching a set of state

Once the quantity $\Pr^{\max}(s, a \models \phi)$ is computed for each state action pair, we can identify acceptable actions that the agent can take without violating the specification. Given the desired threshold λ, the set of acceptable actions is given by:

$$Act(s) = \{a \mid \overset{\max}{\Pr}(s, a \models \phi) > \lambda\} \tag{7.2}$$

where λ is the acceptable threshold on the probability of success.

We can extract t he policy that ensures the maximum level of satisfaction: $\pi_S(s) = \arg\max_a \Pr^{\max}(s, a \models \phi)$. The quantity $\Pr^{\max}(s, a \models \phi)$ can be interpreted as follows: the maximum probability of satisfying ϕ when starting in state s and taking action a and then following π_S for the subsequent time steps. It is analogous the state action value function in MDPs.

The complexity of algorithm 7.1 is polynomial in the number of states and actions, and it can be parallelized. For specific subclasses of LTL formulas, the mapping to a reachability problem is not necessary, and it can be directly computed using variants of algorithm 7.1 that have the same convergence property and the same complexity [107]. More details on the reduction to the reachability problem will be given in Chapter 8. Algorithm 7.1 requires a model of the transition distribution of the MDP. For the result of the model checking to be useful in real worl problems, it is important that the model is conservative.

For discrete states and actions MDPs, algorithm 7.1 gives strong probabilistic guarantees on the performance of π_S. To address continuous environments such as autonomous driving scenarios, one can discretize the state space and represent it in a grid as presented in Section 5.2. The probability table can be computed for each state in the grid. Generalizing to other states can be done via multi-linear interpolation, neares neighbors, or other forms of approximation [23]. Strong probabilistic guarantees can still be drawn by including the error introduced by the local approximation in the threshold. Model-free problems can also be addressed by the approximation methods via sampling. Scaling algorithm 7.1 to large state spaces using global approximation has been partially addressed by recent works that use neural fitted Q-iteration to solve the reachability problem [109].

7.3 Monitoring a Reinforcement Learning Agent

The result of the model checking approach enables distinguishing between acceptable and non acceptable actions with high confidence. This ability can be used to monitor a reinforcement learning agent such that the probabilistic specification hold during training and during the execution of the resulting policy.

In order to maintain the desired property during training, the exploration process of the reinforcement learning algorithm is modified. The learning agent is constrained to select an action from the set given by Equation (7.2). When the set is not empty, the agent can use any exploration scheme like ϵ-greedy, to select among the available actions in the set. With a probability ϵ the agent selects the action that maximizes the state action Q function otherwise it selects a random action in the set. This exploration strategy does not prevent the agent visiting states where no actions are available. In this case, the exploration policy default to π_S which is the policy the most likely to satisfy the LTL specification. A pseudo-code of the constrained exploration strategy is provided in algorithm 7.2.

```
1: input:  a state $s$; the result from model checking: $\pi_S$, $Act$; the current $Q$
   function
2: if $Act(s) = \varnothing$ then
3:   return $\pi^*(s)$
4: else
5:   sample $r \sim \text{Uniform}(0, 1)$
6:   if $r < \epsilon$ then
7:     return random action from $Act(s)$
8:   else
9:     return $\arg\max_{a \in Act(s)} Q(s, a)$
```

Algorithm 7.2: Constrained ϵ-greedy exploration policy

After training, the policy is also constrained at test time. The trained agent selects the acceptable action that maximizes the Q function. It follows π_S if the set of acceptable action is empty.

Mention what kind of guarantees hold or not

The modified exploration strategy does not affect the convergence of the value function in the states that have a high probability of satisfying the specification. A whole part of the state space is avoided during training and the value function for these states will not be accurate. However, the policy is still reliable since the agent relies on π_S when reaching these non acceptable states. This exploration strategy can be applied to any generic reinforcement learning algorithm.

Safe reinforcement learning methods can be categorized into two approaches: modification of the optimality criterion or modification of the exploration process [100], [101]. The proposed technique focuses on modifying the exploration strategy. Another approach consists of modifying the reward function by adding a penalty when the agent violates the LTL specification. However designing a reward signal matching the goal expressed by the LTL formula might be very challenging. Moreover, a more complex reward signal is likely to make the training less efficient.

Our approach allows us to decouple some of the objectives of the problem, which can help simplify the reward signal. The reward function used in reinforcement learning might express a different objective than the one specified through the LTL formula. Our strategy can be interpreted as a constrained optimization problem. We first identify the space of policies where a desired LTL specification is satisfied and then find the policy in this space that maximizes the accumulated reward. The constraints (*e.g.* safety) are encoded in the LTL formulas and handled using the model checking approach. Decoupling the objectives of the problem can help the training significantly as demonstrated in **??**. For example, if a safety constraint is handled by the model checking approach, then the RL agent will only explore safe states, and will not have to learn how to avoid unsafe regions.

7.4 Safety Constrained Reinforcement Learning at Intersections

In this section we demonstrate the model checking approach combined with the constrained reinforcement learning algorithm to learn a self policy to navigate an urban intersection.. We provide an empirical comparison of our method against different baselines.

7.4.1 Model of the environment

To demonstrate the approach we focus on an urban intersection navigation scenario illustrated in Figure 7.1. The ego vehicle must achieve a left-turn in the intersection safely and efficiently. We consider only two other traffic participants: a vehicle and a pedestrians. The two other participants have uncertain behaviors and interact with each other and with the ego vehicle according to simple rule-based policies. The ego vehicle must anticipate their behaviors and maintain safety while trying to reach a goal position as fast as possible. The pedestrian can cross at any of the three crosswalks. The other car comes from the horizontal street, from the left or from the right, and can either turn or go straight. In this scenario, our desired probabilistic specification will be expressing safety. A maximum acceptable level

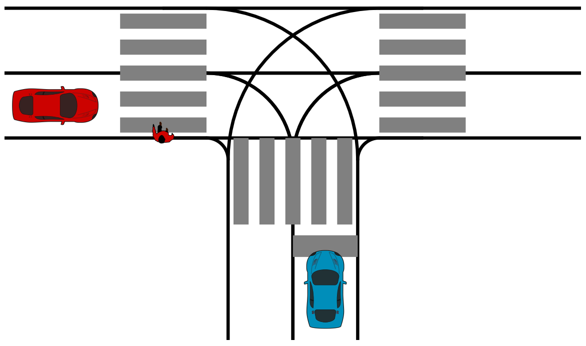


Figure 7.1: The ego vehicle (in blue) must achieve a left turn in an unsignalized intersection involving one other vehicle and one pedestrian interacting with each other.

of risk is imposed on the ego vehicle. We consider three subproblems of various complexity based on the number of agents to avoid: a scenario with one single pedestrian, one with one single car and a scenario with both a car and a pedestrian. This last scenario allows to capture the interaction between the vehicle and the pedestrian.

The intersection is modeled as a Markov decision process, simarly as the problems described in Chapter 2 and Section 5.2. The state of the environment consists of the physical state of the three traffic participants; the ego vehicle, the human driver and the pedestrian. The physical state is described in lane relative coordinates. It consists of the longitudinal position, the longitudinal velocity, the lane, as well as the route. The route indicates the starting position of the traffic participant and its goal, it encodes vehicles intentions such as turning right. In this section we assume that the route is perfectly observable. The other car can take one of four possible routes according to the topology of the intersection: straight from the left, straight from the right, turn left, turn right. A physical state can also take the specific value s_{absent} to model a potential incoming vehicle or pedestrian. The dynamics are the same as described in Section 5.2 at the exception of the policy of the other vehicle which reacts to the pedestrian and the ego vehicle according to the following rules:

- always yield to the pedestrian if it is crossing

- give priority to other vehicles when turning left and rely on the time to collision to decide when to cross
- follow the Intelligent Driver Model otherwise [28]

This simple strategy outputs an acceleration, at which is added a random amount drawn uniformly from the set $\{-1\,\mathrm{m\,s^{-2}}, 0\,\mathrm{m\,s^{-2}}, 1\,\mathrm{m\,s^{-2}}\}$. When the other car, or the pedestrian, is in the state s_{absent}, there is a probability of 0.7 that it appears at the beginning of a lane, or at a crosswalk, at the next time step. They appear with a random velocity, between $0\,\mathrm{m\,s^{-1}}$ and $8\,\mathrm{m\,s^{-1}}$, and with a random route (turn or straight) for the car.

Formulating a joint problem with three agents allows to capture interactions. In this example, the other vehicle is interacting with the pedestrian and with the ego vehicle. It is expected that leveraging this interaction will result in a more efficient policy.

We provided the following specification: $\phi = \neg \texttt{collision} \mathbf{U} \texttt{goal}$ with a threshold of 0.9999, where `collision` evaluates to true when the ego vehicle collides with another traffic participant and `goal` is verified when the ego vehicle achieves the left turn and reaches a position 20 m from the intersection. Satisfying this specification means avoiding any collisions.

7.4.2 Computing a safe policy

In order to compute a safe policy that satisfies the specified risk tolerance, we first run algorithm 7.1 to compute $\Pr^{\max}(s, a \models \phi)$. For the simple safety specification provided, the set $\mathcal{B}$ corresponds to the set of goal states. Collision states are terminal states and the probability of reaching the goal state from these states is zero.

Before applying algorithm 7.1, the continuous state space must be discretized. The position resolution is set to 2 m and the velocity resolution is set to $2\,\mathrm{m\,s^{-1}}$ for vehicles and $1\,\mathrm{m\,s^{-1}}$ for pedestrians. This discretization leads to 204 physical states for the ego vehicle, 793 for the other car, 145 states for the pedestrian.

When computing Equation (7.2) in the continuous space environment, we use multi-linear interpolation to map the continuous state of the traffic participants to a state on the grid [23]. The value at the continuous state is then approximated by a weighted sum of the value of the interpolants:

$$\Pr(s, a \models \phi) \approx \sum_{s_i \in \text{Interpolants}} w_i \Pr(s_i, a \models \phi) \tag{7.3}$$

where w_i is a weight proportional to the distance between the continuous state s and the interpolant s_i in the grid. The multi-linear interpolation is carried independently for each traffic participant on a two dimensional grid corresponding to longitudinal position and velocity. The error introduced by this approximation can be compensated by a more conservative threshold.

7.4.3 *Training with constrained exploration*

The solution from the model checking part is then used to constrain the exploration of the agent in a reinforcement learning algorithm. We used the Deep Q Learning algorithm [18] with prioritized experience replay [56]. To make the state variable a suitable input for the neural network we converted them to Cartesian coordinates, each vehicle is represented by four dimensions (two for the position, one for the heading, one for the velocity). Each variable is normalized. The four routes are represented by a one hot encoding. For the single pedestrian, the input dimension is eight. For the single car problem, it is a twelve dimensional vector since the possible route of the other vehicle must be considered. We used three fully connected layers of thirty-two nodes for the neural network architecture for all scenarios and the same hyperparameters as those reported in Chapter 6.

7.4.4 *Evaluation*

The results of 10,000 Monte Carlo simulations are reported in Table 7.1. We evaluated the policies regarding two metrics: the collision rate and the average number of time steps to reach the goal position. Our approach is compared to different baselines. We adopted the following terminology:

Safe Random: At each state, the agent chooses a random action among the set $Act(s)$. This policy is expected to satisfy the LTL specification while being very inefficient with respect to the reward function. These results serve as an empirical verification of the model checking algorithm.

Rule Based: The ego vehicle follows a handcrafted rule-based policy which is similar to the one followed by other vehicles in the simulation environment. This policy does not result of any optimization procedure but provides an assessment of the difficulty of the environment.

RL: A policy is trained to maximize an accumulated reward function. In addition to the reward for reaching the goal, the agent is penalized for colliding with other traffic participants. This is a typical RL procedure, the policy is expected to be efficient but will not necessary satisfy the specification. It is optimizing a simple reward function attributing a reward of 1.0 for reaching the goal, a high penalty for colliding with another agent, and a smaller penalty for each action. The collision penalty was varied between -1.0 and -5.0 and the action cost between -0.05 and 0.0. By increasing the collision cost, the policy is more risk averse and more likely to satisfy the specification. In contrast, increasing the action cost makes it reach the goal faster, at the expense of a greater risk.

Safe RL: The policy resulting from training a reinforcement learning agent constrained to acceptable actions. The reward function used for training is similar as the one described above. However, since the agent is constrained to safe action only, it will never receive the penalty associated to collision. It is only optimizing for reaching the goal faster.

For the scenario involving both a car and a pedestrian we approximated the set of acceptable actions as the intersection of the sets given by solving the simpler problems of avoiding one car and one pedestrian individually. This approach is suboptimal since it does not leverage the interaction between the two traffic participants. When the intersection is empty, the ego vehicle default to a hard-braking action which is always safe in the scenario of interest. Another approach would be to identify acceptable actions in the joint problem directly. The state space of this problem is much larger and hence the computation of $\Pr^{\max}(s, a \models \phi)$ would be more expensive.

Table 7.1: Performance on the different scenarios

	Collision rate (%)	Time steps
Single Pedestrian		
Safe Random	0.00	77.06
Rule Based	0.05	41.83
RL	0.63	22.71
Safe RL	0.00	17.24
Single Car		
Safe Random	0.00	76.28
Rule Based	2.18	26.62
RL	0.2	14.8
Safe RL	0.00	22.44
Car and Pedestrian		
Safe Random	0.00	87.304
Rule Based	3.62	42.05
RL	0.96	22.16
Safe RL	0.00	28.47

In Table 7.1 we can see that all the policies relying on the model checking technique are safe. When taking random actions inside the set $Act(s)$ the agent can still navigate safely in the environment. This result provides an empirical proof that the safe RL approach will satisfy the specification even at early stage of training. Although no collisions occurred in ten thousand simulations, the probabilistic guarantee only ensures 0.9999 certainty on safety. With more simulations, collisions may occur. We believe that in our evaluation environment, this requirement is strong enough to prevent collisions.

The problem of interest is a multi-objective optimization problem where the agent must balance two conflicting objective: safety and efficiency. We show the Pareto frontier for the safe RL and RL policies in Figure 7.3. As the reward functions parameters are varied we can generate different operating point in the safety versus efficiency diagram. In Figure 7.3, the region of optimality is in the bottom left corner.

The rule based method is dominated both in safety and efficiency by the safe RL

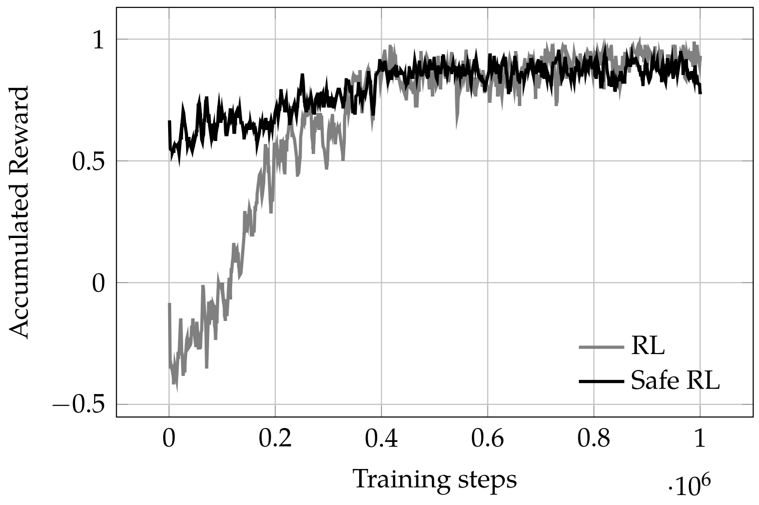


Figure 7.2: Evolution of the accumulated reward during training on the scenario with a car and a pedestrian. Both policies are trained with the same reward function: -1.0 for collision, 1.0 for reaching the goal and no action cost.

and RL policy. Although the RL policy provides the fastest time to cross, it was not able to achieve the safety objective. Across the different collision penalty that we tried, -1.5 gave the safest policy that did not result in an average time greater than the time achieved by safe random. Setting the collision cost to -5 resulted in the ego vehicle staying at its initial position.

As shown in Figure 7.3, the safe RL approach allows us to reach different operating points that are not accessible by tuning the reward function. The difficulty of trading off the two objectives by varying the weights in the reward function highlights the benefit of our approach. Since the safety objective is handled by the model checking step, the reward design is less challenging. In this problem, only efficiency needs to be encoded in the reward function for the safe RL policy.

7.4.5 *Effect on training performance*

The training performance of the safe RL and regular RL approaches is reported in Figure 7.2. The policies are trained with the same reward function. One million steps in the environment were used to ensure the convergence for both policies.

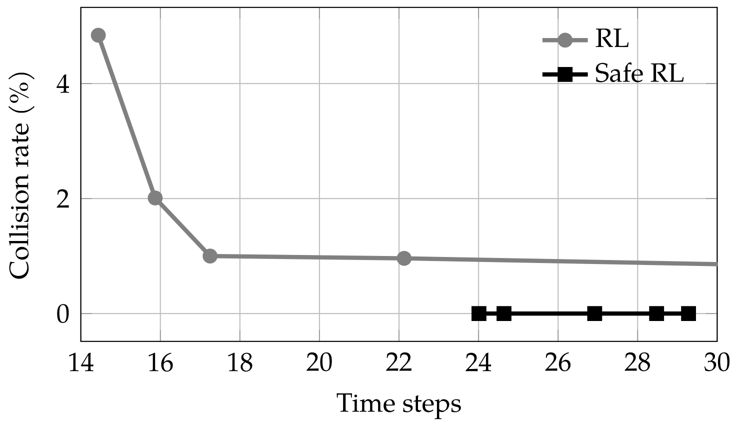


Figure 7.3: Pareto frontiers of the Safe RL technique and regular RL on the scenario with one car and one pedestrian. Different behavior are generated by changing the action cost and the collision cost in the reward function.

Figure 7.2 shows the performance of the policy throughout the training process for the scenario involving a car and a pedestrian. The safe RL policy did not collide during training, as expected from the performance of the safe random policy. Although the action space was constrained, the safe policy is able to accumulate as much reward as the non-safe RL agent.

7.5 Safe Reinforcement Learning with Scene Decomposition

In the previous section, we demonstrated a case study in a scenario involving only one car and one pedestrian. A natural follow up is to try to scale this approach to more complex scenes involving many cars and pedestrians as in Chapter 5.

7.6 Discussion

8 Approximate Model Checking

The previous chapter demonstrated that model checking can be used to constrain the action space of an RL agent to provide safe policies. The algorithm presented in the previous chapter for computed the probability of success in every state action pairs assumed that the agent has full observability. In this chapter, we extend the model checking procedure to partially observable environments. By formulating a planning problem, we show how to use point-based value iteration methods to efficiently approximate the maximum probability of satisyfing a desired logical formula and compute the associated belief state policy. We demonstrate that our method scales to large POMDP domains and provides strong bounds on the performance of the resulting policy. The research presented in this chapter has been carried in collaboration with Jana Tumova [110].

8.1 Related Work

Model checking in finite state MDPs has been studeid extensively and relies on two main solving strategies: value iteration and linear programs [106], [107]. These algorithms scale polynomially in the size of the MDP and efficient tools for probabilistic model checking can synthesize policies satisfying an LTL formula in MDPs with several millions states [111], [112]. However, these tools have little support for environments where the state is not observable, and current methods cannot scale to large POMDPs useful for robotics applications.

The general problem of finding a policy satisfying an LTL formula in an infinite horizon POMDP is undecidable [113], [114]. However, one can often compute approximate solutions by relaxing some aspects of the problem. A possible approach consists of restricting the space of policies to finite state controllers. This assumption can significantly reduce the search space. Chatterjee, Chmelik, Gupta, *et al.* propose an exact algorithm relying on some heuristics to find policies satisfying a formula with probability 1 [114]. This algorithm has been used to synthesize policies in a drone surveillance problem [115]. Other algorithms solve the quantitative model checking problem using parameter synthesis [116] or a variant of value iteration [117]. The restriction to classes of policies with a limited number of internal states allows those approaches to scale to domains with thousands of states. However, in many applications, finite state policies might not be expressive enough to solve the problem. Instead, the policy must be represented as a mapping from a belief state to an action.

Norman, Parker, and Zou addresses the problem of belief state planning with LTL specifications by discretizing the belief space and formulating an MDP over this space [118]. In problems where the state space has more than a few dimensions, discretizing the belief space becomes intractable. We demonstrate that our method scales to problems with an order of magnitude more hidden states. Similarly, abstraction refinement methods were proposed to discretize the belief space in linear Gaussian POMDPs [119]. Another approach for control in the belief space with LTL specifications linear Gaussian systems uses sampling based methods [120]. Wang, Chaudhuri, and Kavraki proposed an online search method to only explore belief points reachable from the current belief but their approach is limited to safe reachability objectives where the agent maximizes the probability of reaching a goal state while avoiding dangerous states [121]. Alternative methods can check that a given belief state policy satisfies a safety or optimality criterion using barrier certificates but do not allow for policy synthesis [122].

In this chapter, we demonstrate a method to synthesize policies mapping belief states to actions with an LTL specification in a POMDP. We show that we can benefit from the advances in POMDP planning algorithms demonstrated in Chapter 2 to solve model checking problems efficiently and avoid a naive discretization of the belief space. In contrast with previous work, we do not assume that the labels constituting the LTL formula are observable. In addition, our method handles stochastic observation models.

8.2 A Planning Approach to Model Checking

In this section we present our approach to solve the quantitative model checking problem in a POMDP. First, we demonstrate how to formulate a planning problem from a given model checking problem. Then we explain how to approximately compute a policy that maximizes the probability of satisfying a given LTL formula.

8.2.1 Problem formulation

The problem of interest consists of computing the maximum probability of satisfying a given linear temporal logic formula ϕ when starting in an initial belief point b in a POMDP.

Given a policy π, $\Pr^{\pi}(b \models \phi)$ represents the probability that a trajectory induced by π starting from belief b will satisfy the LTL formula ϕ. The quantity we wish to compute is expressed as follows:

$$\overset{\max}{\Pr}(b \models \phi) - \max_{\pi} \overset{\pi}{\Pr}(b \models \phi) \tag{8.1}$$

This probelm is a *quantitative* model checking problem as opposed to *qualitative* model checking problems which consists of finding a policy satisfying the formula with probability 1 [114].

Contrary to previous work, we consider that the propositions forming the LTL formula are defined over the states of the POMDP and hence are not observed by the agent. Instead, our formulation captures the information about the atomic propositions in the belief state.

8.2.2 *Reachability*

Point-based value iteration methods can scale to POMDPs with many thousands states [17], [24]. As explained in Chapter 2, these solvers have been designed to solve reward maximization problems. We explain how to formulate reachability problems as reward maximization problems so that we can use these solvers.

A reachability problem consists of computing the maximum probability of reaching a given set of states. If B is a propositional formula then the reachability problem corresponds to computing $\mathrm{Pr}^{\max}(b \models \mathsf{F}B)$ which reads as the maximum probability of reaching a state in which B holds when starting from belief b. For simplicity of the notation, we will also denote B, the set of states where the propositional formula expressed by B holds true. Note the similarity with the reachability problem for MDP described in Chapter 7. We derived a value iteration like algorithm to compute the probability for every state in an MDP. Similarly as we can derive a value iteration algorithm in the belief space, we can solve the reachability problem in the belief space. To make the analogy even clearer, we define a reward function such that the optimal value associated to this reward function corresponds to the probability of reaching B. In other terms, a reachability problem can be interpreted as a planning problem where the goal is to reach the set B. This problem is addressed by defining the following reward function:

$$R_{\text{Reachability}}(s,a) = \begin{cases} 1 & \text{if } s \in B \\ 0 & \text{otherwise} \end{cases} \tag{8.2}$$

In addition, the states in the set B are made terminal states and the initial value of $\mathrm{Pr}^{\max}(b \mid \mathsf{F}B)$ is initialized to 0 for any belief states. We can interpret the reachability problem as a reward maximization problem as follows:

$$\mathrm{Pr}^{\max}(b \models \mathsf{F}B) = \max_{\pi} \mathbb{E}[\sum_{t}^{\infty} R_{\text{Reachability}}(s_t, \pi(b_t)) \mid s_0 \sim b] \tag{8.3}$$

The right side of this equation corresponds to solving a POMDP planning problem with a value-based method [23]. The maximization is over the policy space. Note that in a POMDP, policies map belief states to actions rather than states to actions. The search problem becomes much harder than in MDPs and the value iteration algorithm can no longer scale. It has been proven that computing the maximum expected reward in a POMDP is undecidable [19]. Instead, we can rely on the approximate methods presented in Chapter 2 that scales to POMDP domains with tens of thousands of states,

8.2.3 *From LTL satisfaction to reachability*

In this section we present a technique to reduce any LTL satisfaction to reachability problems. This step mostly consists of defining a new POMDP such that solving the original quantitative model checking problem reduces to a reachability problem in this new model.

It is known that any LTL formula can be represented by a deterministic Rabin automaton [106], which can be defined as follows:

Deterministic Rabin Automata (DRA): A deterministic Rabin automaton is a tuple $\mathcal{R} = (Q, \Pi, \delta, q_0, F)$ where Q is a set of states, Π a set of atomic propositions, $\delta : Q \times 2^{\Pi} \to Q$ is a transition function, q_0 is an initial state, and F is an acceptance condition: $F = \{(L_1, K_1), \ldots, (L_k, K_k)\}$ where L_i and K_i are sets of states for all i.

A trajectory of a Rabin automaton is an infinite sequence of states $\tau = q_0 q_1 \ldots$, where $q_{i+1} = \delta(q_i, \sigma)$ for an input $\sigma \in 2^{\Pi}$. We say that a trajectory is accepting if there exists i such that: $\inf(\tau) \cap K_i \neq \emptyset$ and $\inf(\tau) \cap L_i = \emptyset$ where $\inf(\tau)$ is the set of states visited infinitely often in the trajectory. By converting the LTL formula into a DRA, we have a direct equivalence between accepting trajectories and trajectories satisfying the formula.

In general, converting an LTL formula into a DRA results in a finite state machine with a number of states double exponential in the number of atomic propositions in the formula. In practice, a lot of heuristics can be used to reduce the number of states in the automaton to a reasonable number. We give an example of the automaton

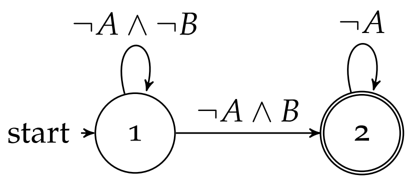


Figure 8.1: Illustration of an automaton generated by converting the LTL formula $\mathsf{G}\neg A \wedge \mathsf{F}B$. State 2 must be visited infinitely often to satisfy the formula. Each propositional formula on the edges represents possibly multiple transitions labeled with the subsets of atomic propositions that satisfy the formula on the edge.

resulting from converting $\mathsf{G}\neg A \wedge \mathsf{F}B$ in Figure 8.1.

Product POMDP: For a POMDP $\mathcal{P}$, and DRA $\mathcal{R}$, we define a product $\mathcal{P} \otimes \mathcal{R}$ as a POMDP:

$$\mathcal{P}' = (\mathcal{S} \times Q, \mathcal{A}, \mathcal{O}, T', O, L) \tag{8.4}$$

where the state space is the Cartesian product of the state space of $\mathcal{P}$ and $\mathcal{R}$ and the transition function satisifies:

$$T'((s,q),a,(s',q')) = \begin{cases} T(s,a,s') & \text{if } q' = \delta(q, L(s)) \\ 0 & \text{otherwise} \end{cases} \tag{8.5}$$

all the other elements of the product are the same as in the original POMDP. In the product, some transitions are prevented by the automaton. We can notice that the transition function defined is no longer a probability distribution. In practice, we can add an additional sink state such that if $\delta(q, L(s)) = \emptyset$, the system transitions in the sink state with probability 1. The new transition function ensures that trajectories that end up in the sink state are not accepted by the automaton (they are violating the specification).

Let aside the model checking problem, the construction of the product POMDP can be interpreted as a principled way to augment the state space in order to account for temporal objective. In addition, one can note that this state space extension is not always necessary. For formulas involving only a single until (U) or eventually (F) temporal operators, the problem can be directly expressed as a reachability problem and does not require a state space augmentation.

Once we formed a product POMDP, the next step consists of identifying a set of states B in the product POMDP, such that reaching a state in this set guarantees the satisfaction of the formula. We call those states *success states*.

From the definition of the DRA, we find that an infinitely long trajectory satisfying the formula must visit certain states infinitely often and others only finitely often. We first start to compute the sets of states that are visited infinitely often in the product POMDP, that is the maximal end component of a POMDP. More precisely, we need to find the maximal end components of the underlying MDP defined by $(\mathcal{S} \times \mathcal{Q}, \mathcal{A}, T')$. Starting from any state, with any policy, the agent will end up in a maximal end component if we consider infinitely long trajectories. Maximal end components can be computed by a graph algorithm that scales polynomially with the size of the state space [106]. Once the end components have been found, we must identify the success states.

Success States: Given a product POMDP $\mathcal{P}'$, its underlying MDP is noted $\mathcal{M}'$. A state contained in a maximal end component EC of $\mathcal{M}'$ is a success state if there exists an i such that $K_i \in EC$ and $L_i \notin EC$, where K_i and L_i results from the accepting conditions of the DRA used to form the product POMDP.

From the previous definition, we can conclude that from a success state, there is a probability of 1 of satisfying the LTL formula associated with the Rabin automaton. We can define a reachability reward function associated to the set of success states and compute the probability of success at a given belief point using Equation (8.3).

The first steps of the model checking approach (product POMDP and reduction to reachability) are identical for POMDPs and MDPs. They are independent of the structure of the observation space and are agnostic to partial observability. State uncertainty will play a role in the last step, which consists of solving the reachability problem.

Result: Given a POMDP and an LTL formula ϕ, the optimal value function of the product POMDP with the reachability reward function associated with the set of success states satisfies: $U^*(b) = \Pr^{\max}(b \models \phi)$, where b is a belief state in the product POMDP. In addition, there is a one to one mapping between the policy maximizing the value function in the product POMDP and the policy maximizing $\Pr(b \models \phi)$.

Proof Sketch: The construction of the product POMDP, and the definition of success states give the following:

$$\Pr^{\max}_{\mathcal{P}}(b \models \phi) = \Pr^{\max}_{\mathcal{P}'}(b \models \mathsf{F}B) \tag{8.6}$$

where on both sides, b is a belief of the product states, that is a belief over both the state of $\mathcal{P}$ and the state of the DRA associated with ϕ, and B is the set of success states in $\mathcal{P}'$. When updating the belief using Equation (2.3), the transition model from the product POMDP is used. Finally, Equation (8.3) holds from the construction of the reachability reward function and the definition of the belief state value function of a POMDP. More precisely, Equation (8.3) can be proven by formulating a belief state MDP [23] and use the equivalent result for MDPs [106].

The agent cannot observe whether it has reached an end component or not, but the belief state characterizes the confidence on whether or not it is in an end component. Previous works often assume that the end components are observed, our algorithm allows to relax this assumption by maintaining a belief on both the state of the environment and the state of the automaton.

8.2.4 Approximate Solution Techniques

The previous sections illustrated how to convert the quantitative model checking problem into a reward maximization problem. This section describes how to solve this problem using existing POMDP planning algorithms and how to interpret the convergence bounds with respect to the problem of interest. As we have shown, $\Pr^{\max}(b \models \phi)$ can be interpreted as a belief value function for a specific POMDP.

Solving POMDPs exactly is generally intractable [19], however approximation techniques have been developed. Approximation methods rely on restricting the policy space, either by considering finite-state controllers or alpha vector representations. Previous work addressed the problem of finding finite state controllers [114], [116]. This work focuses on alpha vector representations of the policy and the value function. An advantage of alpha vectors is that they can be used to represent both the policy and the value function. Hence, we can approximate the quantitative model checking problem and not only the policy synthesis problem.

In Chapter 2 we have described point-based methods, like SARSOP, which rely on alpha vector representations and provide state-of-the-art performance when solving discrete POMDPs offline. Point-based value iteration (PBVI) algorithms, often offer convergence guarantees specified in upper and lower bound on the value function. A precision parameter ϵ is provided and control the tightness of the convergence (by controlling the depth of the tree in SARSOP for example) which yields to:

$$|\overline{U^*(b_0)} - \underline{U^*(b_0)}| < \epsilon \tag{8.7}$$

Given a formula ϕ, we have show how to build a product POMDP in which we have the equivalence between the value function $U^*(b)$ and $\Pr^{\max}(b \models \phi)$. As a consequence, for a given precision parameter, we can directly translate the bounds on the value function in the product POMDP in terms of probability of success for our problem of quantitative model checking:

$$|\overline{\Pr^{\max}(b_0 \models \phi)} - \underline{\Pr^{\max}(b_0 \models \phi)}| < \epsilon \tag{8.8}$$

where $\overline{\Pr^{\max}(b_0 \models \phi)}$ is an upper bound over the actual probability of satisfaction, $\underline{\Pr^{\max}(b_0 \models \phi)}$ is a lower bound, and b_0 is the initial belief. With an infinite computation time, an arbitrary ϵ can be reached. However in practice only a minimum ϵ can be achieved within the computation budget. The original implementation of SARSOP relies on a discount factor. In this work, the discount factor is set to one such that the obtained value function matches exactly with the probability of satisfaction of the LTL formula.

The proposed methodology to solve quantitative model checking problems in POMDPs is agnostic to the planning algorithm. Although we focused the discussion on PBVI solvers, any belief state planner could be used. The strength of the guarantees are directly dependent on the choice of the underlying planning algorithm. For example, one could use the QMDP or FIB approximations to only compute an

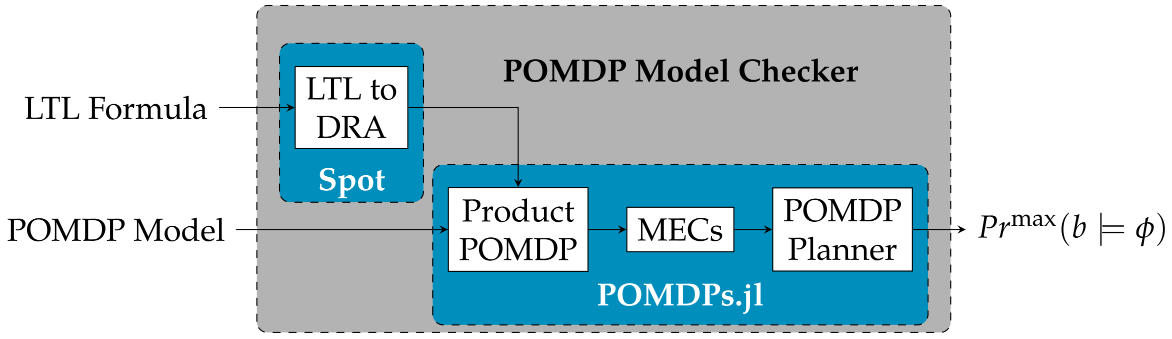


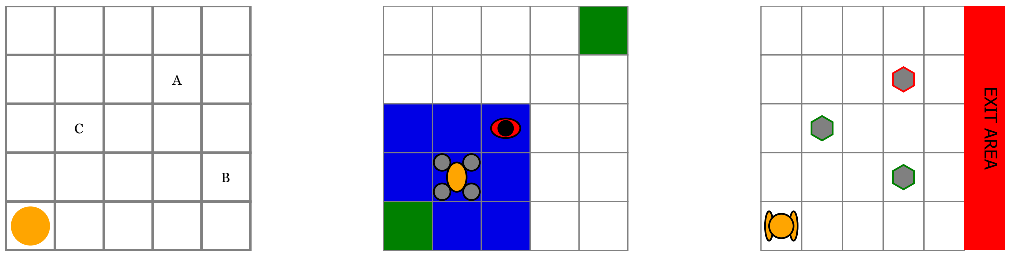
Figure 8.2: Architecture of our tool. We automate the process of converting quantitative model checking problem in POMDPs into reward maximization problems. This approach allows to benefit from existing POMDP planner implementation to efficiently solve model checking problems.

upper bound on the probability of success [20].

We provide a framework for efficiently solving quantitative model checking problems in POMDPs. Our tool is implemented in Julia [123] and builds upon POMDPs.jl, an existing POMDP planning library [57]. The architecture of the methodology is illustrated in Figure 8.2. Our framework takes as input a user defined POMDP problem and an LTL formula as a string. The process of building the product POMDP and the reward maximization problem is automated. We rely on the library Spot [124] to perform the conversion from LTL to DRA. In addition, POMDPs.jl provide a user-friendly and expressive interface to specify POMDPs. The user can use several planning algorithms such as SARSOP or FIB [20]. Our implementation allows the user to easily choose the underlying algorithm among the one available in POMDPs.jl [57].

8.3 Experiments

We evaluate our methodology on three discrete POMDP domains from the literature. The first one is a partially observable slippery grid world, the second one is the rock sample problem [125], and the third is a drone surveillance problem [115]. Those domains have a grid world like structure and can easily be scaled to different size of state and observation spaces to evaluate the scalability of our approach. Figure 8.3 illustrates the different POMDP domains we used to benchmark our model checking methodology.



(a) (b) (c)

Figure 8.3: Illustration of the domains used for experiments: partially observable grid world (a), drone surveillance (b), rock sample (c). The agent is represented in yellow. The size of grid can be varied in each domain to make the problem larger. The position of the rocks in rock sample, and of the labelled states in the grid world can also vary.

8.3.1 *Partially Observable Grid World*

This domain is an $n \times n$ grid with three labels: A, B, and C associated to some cells in the grid. The agent can choose to move left, right, up, and down. It reaches the desired cell with a probability of 0.7 and moves to another neighboring cell with equal probability otherwise. The state is represented by the position of the agent in the grid world. The agent receives a noisy observation of its position generated from a uniform distribution over the neighboring cells (vanish for distances greater than 1). The agent is initialized to a cell in the grid world with uniform probability. We investigated the following specifications:

- $\phi_1 = \neg C \mathsf{U} A \wedge \neg C \mathsf{U} B$: The agent must visit states A and B in any order while avoiding state C. This formula is a constrained reachability objective and does not require to form a product POMDP.
- $\phi_2 = \mathsf{G} \neg C$: The agent must never visit state C.

In this domain we fixed the precision of the solver to 1×10^{-2}.

8.3.2 Drone Surveillance

The drone surveillance problem is inspired by Svorenová, Chmelik, Leahy, *et al.* [115]. An aerial vehicle must survey regions in the corners of a grid like environment while avoiding a ground agent. The ground agent moves randomly and cannot reach the green cells. The drone can observe the location of the ground agent only if it is in its field of view delimited by a 3×3 area centered at the drone location. We labeled the states as A when the drone is in the bottom left corner, B when it is in the top right corner, and `det` when it can be detected by the ground agent (when it is on top of it). The drone is initialized in the bottom left corner while the agent is initialized to any cells but the two green ones. We considered two settings: restricted and unrestricted. In the restricted version, the ground agent is initialized outside of the field of view of the drone with equal probability, and cannot reach cells A and B. In the unrestricted version, the ground agent can reach those cells and is initialized with equal probability to any cells different from A. Our implementation is available at `https://github.com/JuliaPOMDP/DroneSurveillance.jl`. We analyzed one formula: $\neg \texttt{det} \mathsf{U} B$. The drone should eventually reach region B without being detected. Once region B is reached, the mission is over. Note that this is already a reachability objective and does not require the construction of a product POMDP. We set the precision is set to 1×10^{-2}.

8.3.3 Rock Sample

The rock sample problem models a rover exploring a planet and tasked to collect interesting rocks. The environment consists of a grid world with rocks at a known location as well as an exit area. The rocks can be either good or bad and their status is not observable. The robot can move deterministically in each direction or choose to sample a rock (when on top of it), or use its long range sensor to check the quality of a rock. The long range sensor returns the true status of a rock with a probability decaying exponentially with the distance to the rock. Our implementation is available at `https://github.com/JuliaPOMDP/RockSample.jl`. The problem ends when the robot reaches the exit area, this state is labeled as `exit`. In addition we defined two labels for situations when the robot pick a good rock or a bad rock respectively labeled `good` and `bad`. This paper considers three different formulas:

- $\phi_1 = \texttt{G}\neg\texttt{bad}$: This formula expresses that the robot should never pick up a bad rock. There exist a trivial policy that satisfies this formula which is to never pick up any rocks.
- $\phi_2 = \texttt{Fgood} \land \texttt{Fexit}$: This formula expresses that the robot should eventually pick a good rock and eventually reach the exit. Since the exit is a terminal state, the robot must pick up a good rock before reaching the exit. This policy cannot be satisfied with a probability 1 since there is a possibility that all the rocks present are bad.
- $\phi_3 = \texttt{Fgood} \land \texttt{Fexit} \land \texttt{G}\neg\texttt{bad}$: This formula is a combination of the two previous specifications. In addition of bringing a good rock and reaching the exit the robot must not pick a bad rock. A video demonstrating the resulting strategy is provided in the supplementary material.

For this domain, the precision of the solver is set to 1×10^{-3}.

8.4 Results

We applied the proposed methodology on different sizes of the proposed domains with different formulas. We use SARSOP as the underlying POMDP planning algorithm to solve the quantitative model checking problems. Note that our approach is agnostic to the choice of the planning algorithm and other methods could have been used. However, SARSOP is a good candidate for the task since it is one of the most scalable offline POMDP planners [17]. In addition, it provides strong bounds on the results, which can be translated into guarantees on the probability of success.

We compared the performance of SARSOP with the algorithm used by Norman, Parker, and Zou [118]. It consists of computing an upper bound by discretizing the belief space and performing Bellman backups on each of the belief points [126]. The main drawback of this algorithm is that the belief space is high dimensional (12545 dimensions for the largest rock sample), and the size of the grid grows exponentially. Figure 8.4 illustrates the benefits of using SARSOP instead of the Lovejoy algorithm. The discretization scheme is controlled by a granularity parameter m, the bigger m is, the more belief points are used. The Lovejoy line is obtained by varying m from 1 to 8, while the SARSOP line is obtained by specifying different precision targets. In the log scale figure, we can see that it takes much longer time to reach a given precision using the Lovejoy algorithm than SARSOP. In addition, we can see the exponential growth of the number of belief points. As a reference we added the precision given by QMDP [22] and FIB [20] which are two algorithms to compute upper bound on the value of a POMDP. Point-based methods provide both an upper and desired bound and allow the user to specify the precision. As a consequence there is no need to use an abstraction refinement mechanism to choose the right granularity of the belief space as done in previous work [118].

We empirically verify the correctness of the bound provided by SARSOP by simulating the resulting policy in the partially observable grid world with the formula $\neg C \mathsf{U} A \wedge \neg C \mathsf{U} B$. **??** illustrates the convergence of the estimated probability of success with the number of simulation of the policy. The probability of success is estimated using a Monte Carlo estimator. We can see that the estimated value converges towards the lower bound provided by SARSOP (dotted line). In this particular example, the value of the probability of success is around 0.90. The gap between the upper and lower bound provided by the solver can be controlled with the precision, in expense of a longer time to solve. **??** shows that the resulting policy has an empirical performance consistent with the lower bound given by SARSOP.

For the partially observable grid world, we visualize the probability of success obtained for the formula $\neg C \mathsf{U} A \wedge \neg C \mathsf{U} B$ in Figure 8.7. Since the probability is defined in the belief space, we create deterministic beliefs (probability of 1 of being in a given state (s, q)) for every possible state, and visualize the probability of each

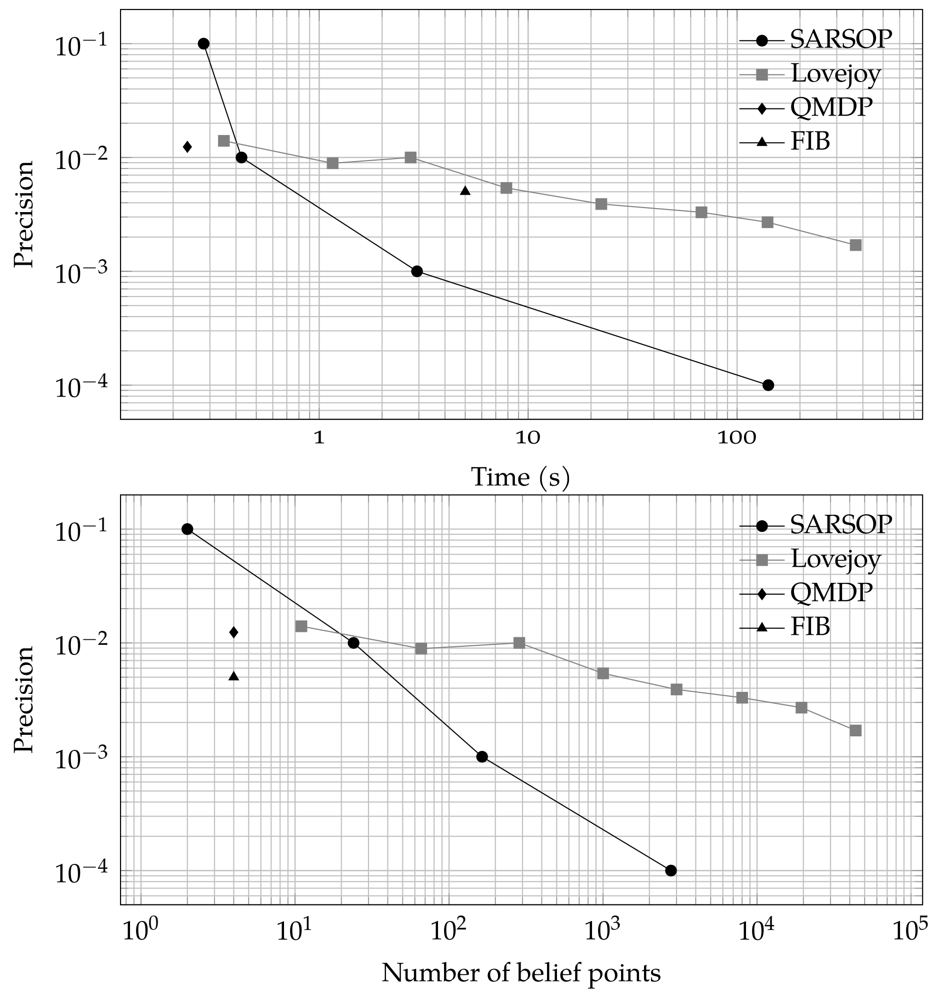


Figure 8.4: Illustration of the time precision trade-off for different algorithms providing upper bounds on the value function in a POMDP. Lovejoy is the algorithm used by Norman, Parker, and Zou. To compute the precision, we used the lower bound computed using SARSOP as a reference. The experiments are carried on a 3×3 partially observable grid world domain.

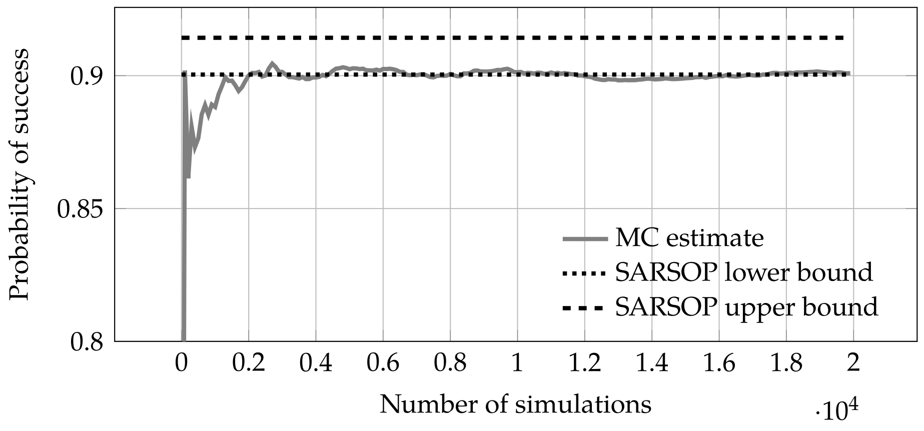


Figure 8.5: Estimate of the probability of success of a policy generated by SARSOP. We simulated 10,000 episodes estimated the probability of success. We compare this result with the upper and lower bound provided by SARSOP.

belief point. We can see that the heatmaps in Figure 8.7 are consistent with the automaton states from Figure 8.1. When state 4 is reached, the objective is achieved, and the probability of success is 1 in every grid world states. When in automaton state 2, state B has already been visited and the policy indicates to visit state A next to achieve the task. Similarly for the figure corresponding to being in automaton state 3.

Table 8.1 summarizes the performance of our approach in solving different tasks. In each case, we report the lower bound on $\Pr^{\max}(b_0 \models \phi)$ as well as the precision ϵ described in previous sections. The upper bound is the sum of the two. In addition, we report the solving time, it takes into account both the time to compute the maximal end components in the product POMDP as well as the time taken by SARSOP to solve the problem. The MEC column reports the time needed to identify the success states and construct the product POMDP (if needed). To control the number of iterations used by SARSOP, we used a threshold on the precision, ϵ *i.e.* after each iteration we check if the precision is lower than the threshold and return the policy and the probability of success if it is. The $|\Gamma|$ columns reports the number of belief points used by the point-based method. These results illustrate that our approach scales to POMDP domains with many thousands states and supports different LTL specifications. We can see from Table 8.1, that the model checker

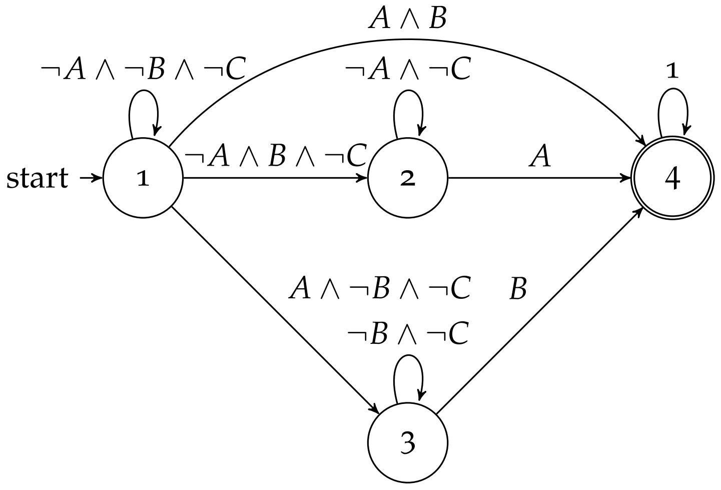

start
1
2
3
4
$\neg A \wedge \neg B \wedge \neg C$
$A \wedge B$
$\neg A \wedge \neg C$
1
$\neg A \wedge B \wedge \neg C$
A
$A \wedge \neg B \wedge \neg C$
B
$\neg B \wedge \neg C$

Figure 8.6: Illustration of an automaton generated by converting the LTL formula $\neg C \mathsf{U} A \wedge \neg C \mathsf{U} B$. State 2 must be visited infinitely often to satisfy the formula. Each propositional formula on the edges represents possibly multiple transitions labeled with the subsets of atomic propositions that satisfy the formula on the edge.

is able to provide an approximate solution in a reasonable time. In contrast with previous work [114], [115], solving a quantitative model checking problem instead of a qualitative problem allows us to find a policy even in cases where satisfiability cannot be guaranteed with probability 1. Moreover, our technique scales to larger state spaces.

In a few cases, the solver returned a policy with perfect precision in a very short time. This is the case for $\mathsf{G}\neg\mathsf{C}$ in grid world, and $\mathsf{G}\neg\mathtt{bad}$ in rock sample. In those two cases, the probability of success can be directly extracted from the maximum end components. In the grid world example, the whole grid world is a maximal end component. The state space is fully connected under any policy because of the probabilistic transitions. As a consequence, there exists no trajectory that would not eventually visit the state C in an infinite time. This problem does not have any success states. In the rock sample problem, the transition is deterministic, there exist many trivial policies to not pick a bad rock. The robot can just stay idle, or reach the exit. In those two examples, computing the maximum end component and performing one iteration of SARSOP is enough to solve the model checking

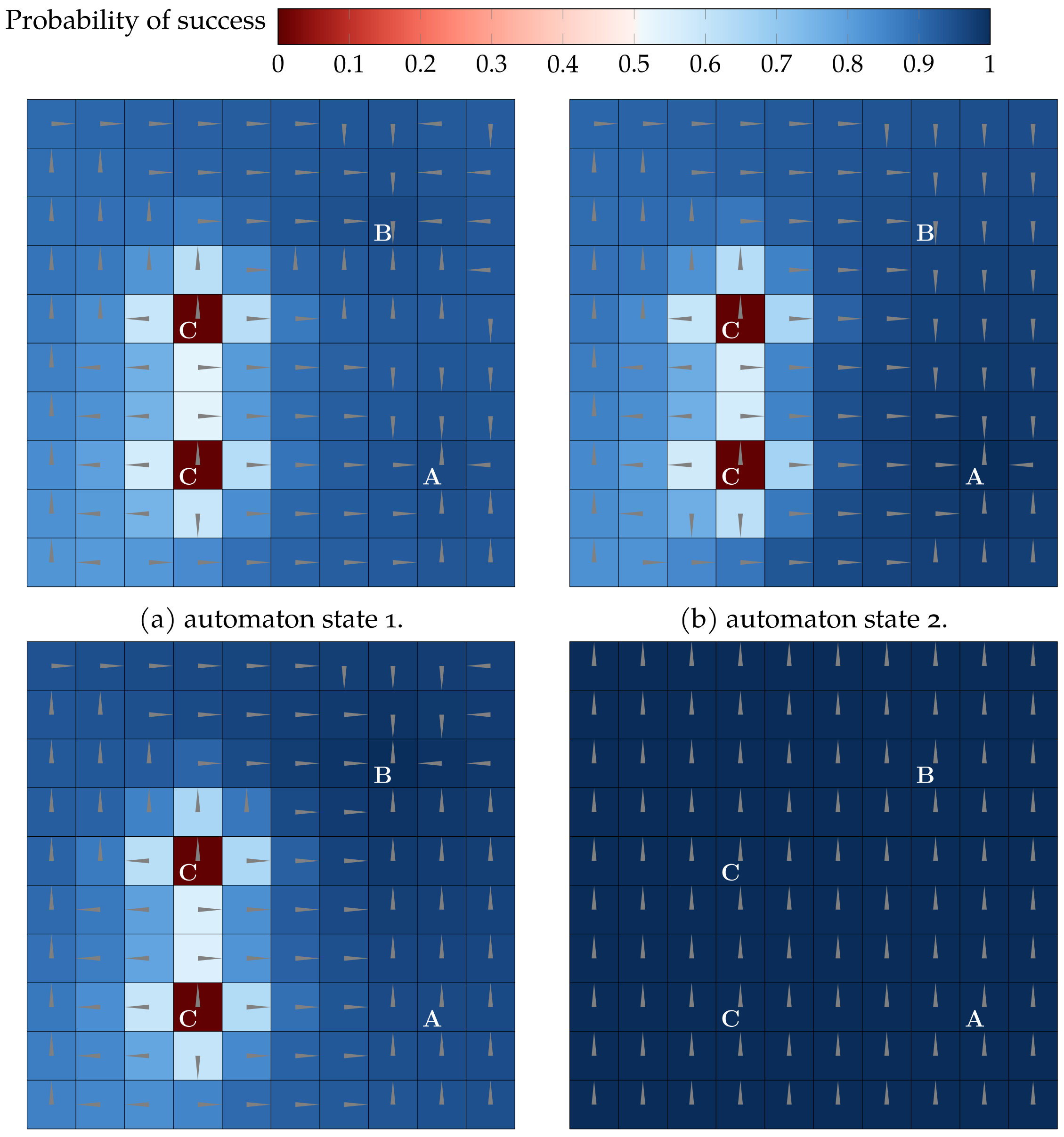


Figure 8.7: Visualization of the probability of success computed by SARSOP in the partially observable grid world with the formula $\neg C \mathcal{U} A \wedge \neg C \mathcal{U} B$. In this figure we assume a deterministic belief on each product state. The automaton state corresponds to the automaton from Figure 8.6.

Table 8.1: Performance of POMDP model checker.

Domain	$\|\mathcal{S}\|$ / $\|\mathcal{A}\|$ / $\|\mathcal{O}\|$	LB	ϵ	$\|\Gamma\|$	MEC (s)	Time (s)
PO Grid World						
$[10,10]$ ϕ_1	101 / 4 / 101	0.904	9.9×10^{-3}	3452	0.64	207.2
$[10,10]$ ϕ_2		0.0099	0	1	0.13	0.4
Drone Surveillance						
$[5,5]$	626 / 5 / 10	0.96	9×10^{-3}	4812	0.73	95.5
$[5,5]$ (U)		0.94	8×10^{-3}	4277	0.73	78.3
$[7,7]$ (U)	2402 / 5 / 10	0.94	1.9×10^{-2}	41799	4.8	12587.5
Rock Sample						
$[4,4]$ ϕ_1	65 / 7 / 3	1.0	0.0	1	0.03	0.02
$[4,4]$ ϕ_2		0.749	9.2×10^{-5}	13	0.09	0.3
$[4,4]$ ϕ_3		0.744	2×10^{-4}	23	0.10	0.4
$[5,5]$ ϕ_1	201 / 8 / 3	1.0	0.0	1	0.19	0.11
$[5,5]$ ϕ_2		0.879	2.8×10^{-4}	24	0.70	0.5
$[5,5]$ ϕ_3		0.865	9×10^{-4}	56	0.70	0.8
$[7,7]$ ϕ_1	12545 / 13 / 3	1.0	0.0	1	11.3	13.4
$[7,7]$ ϕ_2		0.990	9×10^{-4}	378	50.6	77.5
$[7,7]$ ϕ_3		0.979	9×10^{-4}	301	53.5	87.2

problem.

For the large version of the drone surveillance problem, the computation reached a maximum memory limit on the size of the policy and was not able to reach the desired precision. Although this problem is smaller than rock sample, the belief space has a much denser support. The drone maintains a belief over the location of the agent outside its field of view. This characteristic of the belief space makes this problem harder to approximate [127].

The solution provided by our approach is approximate. Although it provides mathematical bounds on the performance, it is not possible to compute the solution exactly. Reaching an arbitrary precision would require exploring the full belief space and take an infinite time. As a consequence, for smaller domains, approaches like the one proposed by Chatterjee, Chmelik, Gupta, *et al.* might be more suitable [114]. However, our approach does allow us to find approximate solutions in domains that were intractable for previous belief state approaches to model checking in

POMDPs. The formulation of the reward function in the product POMDP makes it a goal-oriented POMDP [128]. Our methodology would allow one to replace the POMDP planner by a goal-oriented POMDP solver. It would require extending the algorithm from Kolobov, Mausam, and Weld [128] to POMDPs. A comparison with traditional POMDP planners would be an interesting future direction. The dead end framework could be a useful theoretical framework to analyze the convergence of the solvers in the product POMDPs.

Contrary to previous work [118], we do not assume that the labels are observable. The computed policy maps a belief in the product space (POMDP state and automaton state) to an action. In problems where the automaton state is observable, our approach could still be applied and leverage this mixed observability assumption. This property would certainly help improve the results on the large drone surveillance problem. It has been shown that PBVI algorithms can scale to even larger domains when part of the state is fully observable [129].

8.5 Application to Autonomous Driving

The previous section demonstrated that our model checking methodology can apply to a broad range of problem. In this section we apply the approach to an autonomous driving scenario. From the previous chapter, we have seen how to model those scenarios as POMDPs such that we can apply PBVI techniques to maximize a reward function. One advantage of the framework is that we can use the exact same POMDP model, the handling of LTL formulas, and the conversion to a reachability problem is all done automatically.

We focus on the occluded crosswalk scenario presented in Section 5.2. The choice of PBVI solvers underlying the model checking algorithm enforces our POMDP model to be discrete. We use a similar discretization for the occluded crosswalk considering one single pedestrian as presented in Section 5.2. In this problem, a straight forward specification is to reach the goal without colliding which we can express LTL as follows: `!collisionUgoal` where `collision` is a proposition that holds true when the ego vehicle is in a collision state, and `goal` is a proposition that holds true if we have passed a certain longitudinal distance after the crosswalk (10 m). This specific class of formula represents a *safe reachability* objective. For the formula to be satisfied, the ego has to eventually reach the goal, trajectories that are safe but do not reach the goal violate the formula. Contrary to the reward based approach, no time constraints is encoded in this objective as the problem is not discounted.

Using our POMDP model checking methodology, we can compute the probability of satisfying the formula at a given belief point. We choose SARSOP as the underlying algorithm. For this experiment we set the precision to 1×10^{-3} and fixed the computation time to 1 h.

Figure 8.8 informs us of the performance of the policy. Contrary to Figure 5.2, the values of the plot represent a quantitative information about the performance. We can directly identify in which region of the state space the policy is likely to fail according to the assumed pedestrian model and observation model. Those areas are represented by the red zone. By setting a desired threshold on the probability of success, one could identify a safe set of beliefs. As uncertainty increases, the zone with low probability of success spreads out. This phenomena is consistent with the observations from Section 5.2.

8.6 Discussion

In this chapter we presented a methodology to solve quantitative model checking problems in POMDPs. Given an LTL formula and a POMDP model, our approach approximates the maximum probability of satisfying the formula as well as the corresponding belief state policy. We first convert the LTL formula into an automaton and construct a product POMDP between the automaton and the original POMDP model. By formulating a reward maximization problem, we have shown how to benefit from approximate POMDP planning algorithms to compute a solution to the model checking problem. Our method provides strong convergence bounds on the result. We have shown empirically that our approach applies to a variety of discrete POMDP domains, for different LTL formulas, and scales to larger problem than

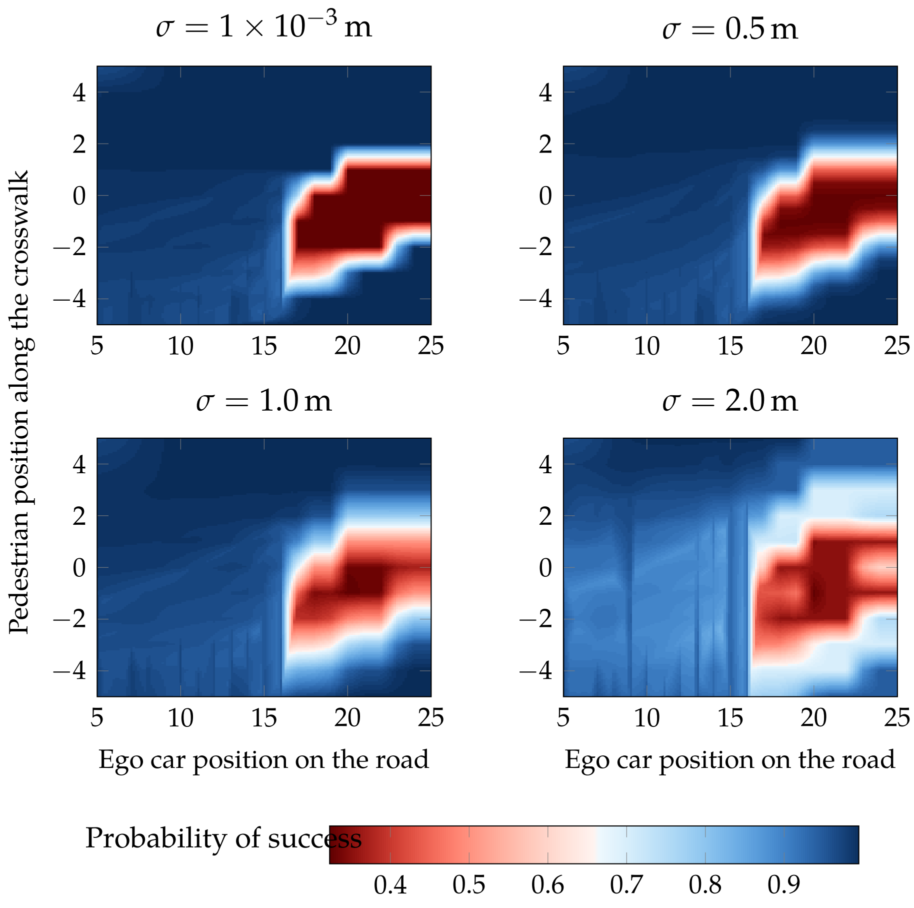


Figure 8.8: Visualization of the probability of success for different belief in the occluded crosswalk scenario. These plots assume a fixed ego velocity at 5.0 m s^{-1} and fixed pedestrian velocity at 1.0 m s^{-1}. Each point represents the probability of satisfying the formula, while being in a belief represented by a Gaussian distribution around the position of the pedestrian with standard deviation σ.

previous belief state techniques [115], [118]. We provide a Julia package for POMDP model checking available at `https://github.com/sisl/POMDPModelChecking.jl`.

The main limitation of the methodology is that it only applies to POMDPs with discrete state spaces. The two bottlenecks are the computation of the maximal end components and the choice of the planning algorithms. For some LTL formula, like constrained reachability [106], or if one is interested in policy synthesis only, the reward maximization problem can be formulated without having to compute maximal end components [130]. Our approach provides a flexible way to integrate LTL objectives in POMDP planning and allows to use any planning algorithm to allow a trade-off between convergence guarantees and scalability. Online POMDP planning algorithms could be used instead of PBVI methods to generate policies from an LTL objective at the price of lacking convergence guarantees.

9 Conclusions and Future Work

write conclusion

Bibliography

[1] World Health Organization, "Global status report on road safety," Tech. Rep., 2018. [Online]. Available: https://www.who.int/violence_injury_prevention/road_safety_status/2018/en/.

[2] National Highway Traffic Safety Administration, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," Tech. Rep. DOT HS 812 506, 2018. [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812506.

[3] National Highway Traffic Safety Administration, "Preparing for the future of transportation: Automated vehicles 3.0," Tech. Rep., 2018. [Online]. Available: https://www.transportation.gov/av/3/preparing-future-transportation-automated-vehicles-3.

[4] W. Schwarting, J. Alonso-Mora, and D. Rus, "Planning and decision-making for autonomous vehicles," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. 187–210, 2018.

[5] National Highway Traffic Safety Administration, "Federal automated vehicles policy: Accelerating the next revolution in roadway safety," Tech. Rep. 12507-102616-v10a, 2016. [Online]. Available: https://www.transportation.gov/AV/federal-automated-vehicles-policy-september-2016.

[6] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 1, pp. 15–24, 2016.

[7] D. Silver, *What hurdles do self-driving cars face as waymo gets ready for prime time?* Forbes.com, Ed., 2018. [Online]. Available: `https://www.forbes.com/sites/davidsilver/2018/10/05/what-hurdles-do-self-driving-cars-face-as-waymo-gets-ready-for-prime-time/`.

[8] B. Paden, M. Cáp, S. Z. Yong, D. S. Yershov, and E. Frazzoli, ''A survey of motion planning and control techniques for self-driving urban vehicles,'' *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[9] S. Kammel, J. Ziegler, B. Pitzer, M. Werling, T. Gindele, D. Jagszent, J. Schröder, M. Thuy, M. Goebl, F. von Hundelshausen, O. Pink, C. Frese, and C. Stiller, ''Team annieway's autonomous system for the DARPA urban challenge 2007,'' in *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*, 2009.

[10] L. Fletcher, S. J. Teller, E. Olson, D. Moore, Y. Kuwata, J. P. How, J. J. Leonard, I. Miller, M. E. Campbell, D. Huttenlocher, A. Nathan, and F. Kline, ''The mit-cornell collision and why it happened,'' *Journal of Fields Robotics*, vol. 25, no. 10, pp. 775–807, 2008.

[11] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, ''End to end learning for self-driving cars,'' vol. abs/1604.07316, 2016. arXiv: `1604.07316`.

[12] C. Chen, A. Seff, A. L. Kornhauser, and J. Xiao, ''Deepdriving: Learning affordance for direct perception in autonomous driving,'' in *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[13] A. Kuefler, J. Morton, T. A. Wheeler, and M. J. Kochenderfer, ''Imitating driver behavior with generative adversarial networks,'' in *IEEE Intelligent Vehicles Symposium (IV)*, 2017.

[14] H. Bai, D. Hsu, and W. S. Lee, ''Integrated perception and planning in the continuous space: A POMDP approach,'' *International Journal of Robotics Research*, vol. 33, no. 9, pp. 1288–1302, 2014.

[15] T. Bandyopadhyay, K. S. Won, E. Frazzoli, D. Hsu, W. S. Lee, and D. Rus, ''Intention-aware motion planning,'' in *Algorithmic Foundations of Robotics X - Workshop on the Algorithmic Foundations of Robotics, WAFR*, 2012.

[16] D. Silver and J. Veness, ''Monte-carlo planning in large POMDPs,'' in *Advances in Neural Information Processing Systems (NIPS)*, 2010.

[17] H. Kurniawati, D. Hsu, and W. S. Lee, ''SARSOP: efficient point-based POMDP planning by approximating optimally reachable belief spaces,'' in *Robotics: Science and Systems*, 2008.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, ''Human-level control through deep reinforcement learning,'' *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[19] O. Madani, S. Hanks, and A. Condon, ''On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision problems,'' in *AAAI Conference on Artificial Intelligence (AAAI)*, 1999.

[20] M. Hauskrecht, ''Value-function approximations for partially observable markov decision processes,'' *Journal of Artificial Intelligence Research*, vol. 13, pp. 33–94, 2000.

[21] E. J. Sondik, ''The optimal control of partially observable markov processes over the infinite horizon: Discounted costs,'' *Operations Research*, vol. 26, no. 2, pp. 282–304, 1978.

[22] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling, ''Learning policies for partially observable environments: Scaling up,'' in *International Conference on Machine Learning (ICML)*, 1995.

[23] M. J. Kochenderfer, *Decision making under uncertainty: theory and application*. MIT press, 2015.

[24] G. Shani, J. Pineau, and R. Kaplow, "A survey of point-based POMDP solvers," *Autonomous Agents and Multi-Agent Systems*, vol. 27, no. 1, pp. 1–51, 2013.

[25] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European Conference on Machine Learning (ECML)*, 2006.

[26] A. Couëtoux, J. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard, "Continuous upper confidence trees," in *International Conference on Learning and Intelligent Optimization (LION)*, 2011.

[27] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan, *Estimation with applications to tracking and navigation: theory algorithms and software*. John Wiley & Sons, 2004.

[28] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical review E*, vol. 62, no. 2, p. 1805, 2000.

[29] D. Hadfield-Menell, S. Milli, P. Abbeel, S. J. Russell, and A. D. Dragan, "Inverse reward design," in *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[30] S. M. Thornton, F. E. Lewis, V. Zhang, M. J. Kochenderfer, and J. C. Gerdes, "Value sensitive design for autonomous vehicle motion planning," in *IEEE Intelligent Vehicles Symposium (IV)*, 2018.

[31] M. Bouton, A. Cosgun, and M. J. Kochenderfer, "Belief state planning for autonomously navigating urban intersections," in *IEEE Intelligent Vehicles Symposium (IV)*, 2017.

[32] D.-K. Kye, S.-W. Kim, and S.-W. Seo, "Decision making for automated driving at unsignalized intersection," in *International Conference on Control, Automation and Systems (ICCAS)*, 2015.

[33] W. Song, G. Xiong, and H. Chen, "Intention-aware autonomous driving decision-making in an uncontrolled intersection," *Mathematical Problems in Engineering*, vol. 2016, 2016.

[34] T. Tram, A. Jansson, R. Grönberg, M. Ali, and J. Sjöberg, "Learning negotiating behavior between cars in intersections using deep q-learning," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2018.

[35] C. Hubmann, J. Schulz, G. Xu, D. Althoff, and C. Stiller, "A belief state planner for interactive merge maneuvers in congested traffic," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2018.

[36] H. Kurniawati and V. Yadav, "An online POMDP solver for uncertainty planning in dynamic environment," in *International Symposium on Robotics Research (ISRR)*, 2013.

[37] A. T. Schulz and R. Stiefelhagen, "A controlled interactive multiple model filter for combined pedestrian intention recognition and path prediction," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2015.

[38] A. F. Genovese, "The interacting multiple model algorithm for accurate state estimation of maneuvering targets," *Johns Hopkins APL technical digest*, vol. 22, no. 4, pp. 614–623, 2001.

[39] A. Goswami and C. S. G. Lee, "Design of an interactive multiple model based two-stage multi-vehicle tracking algorithm for autonomous navigation," in *IEEE Intelligent Vehicles Symposium (IV)*, 2015.

[40] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of sumo-simulation of urban mobility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, 2012.

[41] M. Bouton, A. Nakhaei, K. Fujimura, and M. J. Kochenderfer, "Cooperation-aware reinforcement learning for merging in dense traffic," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2019.

[42] P. Trautman and A. Krause, "Unfreezing the robot: Navigation in dense, interacting crowds," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[43] E. Ward, N. Evestedt, D. Axehill, and J. Folkesson, ''Probabilistic model for interaction aware planning in merge scenarios,'' *IEEE Transactions on Intelligent Vehicles*, vol. 2, no. 2, pp. 133–146, 2017.

[44] E. Schmerling, K. Leung, W. Vollprecht, and M. Pavone, ''Multimodal probabilistic model-based planning for human-robot interaction,'' in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[45] J. F. Fisac, E. Bronstein, E. Stefansson, D. Sadigh, S. S. Sastry, and A. D. Dragan, ''Hierarchical game-theoretic planning for autonomous vehicles,'' in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[46] S. Bansal, A. Cosgun, A. Nakhaei, and K. Fujimura, ''Collaborative planning for mixed-autonomy lane merging,'' in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[47] Z. N. Sunberg, C. J. Ho, and M. J. Kochenderfer, ''The value of inferring the internal state of traffic participants for autonomous freeway driving,'' in *American Control Conference (ACC)*, 2017.

[48] C. Dong, J. M. Dolan, and B. Litkouhi, ''Intention estimation for ramp merging control in autonomous driving (in review),'' in *IEEE Intelligent Vehicles Symposium (IV)*, 2017.

[49] D. Sadigh, S. Sastry, S. A. Seshia, and A. D. Dragan, ''Planning for autonomous cars that leverage effects on human actions,'' in *Robotics: Science and Systems*, 2016.

[50] P. Wang, C. Chan, and A. de La Fortelle, ''A reinforcement learning based approach for automated lane change maneuvers,'' in *IEEE Intelligent Vehicles Symposium (IV)*, 2018.

[51] M. Bouton, K. D. Julian, A. Nakhaei, K. Fujimura, and M. J. Kochenderfer, ''Decomposition methods with deep corrections for reinforcement learning,'' *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 33, no. 3, pp. 330–352, 2019.

[52] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[53] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California University of San Diego La Jolla Institute for Cognitive Science, Tech. Rep., 1985.

[54] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

[55] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2016.

[56] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *International Conference on Learning Representations*, 2016.

[57] M. Egorov, Z. N. Sunberg, E. Balaban, T. A. Wheeler, J. K. Gupta, and M. J. Kochenderfer, "Pomdps.jl: A framework for sequential decision making under uncertainty," *Journal of Machine Learning Research*, vol. 18, 26:1–26:5, 2017.

[58] M. Innes, "Flux: Elegant machine learning with julia," *Journal of Open Source Software*, vol. 3, no. 25, p. 602, May 3, 2018, ISSN: 2475-9066.

[59] Y. Hu, A. Nakhaei, M. Tomizuka, and K. Fujimura, "Interaction-aware decision making with adaptive strategies under merging scenarios," *arXiv preprint arXiv:1904.06025*, 2019.

[60] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[61] T. A. Wheeler, M. J. Kochenderfer, and P. Robbel, "Initial scene configurations for highway traffic propagation," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2015.

[62] M. Bouton, A. Nakhaei, K. Fujimura, and M. J. Kochenderfer, "Safe reinforcement learning with scene decomposition for navigating complex urban environments," in *IEEE Intelligent Vehicles Symposium (IV)*, 2019.

[63] M. Bouton, J. Karlsson, A. Nakhaei, K. Fujimura, M. J. Kochenderfer, and J. Tumova, "Reinforcement learning with probabilistic guarantees for autonomous driving," in *Workshop on Safety Risk and Uncertainty in Reinforcement Learning, Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018.

[64] M. Bouton, A. Nakhaei, K. Fujimura, and M. J. Kochenderfer, "Scalable decision making with sensor occlusions for autonomous driving," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[65] M. Schratter, M. Bouton, M. J. Kochenderfer, and D. Watzenig, "Pedestrian collision avoidance system for scenarios with occlusions," in *IEEE Intelligent Vehicles Symposium (IV)*, 2019.

[66] R. Alami, T. Simeon, and K. M. Krishna, "On the influence of sensor capacities and environment dynamics onto collision-free motion plans," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, 2002.

[67] M. Koschi, C. Pek, M. Beikirch, and M. Althoff, "Set-based prediction of pedestrians in urban environments considering formalized traffic rules," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2018.

[68] P. F. Orzechowski, A. Meyer, and M. Lauer, "Tackling occlusions & limited sensor range with set-based safety verification," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2018.

[69] S. Magdici and M. Althoff, "Fail-safe motion planning of autonomous vehicles," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2016.

[70] J. Miura, Y. Negishi, and Y. Shirai, "Adaptive robot speed control by considering map and motion uncertainty," *Robotics and Autonomous Systems*, vol. 54, no. 2, pp. 110–117, 2006.

[71] J. Rosenblatt, "Optimal selection of uncertain actions by maximizing expected utility," *Autonomous Robots*, vol. 9, no. 1, pp. 17–25, 2000.

[72] J. P. Chryssanthacopoulos and M. J. Kochenderfer, "Decomposition methods for optimized collision avoidance with multiple threats," *AIAA Journal of Guidance, Control, and Dynamics*, vol. 35, no. 2, pp. 398–405, 2012.

[73] H. Y. Ong and M. J. Kochenderfer, "Short-term conflict resolution for unmanned aircraft traffic management," in *Digital Avionics Systems Conference (DASC)*, 2015.

[74] S. J. Russell and A. Zimdars, "Q-decomposition for reinforcement learning agents," in *International Conference on Machine Learning (ICML)*, 2003.

[75] K. H. Wray, S. J. Witwicki, and S. Zilberstein, "Online decision-making for scalable autonomous systems," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

[76] R. E. Tompa and M. J. Kochenderfer, "Optimal aircraft rerouting during space launches using adaptive spatial discretization," in *Digital Avionics Systems Conference (DASC)*, 2018.

[77] B. Chen, D. Zhao, and H. Peng, "Evaluation of automated vehicles encountering pedestrians at unsignalized crossings," in *IEEE Intelligent Vehicles Symposium (IV)*, 2017.

[78] C. Ericson, *Real-time collision detection*. CRC Press, 2004.

[79] S. Brechtel, T. Gindele, and R. Dillmann, "Probabilistic decision-making under uncertainty for autonomous driving using continuous pomdps," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2014.

[80] H. Bai, S. Cai, N. Ye, D. Hsu, and W. S. Lee, "Intention-aware online POMDP planning for autonomous driving in a crowd," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[81] G. Tesauro, "Online resource allocation using decompositional reinforcement learning," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2005.

[82] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The complexity of decentralized control of markov decision processes," *Mathematics of operations research*, vol. 27, no. 4, pp. 819–840, 2002.

[83] S. Hung and S. N. Givigi, "A q-learning approach to flocking with uavs in a stochastic environment," *IEEE Transactions on Cybernetics*, vol. 47, no. 1, pp. 186–197, 2016.

[84] M. Tan, "Multi-agent reinforcement learning: Independent versus cooperative agents," in *International Conference on Machine Learning (ICML)*, 1993.

[85] C. Claus and C. Boutilier, "The dynamics of reinforcement learning in cooperative multiagent systems," in *AAAI Conference on Artificial Intelligence (AAAI)*, 1998.

[86] E. Van der Pol and F. A. Oliehoek, "Coordinated deep reinforcement learners for traffic light control," in *NIPS Workshop on learning, inference and control of multi-agent systems*, 2016.

[87] K. D. Julian and M. J. Kochenderfer, "Autonomous distributed wildfire surveillance using deep reinforcement learning," in *AIAA Guidance, Navigation, and Control Conference (GNC)*, 2018.

[88] F. A. Oliehoek, S. Whiteson, and M. T. J. Spaan, "Approximate solutions for factored dec-pomdps with many agents," in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2013.

[89] G. Hardin, "The tragedy of the commons," *Science*, vol. 162, no. 3859, pp. 1243–1248, 1968.

[90] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. F. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, "Value-decomposition networks for cooperative multi-agent learning based on team reward," in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2018.

[91] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. N. Foerster, and S. Whiteson, "QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2018.

[92] S. Gu, E. Holly, T. P. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[93] W. D. Smart and L. P. Kaelbling, "Effective reinforcement learning for mobile robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[94] M. Gottwald, D. Meyer, H. Shen, and K. Diepold, "Learning to walk with prior knowledge," in *IEEE International Conference on Advanced Intelligent Mechatronics*, 2017.

[95] M. Cutler, T. J. Walsh, and J. P. How, "Real-world reinforcement learning via multifidelity simulators," *IEEE Transactions on Robotics*, vol. 31, no. 3, pp. 655–671, 2015.

[96] M. Eldred and D. Dunlavy, "Formulations for surrogate-based optimization with data fit, multifidelity, and reduced-order models," in *AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2006.

[97] D. Rajnarayan, A. Haas, and I. Kroo, "A multifidelity gradient-free optimization method and application to aerodynamic design," in *AIAA/ISSMO multidisciplinary Analysis and Optimization Conference*, 2008.

[98] S. Li, M. Egorov, and M. J. Kochenderfer, "Optimizing collision avoidance in dense airspace using deep reinforcement learning," in *Air Traffic Management Research and Development Seminar*, 2019.

[99] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *International Conference on Computer-Aided Verification*, 2017.

[100] N. Fulton and A. Platzer, "Safe reinforcement learning via formal methods: Toward safe control through proof and learning," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[101] J. Garcıa and F. Fernández, ''A comprehensive survey on safe reinforcement learning,'' *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[102] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, ''Safe reinforcement learning via shielding,'' in *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[103] M. Mukadam, A. Cosgun, A. Nakhaei, and K. Fujimura, ''Tactical decision making for lane changing with deep reinforcement learning,'' in *NIPS Workshop on Machine Learning for Intelligent Transportation Systems*, 2017.

[104] A. Pnueli, ''The temporal logic of programs,'' in *Symposium on Foundations of Computer Science*, 1977.

[105] C. Finucane, G. Jing, and H. Kress-Gazit, ''LTLMoP: Experimenting with language, temporal logic and robot control,'' in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[106] C. Baier and J. Katoen, *Principles of model checking*. MIT Press, 2008.

[107] M Lahijanian, S. Andersson, and C Belta, ''Control of markov decision processes from PCTL specifications,'' in *American Control Conference (ACC)*, 2011.

[108] M. Lahijanian, J. Wasniewski, S. B. Andersson, and C. Belta, ''Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees,'' in *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[109] M. Hasanbeig, A. Abate, and D. Kroening, ''Logically-constrained neural fitted q-iteration,'' in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2019.

[110] M. Bouton, J. Tumova, and M. J. Kochenderfer, ''Point-based methods for model checking in partially observable markov decision processes,'' in *AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[111] M. Z. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *International Conference on Computer-Aided Verification*, 2011.

[112] C. Dehnert, S. Junges, J. Katoen, and M. Volk, "A storm is coming: A modern probabilistic model checker," in *International Conference on Computer-Aided Verification*, 2017.

[113] K. Chatterjee, M. Chmelik, and M. Tracol, "What is decidable about partially observable markov decision processes with omega-regular objectives," in *Computer Science Logic (CSL)*, 2013.

[114] K. Chatterjee, M. Chmelik, R. Gupta, and A. Kanodia, "Qualitative analysis of POMDPs with temporal logic specifications for robotics applications," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[115] M. Svorenová, M. Chmelik, K. Leahy, H. F. Eniser, K. Chatterjee, I. Cerná, and C. Belta, "Temporal logic motion planning using POMDPs with parity objectives: Case study paper," in *International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2015.

[116] S. Junges, N. Jansen, R. Wimmer, T. Quatmann, L. Winterer, J. Katoen, and B. Becker, "Finite-state controllers of POMDPs using parameter synthesis," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018.

[117] R. Sharan and J. W. Burdick, "Finite state control of POMDPs with LTL specifications," in *American Control Conference (ACC)*, 2014.

[118] G. Norman, D. Parker, and X. Zou, "Verification and control of partially observable probabilistic systems," in *Real-Time Systems*, vol. 53, 2017.

[119] S. Haesaert, P. Nilsson, C. I. Vasile, R. Thakker, A. Agha-mohammadi, A. D. Ames, and R. M. Murray, "Temporal logic control of POMDPs via label-based stochastic simulation relations," in *IFAC Conference on Analysis and Design of Hybrid Systems, ADHS*, 2018.

[120] C. I. Vasile, K. Leahy, E. Cristofalo, A. Jones, M. Schwager, and C. Belta, "Control in belief space with temporal logic specifications," in *IEEE Conference on Decision and Control (CDC)*, 2016.

[121] Y. Wang, S. Chaudhuri, and L. E. Kavraki, "Bounded policy synthesis for POMDPs with safe-reachability objectives," in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2018.

[122] M. Ahmadi, M. Cubuktepe, N. Jansen, and U. Topcu, "Verification of uncertain POMDPs using barrier certificates," in *Allerton Conference on Communication, Control, and Computing*, 2018.

[123] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017.

[124] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu, "Spot 2.0 – a framework for LTL and ω-automata manipulation," in *Automated Technology for Verification and Analysis (ATVA)*, ser. Lecture Notes in Computer Science, vol. 9938, 2016.

[125] T. Smith and R. G. Simmons, "Heuristic search value iteration for POMDPs," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2004.

[126] W. S. Lovejoy, "Computationally feasible bounds for partially observed markov decision processes," *Operations Research*, vol. 39, no. 1, pp. 162–175, 1991.

[127] D. Hsu, W. S. Lee, and N. Rong, "What makes some POMDP problems easy to approximate?" In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

[128] A. Kolobov, Mausam, and D. S. Weld, "A theory of goal-oriented mdps with dead ends," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2012.

[129] S. C. W. Ong, S. W. Png, D. Hsu, and W. S. Lee, "POMDPs for robotic tasks with mixed observability," in *Robotics: Science and Systems*, 2009.

[130] D. Sadigh, E. S. Kim, S. Coogan, S. S. Sastry, and S. A. Seshia, ''A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications,'' in *IEEE Conference on Decision and Control* (*CDC*), 2014.