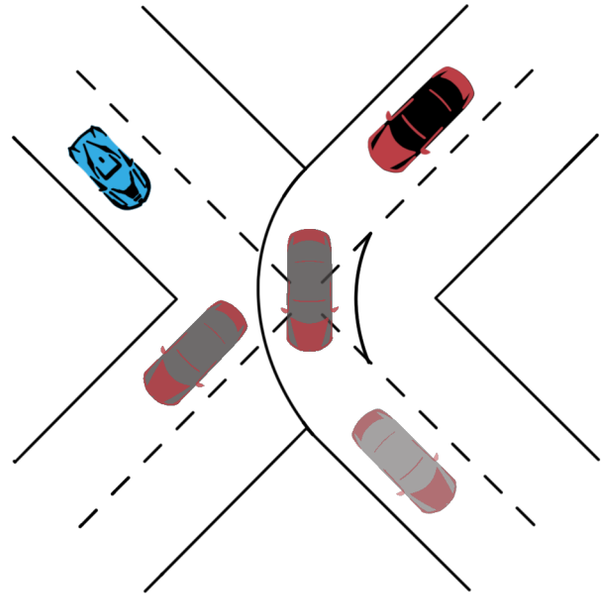
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

PHD THESIS

Our world of transportation is rapidly evolving, with autonomous driving (AD) technology set to revolutionize urban mobility by redefining how we navigate cities, enhancing safety, and paving the way for a more efficient, inclusive, and sustainable future. However, navigating intersections safely remains a critical challenge on the path to this future. This thesis explores using Deep Q-learning to train decision-making agents for intersection navigation, focusing on understanding and managing uncertainty in other drivers' intentions. Deep Q-learning is a Reinforcement Learning (RL) algorithm which learns through trial and error. In this thesis, agents were developed and tested to master intersection strategies within simulated environments. The goal is to equip these agents with the ability to make split-second decisions based on real-time data about vehicle positions and velocities to estimate what other drivers might do next.

Key highlights of the research include the successful integration of RL algorithms with advanced control methods like Model Predictive Control (MPC), addressing uncertainties through sophisticated AI techniques, and leveraging previous models to train new ones. This combined approach significantly enhanced the agents' ability to anticipate and respond to unpredictable driver behaviors, contributing to the advancement of safety and reliability in autonomous driving systems within real-world environments. The findings of this thesis hold promise for future innovations in AI-driven transportation systems, aiming towards safer roads and more efficient traffic management solutions in our increasingly interconnected world.

TOMMY FUJITA TRAM • Learning When To Drive in Uncertrain Scenarios • 2024



Learning When To Drive in Uncertrain Scenarios

A deep Q-learning approach

TOMMY FUJITA TRAM

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY

THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Learning When To Drive in Uncertrain Scenarios

A deep Q-learning approach

Tommy Fujita Tram

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2024

Learning When To Drive in Uncertrain Scenarios
A deep Q-learning approach

Tommy Fujita Tram
ISBN 978-91-7905-623-0

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 5089
ISSN 0346-718X

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
Phone: +46 (0)31 772 1000
Email: tommy.tram@chalmers.se; tommy.tram@gmail.com

Cover:
An illustration of two vehicles approching an intersection. The blue vehicle has to make a decision to yield or drive while the intention of the red vehcile is uncertain.

Printed by Chalmers Reproservice
Gothenburg, Sweden, June 2024

To my loving family

Learning When To Drive in Uncertrain Scenarios

A deep Q-learning approach

Tommy Fujita Tram
Department of Electrical Engineering
Chalmers University of Technology

Abstract

The main focus of this thesis is tactical decision-making for autonomous driving (AD) through intersections with other road users. Human drivers can navigate diverse environments and situations, even those they have never encountered before. Autonomous vehicles are expected to have similar capabilities. This thesis specifically addresses the challenge of navigating intersections where the intentions of other drivers are unknown, as these intentions can be influenced by factors such as driver mood, attention, right-of-way, and traffic signals.

To tackle the complexity of manually specifying reactions for every possible situation, this thesis adopts a learning-based strategy using reinforcement learning (RL). The problem is formulated as a partially observable Markov decision process (POMDP) to account for the uncertainty of unknown driver intentions. A general decision-making agent, based on the deep Q-learning algorithm, is proposed. The contributions of this thesis include the development and application of this method to various simulated intersection scenarios, demonstrating its adaptability and effectiveness in different environments with minimal modifications. By accounting for the inherent uncertainty in driver behavior, this approach enhances the robustness and reliability of the autonomous driving system.

Keywords: Autonomous driving, reinforcement learning, decision making, uncertain environments, Partially observable Markov decision process, deep Q-learning, transfer learning, model predictive control, neural networks

List of Publications

This thesis is based on the following publications:

[A] Tommy Tram, Anton Jansson, Robin Grönberg, Mohammad Ali, and Jonas Sjöberg, "Learning Negotiating Behavior Between Cars in Intersections using Deep Q-Learning". *Published in 2018 21st International Conference on Intelligent Transportation Systems (ITSC).*

[B] Tommy Tram, Ivo Batković, Mohammad Ali, and Jonas Sjöberg, "Learning When to Drive in Intersections by Combining Reinforcement Learning and Model Predictive Control". *Published in 2019 IEEE Intelligent Transportation Systems Conference (ITSC).*

[C] Carl-Johan Hoel, Tommy Tram, and Jonas Sjöberg, "Reinforcement Learning with Uncertainty Estimation for
Tactical Decision-Making in Intersections". *Published in 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC).*

[D] Tommy Tram, Maxime Bouton, Jonas Fredriksson, Jonas Sjöberg, and Mykel Kochenderfer, "Belief State Reinforcement Learning for Autonomous Vehicles in Intersections". *Submitted to IEEE Transactions on Intelligent Transportation Systems.*

[E] Hannes Eriksson, Tommy Tram, Debabrota Basu, Mina Alibeigi, and Christos Dimitrakakis, "Reinforcement Learning in the Wild with Maximum Likelihood-based Model Transfer". *Published in 2024 International Conference on Adaptive Agents and Multi-Agent Systems (AAMAS).*

Acknowledgments

This thesis is the result of many years of hard work, lots of travel and a great learning experience. Even in times when things did not work as expected or unexpected problems emerged, I never felt that I was alone. That is why I would like to acknowledge the people who have made this work possible and memorable.

First, I want to thank my supervisor and examiner Professor Jonas Sjöberg. Thank you for believing in me and for the opportunity of pursuing a doctoral degree as your student. It has been an honor and a pleasure. I would also like to thank Jonas Fredriksson that assisted in the final year and made it possible for me to cross the finish line.

A special thanks to my industrial supervisor Dr. Mohammad Ali, who chose and encouraged me to pursue a PhD in this field. I still remember the pitch that convinced me to do it. You said: "Pilot assist is mostly done now, we need someone to look towards the future of decision-making and control". We had a lot of fun in the beginning of this journey and I will always be grateful for the guidance and support you provided, and most of all for believing in me.

The work presented in this thesis would not have been possible without the financial support from VCC, Zenuity, Zenseact, and the Wallenberg AI, Autonomous Systems and Software Program (WASP). I am also grateful for the community and environment created by both Zenseact and WASP. The Advanced Graduate Program led by Dr. Mats Nordlund and Dr. Carl Lindberg created an environment within the company which cultivated a safe space to share ideas, talk amongst peers and fun to go to work in the morning.

I would especially like to thank Dr. Carl Lindberg for the coaching during these final years of my PhD. At the start of the pandemic, you paid attention to our casual conversations, identified problems and proposed solutions I did not think was possible, but more importantly you cared. Thanks to you, I was able to finish my final year of my PhD together with my wife on the other side of the world and as a direct result of that, I now have a beautiful son. For this, I am eternally grateful.

I especially want to thank my great friends and co-authors with whom I embarked on this PhD journey with. Carl-Johan Hoel, my research twin, thank you for all the support over the years, the enlightening conversations and most of all for being a friend. Ivo Batkovic, thank you for all the good times we shared. I have always admired your determination and focus. Working

with you has been a pleasure and a privilege. I am happy to call you my friend. I would also like to thank Anton Jansson and Robin Grönberg for their well executed master thesis from where I found the direction of this thesis. Additionally, I am grateful to Hannes Eriksson and Debabrota Basu for our collaboration on the transfer learning experiments.

Special thanks go to Christian Rodriguez for being a great friend and supporting me whenever times were tough.

I would like to thank WASP for all the interesting study trips, courses, and events that not only widened my skills, but also introduced me to a vast network of colleagues and friends across the world. In addition, the WASP exchange program gave me the opportunity to spend six months as a visiting research student at the Stanford Intelligent Systems Laboratory (SISL). To that end, I am deeply grateful to Professor Mykel Kochenderfer who thought me that life is a POMDP and gave me the opportunity to spend time and work with his research group. Additionally, special thanks to Maxime Bouton for the fun we had during my time at SISL. I learned a lot, both on and off campus.

Tommy Fujita Tram,
Göteborg, June, 2024.

Acronyms

AD:	Autonomous driving
ADAS:	Advanced driving assistance systems
AV:	Autonomous vehicles
ASIL:	Automotive Safety Integrity Level
CNN:	Convolutional neural network
DQN:	Deep Q-network
DRQN:	Deep Recurrent Q-network
EQN:	Ensemble quantile networks
IDM:	Intelligent driver model
LSTM:	Long short-term memory
LQR:	Linear Quadratic Regulator
MDP:	Markov decision process
MPC:	Model predictive control
MCTS:	Monte Carlo tree search
NN:	Neural network
POMDP:	Partially observable Markov decision process
SGD:	Stochastic gradient descent
RPF:	Randomized prior functions
RL:	Reinforcement learning

Contents

Part I

Overview

CHAPTER 1

Introduction

The way we transport ourselves is currently evolving, and Autonomous Driving (AD) technology is expected to have a big impact on this transformation [1], [2]. With Autonomous Vehicles (AV) the efficiency of traffic can be improved by scheduling commercial transports outside of rush hours [3]. The number of parking spots in cities can be reduced if the vehicles can autonomously drive itself to a less crowded area when not in use and drive back when needed. Congestion and traffic jams could also be reduced if a large amount of vehicles in traffic are autonomous and optimize around the same goal e.g., traffic flow or fuel efficiency.

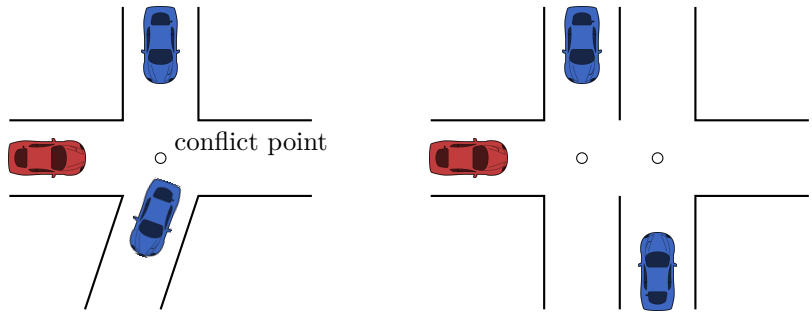
The rapid success of Machine Learning (ML) during the last decades has lead to major progress towards deploying AVs in the real world. One clear beneficiary of these new ML techniques are the perception systems [4]. Better perception enables more accurate representation of the environment. However, navigating complex scenarios such as urban intersections and roundabouts with dense traffic remain challenging for AVs because it requires a higher level of interaction between road users, as summarized in a review paper [5]. When human drivers approaches an intersection, they normally assess who has the right of way, before deciding whether to proceed or yield to other road

users. However, human drivers do not always follow the rules. According to the Insurance Institute for Highway Safety [6], in 2019, an estimated 115,741 people were injured by drivers running a red light, whereas 928 of them were killed. While these accidents were mainly caused by driver inattention or reckless driving, it motivates the development of decision-making algorithms for AVs which not only follow the traffic rules but can also take into account other drivers future actions and inattention, which is the main focus in this thesis.

1.1 Autonomous Driving Levels

When talking about autonomous driving, it is first important to specify which level of autonomy that is being discussed. The Society of Automotive Engineers has classified these different levels of autonomy ranging from zero to five [7], also referred to as L0-L5. The first level L0 is a vehicle with no autonomy, whereas a fully AV that can operate in any environment and without any human supervision is defined as L5. Popular Advanced Driver Assistance Systems (ADAS) functions today, like lane centering or adaptive cruise control are classified as L1, while the Volvo Pilot Assist and Tesla Autopilot that provide both steering and acceleration/braking are classified as level 2. The main criterion for L2 systems is that the driver is in control and only supported by the system. This requires the driver to always supervise the vehicle and take over when needed to ensure safety. For L3 and higher, the responsibility of driving shifts to the system. At L3, the driver still has to take control over the vehicle, but only when the systems request it. At L4 and L5, the autonomous driving features no longer require the driver to take over. The main difference between L4 and L5 is the capability of driving anywhere under all conditions. Examples of L4 are robot taxis developed by Waymo, Zoox, Cruise and Toyota which only operate in a specified area or city.

The methods presented in this thesis are aimed at an autonomy level L3-L5. At the highest level the system is expected to handle all aspects of driving within a specific task such as crossing an intersection at any location.



(a) Single intersection **(b)** Double intersection

Figure 1.1: Examples of different intersections

1.2 Intersections, intentions and scenarios

This section aims to clarify the use of the terms' *intersection*, *intention*, and *scenario* within the context of this thesis. An intersection refers to the geometric layout of roads intersecting each other, encompassing elements such as the number of junctions, conflict points, turns, and angles of incidence, as illustrated in Figure 1.1. Intersections can be categorized as signalized or unsignalized. A signalized intersection is equipped with infrastructure to designate the right-of-way, such as regulatory signs (e.g., STOP or YIELD) or traffic signals. In contrast, an unsignalized intersection lacks such features, relying on local driving rules, such as giving the right-of-way to vehicles approaching from the right. However, as emphasized in the introduction, human drivers do not always follow these right-of-way rules, which can result in accidents. Therefore, this thesis defines intentions as the anticipated actions of other vehicles in the future, such as stopping or proceeding through the intersection.

Assume there are two main intentions: "take way" and "give way" (yield). In Figure 1.2, three different agents with a starting velocity 12m/s are approaching an intersection. At time 0s, agents with a give way intention will start to brake. With these two main intentions, other intentions can be derived; for example, a cautious intention can be modeled as a give way agent that changes its intention to take way at 2 seconds. The reason for the change can be that a cautious agent lowers the speed initially to have the possibility to give way,

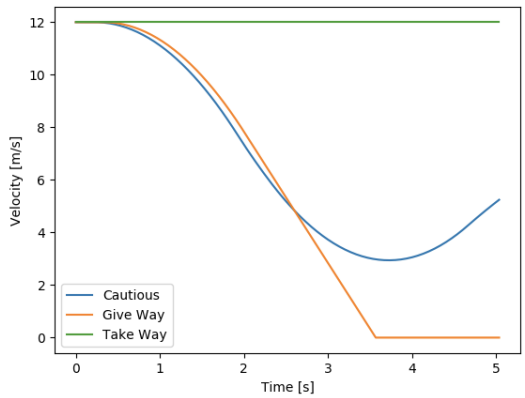


Figure 1.2: Velocity profiles illustrating three different intentions for an agent approaching an intersection from the same initial position and velocity. The green curve represents an agent with a 'take way' intention, while the orange curve shows an agent intending to 'give way,' stopping before the intersection. The blue curve depicts a cautious intention, starting with a slowdown but without coming to a complete stop.

and later speed up only if there is no potential conflict. Distinguishing between the 'give way' and 'take way' intentions poses a challenge because even after the initial braking at 0 seconds, it is not certain that the agent will stop until 3 seconds after the initial braking. This uncertainty may be considered too conservative by the person riding the AV.

If driver intentions can be accurately predicted, intersections could be managed without the need for traffic lights, as right-of-way decisions can be made based on intentions rather than relying solely on infrastructure. A Reinforcement Learning (RL) approach could enhance driving in scenarios where understanding intentions is crucial, such as at intersections with both stop signs and traffic lights, or when a vehicle violates traffic rules by running a red light. By training an RL agent in a simulation environment with varying driver intentions, it can learn to infer these intentions and make optimal decisions through trial and error.

1.3 Research questions

This thesis focuses on investigating and evaluating RL agents for navigating complex intersection scenarios, with a particular emphasis on managing uncertainty. The intersection navigation problem is formulated as a Partially Observable Markov Decision Process (POMDP), where observable states include positions and velocities of vehicles, while unobservable states pertain to the intentions of surrounding drivers. With this in mind, this thesis aims to answer the following research questions:

Q1. How can RL techniques be used to develop a decision-making agent that effectively navigates intersections without explicitly estimating the intention state of other vehicles?

A deep Q-learning approach is used to solve the POMDP, with short-term goal as actions. These short-term goals are translated into reference points and constraints for a controller. Paper A uses a sliding mode controller, while Paper B uses a Model Predictive Control (MPC) to generate the vehicles' acceleration. To address the challenge of unobservable intentions, the hidden state in the Long Short-Term Memory (LSTM) layer of the RL agent is designed to incorporate estimations of these intentions.

Q2. How can an RL agent utilize the uncertainty in its predictions and actions to enhance decision-making in complex environments?

Two main sources of uncertainty are tackled in this thesis: Q-value uncertainty and intention state uncertainty. For Q-value uncertainty, Paper C estimates the uncertainty in the Q-values predicted by using an ensemble of neural networks on different subsets of the available data. This ensemble provides a distribution over the estimated Q-values and the uncertainty in choosing different actions can be defined as the coefficient of variation of the Q-values. In Paper D, the hidden intention states of other vehicles are represented using a belief state generated via a particle filter. This belief state provides a probability distribution over possible true states, which informs the agent's decisions.

Two methods are proposed to handle intention estimation: QMDP-IE and QID. QMDP-IE, derived from the QMDP algorithm, uses a single estimated intention state, reducing computational complexity while blending elements of POMDPs and MDPs. In contrast, QID represents the intention state using a

belief state distribution, allowing for more adaptive and robust decision-making in uncertain environments.

Q3. How can an RL agent handle situations it has not been trained on?

A confidence criterion is applied to the uncertainty of the estimated Q-values in Paper C, making actions with high uncertainty invalid and if no action satisfies the confidence criterion a backup action e.g., emergency braking, can be applied avoiding potential collisions. If the uncertainty measure from Paper C and Paper D can be used to identify that the agent is in the wrong environment, the transfer RL approach from Paper E can be employed to determine which environment out of a set of environments the agent is currently in, or to select the policy that best fits the current situation

1.4 Scope and limitations

The following aspects of creating a tactical decision-making agent for autonomous driving in uncertain environments are not considered in this thesis.

1. Guaranteeing safety in AD systems is an important open question that is out of scope for this thesis.
2. The work in this thesis is tested in simulation environments and not real world.
3. This work considers the control of one vehicle and not multiple agents.

1.5 Contributions

The main contributions of this thesis are:

1. A deep Q-learning approach for creating a decision-making agent navigating intersection that considers the intentions of other drivers.
2. A neural network architecture that is invariant to permutations of the order of which surrounding traffic participants are observed, which speeds up training and improves the quality of the trained agent.
3. A belief state representation of driver intentions using a particle filter.

4. A belief state Deep Q-network (DQN) method that can adjust the aggressiveness of the policy using one threshold parameter.

5. Extension of RL methods that provide an uncertainty estimate of the proposed decisions and use it to create a confidence criterion that can identify situations with high uncertainty.

6. A transfer learning method that is able to identify which Markov Decision Process (MDP) the agent is in from a set of MDPs.

1.6 Thesis outline

The outline of the thesis is as follows: in Chapter 2 other research in the same field is presented. Chapter 3 introduce the mathematical framework MDP and POMDP with a brief theory of RL, deep Q-learning and the Intelligent Driver Model (IDM). Chapter 4 is where the problem is formulated by defining the components of the POMDP. Results from using deep Q-learning to solve the POMDP is presented and later combined with an MPC to improve the actions. Later in Chapter 5 two approaches to handle the uncertainty is presented. First the uncertainty in the decisions from the RL algorithm and then an empirical study of how well a DQN can handle uncertainty of others driving intentions. Chapter 6 present an approach to generalize over different MDPs more specifically policies learned from different transfer functions. The synergies and differences of the different methods are highlighted in Chapter 7. Finally, Chapter 8 provides some concluding remarks and future research directions.

CHAPTER 2

Related work

In recent years, decision-making for AVs in structured scenarios like intersections has attracted a lot of attention in the literature. This chapter provides a broad introduction to the primary research directions and outlines how the contributions in this thesis relate to existing work. However, it does not aim to provide a comprehensive survey of every approach.

2.1 Rule-based methods

A skilled engineer can sometimes solve the decision-making problem for structured traffic scenarios using rule-based methods. One example of a rule-based method was implemented using hierarchical state machines to switch between predefined behaviors depending on what scenarios was encountered [8], [9]. These methods were successful for a limited and controlled environment such as the Urban Challenge event, but it is difficult for an engineer to anticipate every situation that may occur in the real world and design a suitable strategy that can solve all of them, in particular when drivers are not following the law [10]. The limitations of rule-based approaches motivate the choice of more adaptive and flexible methods, such as a planning-based or learning-based

method.

2.2 Planning-based methods

Planning-based methods treats the decision-making task as a motion planning problem. Commonly, a prediction model is used to predict the motion of the other agents, and then the behavior of the ego vehicle that is being controlled is planned accordingly. Liebner *et al.* [11] used to the IDM infer driver intent in urban intersections and Hoermann *et al.* [12] used a particle filter to estimate the parameters of the IDM, both works showed promising results when evaluated on real-world data. Consequently, the IDM is used in this thesis to model the driving behavior of other vehicles.

One planning-based method is using Monte Carlo Tree Search (MCTS) [13], but since the predictions are independent of the ego vehicle plan results in a reactive behavior [14], [15]. Therefore, the interaction between the ego vehicle and other agents is not explicitly considered, but may happen implicitly by frequent replanning. MCTS also requires extensive online computation and can be hard to scale in complex traffic situations with an increasing number of traffic participants.

Another approach to solve the motion planning problem is to use optimal control, which was applied to highway driving scenarios by Werling *et al.* [16]. Since human behavior is complex and varies between individuals, a study by Damerow *et al.* [17] use a probabilistic prediction as input to the motion planning, which aims to minimize the risk of collision during an intersection scenario. While Batkovic *et al.* [18] used a robust scenario MPC approach to handle uncertain multi-modal road users. Other approaches to motion planning for autonomous driving are provided in the surveys by González *et al.* [19] and Paden *et al.* [20]. However, these planning-based methods rely on the accuracy of the prediction models and require a lot of on-board computing power which may be limited in an AV.

2.3 Learning based methods

Learning-based approaches offer the ability to learn from experience, adapt to new situations, and make decisions based on a wide range of scenarios, rather than relying on predefined rules or models. RL methods can help relieve

the burden of designing hand-crafted solutions for all possible scenarios [21], [22]. The work by Mnih *et al.* [23] showed a DQN that achieved impressive results in training agents to play Atari games from raw pixel inputs using experience replay, highlighting DQNs ability to learn complex behaviors from high-dimensional sensory data, a key requirement for autonomous vehicles navigating intersections. In order to handle the uncertainty of predicting other traffic participants' behaviors or intentions, the literature formulates the problem of driving under uncertainty as a POMDP [24]. Deep Recurrent Q-Network (DRQN) approaches, such as the ones from Hausknecht *et al.* [25] and Zhu *et al.* [26], showed some promise solving POMDP with non-observable states by leveraging past observations or actions.

Another approach by Bouton *et al.* [27] used belief states to capture uncertainties in the environment. The belief state can be used to model the probability distribution over the uncertain world states, e.g., the intention of other road users. Wang *et al.* [28] decoupled the belief state modeling (via unsupervised learning) from policy optimization (via RL) and Littman *et al.* [29] claimed that having full observability at learning time, combined with knowing what will not be observable at deployment time, enables an RL agent to learn a policy that is more robust to its unobservable states.

Advantage of DQN methods, compared to planning based methods, is that DQN learns through interaction with the environment. Experience replay allows DQN to learn efficiently from past experiences, improving its performance with less real-world data compared to methods without it. This capability enables autonomous vehicles to adapt to unseen situations and make informed decisions based on accumulated experiences at intersections. While DQN doesn't directly address driver intentions, it can learn to infer them indirectly from traffic patterns and historical data. This allows for adaptive decision-making based on the perceived likelihood of driver actions at intersections. Therefore, DQN is a suitable choice for this thesis due to its adaptability and efficiency in learning from complex, dynamic environments.

CHAPTER 3

Technical background

This chapter briefly introduces the MDP framework and its extension POMDP and RL. A more comprehensive overview of POMDPs and RL is given in the books by Kochenderfer [24] and Sutton and Barto [21], upon which this chapter is based. The purpose of the chapter is to summarize the most important concepts and introduce the notation that are used in the subsequent chapters.

3.1 Markov decision process

A MDP is a mathematical framework for modeling discrete time sequential decision-making problems. It involves an agent making decisions in an environment evolving over time according to a stochastic process. The state of the environment contains all the information necessary about the agent and environment at a given time to be able to transition to any given state. This property is referred to as the Markov property.

The MDP is formally defined as the tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, described by the following list [24]:

- The state space $\mathcal{S}$ represents the set of all possible states of the environ-

ment. This set could consist of both discrete and continuous states.

- The action space $\mathcal{A}$ represents the set of all possible actions the agent can take. The action space can consist of both discrete and continuous actions. Since this thesis focuses on high-level decision-making, only discrete actions are considered.

- The state transition model $T(s' \mid s, a)$ describes the probability $\Pr(s' \mid s, a)$ that the system transitions to the next state $s' \in \mathcal{S}$ from state $s \in \mathcal{S}$ when action $a \in \mathcal{A}$ is taken.

- The reward function $R(s, a)$ returns a scalar reward r for each action a an agent takes in a given state s. The design of the reward function should reflect the overall objective that the agent should maximize.

- The discount factor $\gamma \in [0, 1)$ is a scalar that discounts the value of future rewards. The discount factor γ will affect the results of the optimization problem. A discount factor set close to 0 will make immediate rewards more important while a γ closer to 1 would give some weight to expected future reward as well.

A policy π is defined as the mapping from state s to action a and the goal of the agent is to take a sequence of actions that maximizes the accumulated reward r. The value of being in a state while following a policy is described by the value function

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(s_t, a_t) | s_0 = s, \pi\right]. \tag{3.1}$$

The optimal value function V^* is unique and follows the Bellman equation:

$$V^*(s) = \max_a \left[R(s, a) + \gamma \sum_{s'} T(s, a, s') V^*(s')\right]. \tag{3.2}$$

From the bellman equation one can deduce a state-action value function $Q(s, a)$ that satisfies $V^*(s) = \max_a Q(s, a)$. Given this Q function, a policy can be derived as $\pi(s) = \operatorname{argmax}_a Q(s, a)$.

3.1.1 Partially observable Markov decision process

Sometimes the agent does not have direct access to the entire state of the environment. In these cases, it is more common to model the problem as a POMDP, which is an extension to the MDP. A POMDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, O, R, \gamma)$, where the state space, action space, transition model, reward and discount factor is the same as the MDP, but a POMDP has two additional elements:

- The observation space $\mathcal{O}$, which represents all possible observations that the agent can receive. This can be both discrete and continuous.
- The observation model $O(o|s', a)$, which describes the probability of observing $o \in \mathcal{O}$ in a given state s' after taking an action a: $O(o, s', a) = \Pr(o|s', a)$.

In a POMDP, the agent takes an action a from a given state s and the environment transitions to the next state s' according to the transition model T. The agent then receives an observation o related to s' and a according to the observation model O. After the agent takes an action a from a given state s and the environment transitions to the next state s' according to the transition model T, the agent receives an observation o. This observation o is drawn according to the observation model $O(o|s', a)$, which specifies the probability of observing the given new state s' and taken action a.

Since the state is not observable, policies in a POMDP are no longer described by mapping from states to actions. Instead, the agent must reason about the history of observations and actions. This history can sometimes be summarized in a statistic referred to as a belief state (or belief). A belief state b is a probability distribution such that

$$b(s) = \Pr(s \mid o_{1:t}) \tag{3.3}$$

is the probability of being in state s at time t, given observations $o_{1:t} := \{o_1, o_2, ..., o_t\}$ up to time t. At each time step t, the agent updates its belief using a Bayesian filtering approach given the previous belief and the current observation as follows:

$$b'(s') \propto O(o \mid s', a) \int_{s \in S} T(s' \mid s, a) b(s), \tag{3.4}$$

where b' is the updated belief state after taking action a and receiving observation o. By summarizing all relevant information into the current belief state, a POMDP also satisfy the Markov property [21, Ch. 17], ensuring that future states depend only on the current belief state and not on the entire history of actions and observations. This approximation is referred to as a k-Markov approximation, where k defines the length of the included history. With a sufficiently long history, the Markov property is assumed to approximately hold, even in a partially observable environment.

Policies are now described as mappings from beliefs to actions. The optimal belief state value function $V^*(b')$ that satisfies the Bellman equation can be formulated as:

$$V^*(b) = \max_a \left[R(b,a) + \gamma \sum_{o \in \mathcal{O}} \Pr(o \mid b, a) V^*(b') \right] \tag{3.5}$$

where b' is computed using (3.4), and $R(b,a) = \int_{s \in \mathcal{S}} b(s) R(s,a)$ is the expected reward in a belief state b. Similarly, as in the MDP one can deduce a belief-action function $Q(b,a)$ that satisfy $V^*(b) = \max_a Q(b,a)$ and the policy a policy can be derived as $\pi(b) = \operatorname{argmax}_a Q(b,a)$.

The observable states in this work are information that sensors on the ego vehicle can provide e.g., distance to intersection, position and speeds of other vehicles. While the unobservable states are the intentions of other drivers that are approaching the same intersection as ego. Chapter 4 will later formulate the POMDP studied in this work.

3.2 Reinforcement learning

In some problems, the state transition probabilities or the reward function are unknown. These problems can be addressed using reinforcement learning, where the agent learns how to behave by interacting with the environment [24, Ch. 5]. The data available to an RL agent depends on its current policy, necessitating a balance between exploring the environment and exploiting existing knowledge. Moreover, the reward an agent receives might hinge on a crucial decision made earlier, making it essential to attribute rewards to the correct decisions.

RL algorithms are categorized into two approaches: model-based and model-free [24, Ch. 5]. In model-based methods, the agent first estimates a represen-

tation of the state transition function T and then uses a planning algorithm to find a policy. Conversely, model-free RL algorithms, as the name suggests, do not explicitly construct a model of the environment to determine actions.

Model-free approaches can be further divided into value-based and policy-based techniques. Value-based algorithms, such as Q-learning [30], aim to learn the value of each state, thereby implicitly defining a policy. Policy-based techniques [31] directly search for the optimal policy within the policy space, either through policy gradient methods [32] or gradient-free methods like evolutionary optimization. Hybrid techniques, such as actor-critic methods [33], combine both policy and value-based approaches.

RL algorithms typically assume that the environment is modeled as a MDP, where the agent knows the state of the environment. However, in many practical situations, only partial information about the state of the environment is available, which is modeled within the POMDP framework. In such cases, it is common to approximate the state as a belief b [34].

3.2.1 Deep Q-learning

Q-learning [30] is a model-free and value-based RL algorithm, where the objective of the agent is to learn the optimal state-action value function $Q^*(s, a)$. This function is defined as the discounted expected return when taking action a from state s and then following the optimal policy π^*, i.e.,

$$Q^*(s, a) = \max_{\pi} \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(s_k, a_k) | s_0 = s, a_0 = a, \pi\right]. \tag{3.6}$$

The optimal state-action value function follows the Bellman equation

$$Q^*(s, a) = \mathbb{E}_{s' \sim T(s'|s,a)}\left[R(s, a) + \gamma \max_{a'} Q^*(s', a')\right], \tag{3.7}$$

which recursively defines the Q-values of the state-action pairs (s, a). The equation can intuitively be understood by the fact that if Q^* is known, the optimal policy is to select the action a' that maximizes $Q^*(s', a')$.

In the DQN algorithm, a Neural Network (NN) with weights θ is used as a function approximator of the optimal state-action value function, $Q(s, a; \theta) \approx Q^*(s, a)$ [35]. The weights of the network are adjusted to minimize the temporal difference (TD) error in the Bellman equation, typically with some kind of Stochastic Gradient Descent (SGD) algorithm. Mini-batches with size M of

experiences, $e = (s, a, r, s')$, are drawn from an experience replay memory, and the loss is calculated as

$$L(\theta) = \mathbb{E}_{\mathrm{M}}\Big[(r + \gamma \max_{a'} Q(s', a'; \theta^{-}) - Q(s, a; \theta))^2\Big]. \tag{3.8}$$

Here, θ^{-} represents the NN parameters of a target network, which is kept fixed for a number of training steps, in order to stabilize the training process. Several of improvements to this standard form of DQN have been proposed and are compared by Hessel et al. [36]. See Paper A for the details of the DQN implementation in this thesis.

CHAPTER 4

Modeling Intersection Driving Scenarios

Navigating an intersection is a sequential decision-making problem that can be mathematically modeled using an MDP, as introduced in Section 3.1. This thesis focuses on driving through intersections in the presence of other drivers, emphasizing not only to follow traffic rules but also the ability to adapt to the intentions of other drivers. Given that current sensors cannot directly observe other drivers' intentions, a POMDP is a more suitable framework for formulating this problem.

This chapter explores the modeling of intersection driving scenarios for autonomous vehicles.

4.1 POMDP formulation

Effectively modeling the intersection problem is key for developing an optimal decision-making policy. This section outlines each component of the POMDP framework as applied to the intersection problem in this thesis. While specific details of the POMDP may vary between Paper A-D, the general description remains consistent.

4.1.1 State space

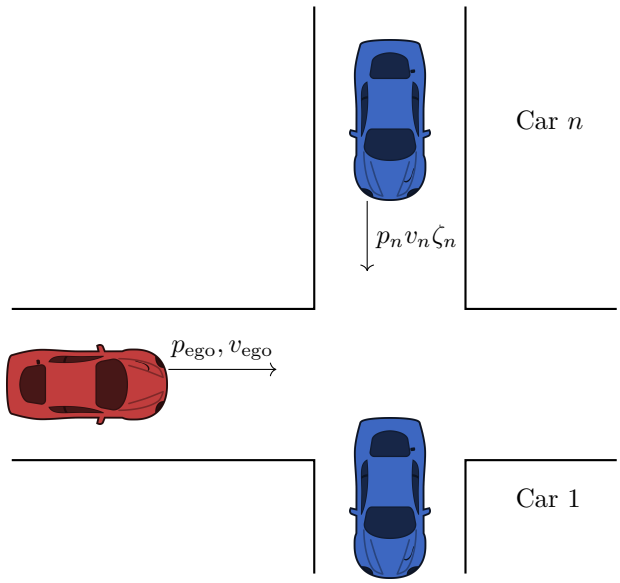


Figure 4.1: General description of the states in a simple intersection. The ego vehicle in red is controlled by the agent, while the blue vehicles are the other vehicles crossing the same intersection. Each blue vehicle is described by an index n, a position p_n, a velocity v_n and hidden intention ζ_n

.

From Section 3.1, the state space contains all the information necessary about the agent and environment to be able to transition from any given state s to the next state s'. In the scenario shown in Figure 4.1, the red car on the horizontal lane represents the ego vehicle controlled by the agent while the blue cars on the vertical lane are the other vehicles which the ego vehicle needs to interact with in order to cross the intersection.

A simplified description of the state,

$$s = (p_{\text{ego}}, v_{\text{ego}}, \{p_n, v_n, \zeta_n\}_{n=1}^{N}), \tag{4.1}$$

consists of positions state p_{ego} and p_n, where subscript *ego* and n denotes the ego vehicle and the index of the surrounding vehicle up to N vehicles.

Instead of using a Cartesian coordinate system to describe the position p_{ego} and p_n, relative distance measures are proposed. This way, the state space is generalizable to different intersection designs, e.g., the angle of incidence and the number of crossing points. The velocity of ego v_{ego} and of all the other traffic participants v_n are also necessary to be able to predict what position they will be in the next state. Finally, the intention of all the other participants ζ_n. As mentioned in Section 1.2, ζ_n encapsulates information such as stop sign, traffic light or even inattention in to one variable. Paper D shows a comparison between two fully observable MDPs, one with intention and the other one without and the results show that having an intention state reduce number of collisions.

4.1.2 Action space

One limitation of deep Q-learning is the requirement for a discrete action space. In various AD studies [37], it is common practice to define the action space in terms of different discretized acceleration requests. To address the challenge of managing a potentially high-dimensional action space with fine discretization or a coarse action space with large steps between accelerations, Paper A proposes using short-term goals as actions: $\{$*'take way', 'yield', 'follow car* $\{1, \ldots, N\}$*'*$\}$.

These short-term goals represent high-level objectives, such as driving through an intersection (take way), stopping at the start of an intersection (yield) or drive behind a specific car (follow car n). Each high-level action is translated into a set of parameters that are input into a sliding mode controller in Paper A and a MPC in Paper B, which then generates the appropriate acceleration to control the ego car.

4.1.3 Transition model

The transition model is initially unknown, and RL is employed to implicitly learn this model by taking actions in the environment from different states, recording the resulting rewards, and noting the subsequent state transitions. In this work, the environment is a simulator, and the primary objective for the agent is to learn the transition dynamics of other vehicles, which depend on their intentions ζ. These intentions are modeled as predetermined actions governed by an IDM. Using the IDM to guide predetermined actions enhances the

dynamics of vehicle interactions, bringing them closer to real-world scenarios.

Intelligent Driver Model

The IDM is a widely used car-following model in traffic flow theory and simulation [38]. It describes how drivers adapt their speed and spacing based on the distance to the vehicle ahead. IDM helps in understanding and predicting traffic dynamics, optimizing traffic flow, and developing ADAS functions. In this thesis the IDM is used to model the general behavior of surrounding vehicles. The IDM models a vehicle n's position p_n and velocity v_n as

$$\dot{p}_n = v_n \tag{4.2}$$

$$\dot{v}_n = a_{\max}\Big(1 - \Big(\frac{v_n}{v_n^{\text{desired}}}\Big)^{\delta} - \Big(\frac{d^*(v_n, \Delta v_n)}{d_n}\Big)^2\Big) \tag{4.3}$$

$$\text{with } d^*(v_n, \Delta v_n) = d_0 + v_n T_{\text{gap}} + \frac{v_n \Delta v_n}{2\sqrt{a_{\max}\alpha_b}}$$

where v_n^{desired} is the desired velocity, d_0 is the minimum distance between cars, T_{gap} is the desired time gap to the vehicle in front, $a_{\max}$ is the maximum vehicle acceleration, d_n is the distance to the vehicle in front, $\Delta v_n = v_n - v_{n-1}$ is the velocity difference between vehicle n and the vehicle directly in front $n-1$, and α_b and δ are model parameters for comfortable deceleration or acceleration.

The acceleration can be simplified into two terms: an interaction term for when there is a vehicle in front

$$\begin{aligned} a_n^{\text{int}} &= -a_{max}\Big(\frac{d^*(v_n, \Delta v_n)}{d_n}\Big)^2 \\ &= -a_{max}\Big(\frac{d_0 + v_n T_{\text{gap}}}{d_n} + \frac{v_n \Delta v_n}{2\sqrt{a_{\max}\alpha_b} d_n}\Big)^2 \end{aligned} \tag{4.4}$$

and free road term, when there is no leading vehicle

$$a_n^{\text{free}} = a_{\max}\Big(1 - \Big(\frac{v_n}{v_n^{\text{desired}}}\Big)^{\delta}\Big). \tag{4.5}$$

In this thesis, IDM's application ensures realistic simulation of surrounding vehicle behavior, which is crucial for testing and validating the proposed decision-making algorithms for autonomous vehicles.

4.1.4 Observation model

The observation space is closely aligned with the state space $\mathcal{S}$, but it includes some added noise and excludes the intention state ζ_n, because current sensors cannot directly detect the intentions of other drivers. The observation

$$o = (p_{\text{ego}}, v_{\text{ego}}, \{\hat{p}_n, \hat{v}_n\}_{n=1}^{N}), \tag{4.6}$$

encompasses all observable elements of the state, detailed in (4.1). The ego vehicle accurately observes its own states, while it observes noisy measurements of the positions $\hat{p}_n$ and velocities $\hat{p}_n$ of the surrounding vehicles. These measurements are given by:

$$\hat{p}_n = p_n + \epsilon_{\text{p}}, \tag{4.7}$$

$$\hat{v}_n = v_n + \epsilon_{\text{v}} \tag{4.8}$$

where, $\epsilon_{\text{p}} \sim \mathcal{N}(0, \sigma_p^2)$ and $\epsilon_{\text{v}} \sim \mathcal{N}(0, \sigma_v^2)$.

4.1.5 Reward function

The design of the reward function is pivotal as it determines the value associated with each state, ultimately shaping the driving policy of the agent. A well-crafted reward function is instrumental in guiding the agent towards achieving its objectives effectively. The reward model in this thesis is formulated based on terminal states, including reaching the goal r^{succ}, collision events r^{fail}, timeouts $r^{\text{t.o.}}$, and on non-terminal states r^{comf} e.g, every step update the agent has not reached a terminal state. These terminal states play a critical role in defining the success or failure of the agent's driving behavior and are accordingly reflected in the reward structure. One example of a reward function used in this thesis, from Paper B, is

$$r_t = \begin{cases} r^{\text{succ}} & \text{on success,} \\ r^{\text{fail}} & \text{on failure,} \\ r^{\text{t.o.}} & \text{on timeout, i.e. } \tau \geq \tau_m, \\ r^{\text{comf}} & \text{on non-terminating updates,} \end{cases} \tag{4.9}$$

where the reward for reaching the goal is typically assigned a relatively high value, such as $r^{\text{succ}} = 1$, while the reward for colliding is assigned a very low

value, such as $r^{\text{fail}} = -1$. These two rewards establish the maximum and minimum possible rewards for a single simulation run. If the defined reward values are too large, the Q-values in (3.8) can become large and cause the gradients to grow and potentially lead to instability [39]. Although the absolute values can be chosen arbitrarily, the most important aspect is their relative values to each other. This relative scaling ensures that the agent prioritizes achieving the goal and avoiding collisions appropriately. The other rewards for timeout and non-terminal states are then usually set somewhere between r^{succ} and r^{fail}, such as $r^{\text{t.o.}} = -0.1$. The reward for non-terminal states r^{comf} was utilized differently: Paper A used r^{comf} to account for comfort by assigning a higher negative reward for high acceleration jerk, while Paper B added a negative reward was given if the MPC predicted a collision.

4.2 Simulation environment

The simulation environment in this thesis, first introduced in Paper A, places an agent at an intersection tasked with reaching the goal across the intersection while interacting with up to N other cars on the intersecting lane. At the beginning of each episode, up to N vehicles are initialized with initial positions p_n^0 distributed along the intersecting lane, starting velocities v_n^0 and a deterministic policy that defines their intention ζ_n. Each vehicle, including the ego vehicle, adheres to the IDM (Section 4.1.3) aimed at maintaining specific speeds and safe distances, with the maximum acceleration is capped at $5m/s^2$ to ensure comfort and safety under normal driving conditions. For example, if a vehicle's intention ζ_n is yield, it would set the IDM for the distance to the object in front d_n to the distance to the start of the intersection, and its velocity v_{n-1} would be set to 0. Alternatively, a vehicle with a take way intention would follow the IDM only considering the vehicle directly in front of it.

During the simulation, whenever a vehicle in the perpendicular lane crosses the intersection, it is removed from the environment. Subsequently, a new vehicle is spawned at the start of that lane at a random time, with new initial values for position p_n^0, velocity v_n^0, desired velocity v_n^{desired} and intention ζ_n. This dynamic spawning process ensures that the traffic scenario continually evolves, presenting varying challenges and interactions for the agent navigating the intersection. Each episode continues until a terminal state is reached,

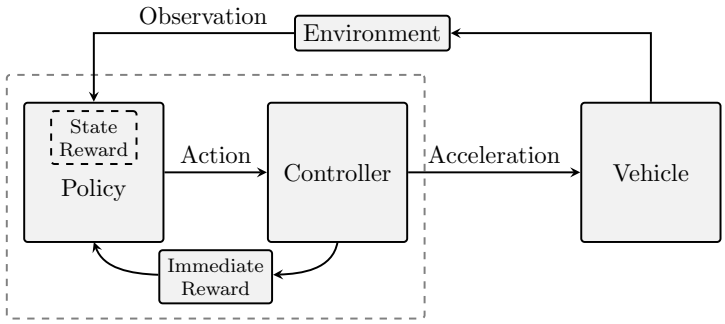


Figure 4.2: Representation of the decision-making architecture. Policy represents the DQN that chooses an action that is sent to a Controller that can either be a sliding mode controller or MPC. The Environment represents the simulation environment in which the vehicles operate, but can be replaced by the real world. The State Reward is the reward given by the terminal state ($r^{\text{succ}}, r^{\text{fail}}, r^{\text{t.o.}}$) from the environment and the Immediate Reward r^{comf} is given by the controller.

which can be reaching the goal, collision, or timeout for Paper A-C and goal, collision, safe stop, or deadlock for Paper D. Rewards are assigned based on a predefined reward function designed to reinforce desired behaviors.

4.3 Deep Q-learning approach

To find a driving policy π for the POMDP detailed in the previous section, both Paper A and Paper B used deep Q-learning, described Chapter 3.2.1, using a decision-making architecture shown in Figure 4.2. Paper A proposed a network architecture with shared weights and LSTM layer to approximate the Q-function. The shared weight effectively reduces the input space, as the state for each car n can be initially weighted the same as any other car, independent of the order it comes into the neural network. While the LSTM layer has the role of utilizing the previous states to implicitly estimate a hidden state that could possibly encapsulate the intention of other cars.

As mentioned in Chapter 4.1.2, the actions in Paper A are controlled by a sliding mode controller while Paper B uses an MPC to generate a velocity profile for a short time horizon. The sliding mode controller, which is used

interchangeably with the IDM in this thesis, aims to maintain a minimum distance from a target vehicle by controlling acceleration in a manner similar to IDM.

4.3.1 Model Predictive Control

MPC is an optimization-based control technique where an Optimal Control Problem (OCP) is repeatedly solved over a receding limited time horizon, starting from the current system state. For every time instance, a mathematical model of the controlled system is used to simulate future states over a finite horizon, while a sequence of control inputs is selected and optimized given an objective cost function. The first element in the sequence of control inputs is then applied to the real system, and a new OCP with an updated state is solved at the next time instance.

MPC faces challenges as a mixed integer problem, where calculating the optimal path for all possible actions is computationally intensive. DQN on the other hand, only handles discrete actions. While DQN cannot guarantee safety, it excels at choosing actions with the highest utility (Q-value). The reward function in DQN can incorporate the predicted outcome from the MPC model, penalizing suboptimal actions. However, if experience shows a better outcome than the model predicts, DQN can opt for an action that leads to a better total reward. Paper B integrates the MPC cost as a negative immediate reward, allowing the DQN to balance this cost with the high-level goal of reaching the target. This enables the DQN to choose actions that are generally beneficial at a high level, even if they may not seem optimal to the MPC.

The agents from Paper A and Paper B are evaluated on two scenarios: a single intersection and a double intersection with two crossing points, both shown in Figure 1.1. The next section presents the experimental results from Paper A and Paper B for both intersections.

4.4 Results from simulation

The results from Table 4.1 show that the proposed DQN agent from Paper A found a policy that successfully crossed a single intersection 96.1% of the time, with 2.8% resulting in collisions and 1.1% resulting in timeouts. When comparing agents trained with and without an LSTM layer, those with the

LSTM succeeded in 3 out of 4 attempts where those without it would fail. This demonstrates that deep Q-learning has great potential for creating decision-making agents capable of navigating intersections. OA key contribution of this thesis is the proposal of network architectures specifically designed for this problem. In Paper A, an architecture employing shared weights is introduced, and the results shown in Figure 4.3 reveal that utilizing a network with shared weights for processing information about observed vehicles notably enhances convergence speed. The combination of shared weights with other improvements, such as dropout and experience replay, is crucial for achieving these performance gains.

Figure 4.3: The figure shows that the success rate for a network with shared weights (brown line) converge faster than the fully connected network structure which do not share weights (turquoise line).

As mentioned previously, the controller from Paper A could only consider one car at a time, which placed a heavy burden on the DQN compensate by frequently switching actions. In contrast, in Paper B, the MPC showed significant improvement in handling more complex intersections. For instance, in the double intersection, the MPC agent succeeded 95.2% of the time with a collision rate of 3.6%, compared to the sliding mode controller, which only succeeded 90.9% of the time with a collision rate of 8.3%.

Controller	Success Rate		Collision Rate	
	Single	Double	Single	Double
SM (Paper A)	96.1%	90.9%	2.8%	8.3%
MPC (Paper B)	97.3%	95.2%	1.2%	3.6%

Table 4.1: Average success rates and collision rates for a fully trained DQN agent driving through a single and double intersections. If the agent failed to reach the goal or collide within a given time, the terminal state was classified as a timeout.

4.5 Discussion

Observing the behavior of a fully trained agent from Paper A and Paper B provides the insight that the path of the ego vehicle can be segmented into four zones, illustrated in Figure 4.4. Starting from the right, Zone 0 represents the *safe zone*, where the ego vehicle is out of danger and can resume nominal driving. Zone 1 is the *conflict zone*, where a collision with another vehicle is possible. Zone 2, the *critical decision zone*, is the final opportunity for the vehicle to either stop or proceed through the intersection. The size of zone 2 is determined by the minimum distance required for the vehicle to come to a complete stop before entering the conflict zone, ensuring sufficient time for safe decision-making. Lastly, Zone 3, the *information gathering zone*, is situated furthest from the intersection. Here, the agent can observe how other vehicles behave over time to estimate their intentions.

The goal is to reach Zone 0. To achieve this, the agent aims to minimize the time spent in Zone 1 if there is a chance of intersection with another car. Our actions are formulated as short-term goals, designed for comfortable use with lower acceleration rates. The size of Zone 2 depends on the vehicle's current speed, which is influenced by its behavior in Zone 3.

Now, two conflicting strategies emerge: to minimize time in Zone 1, the agent desires a high speed entering the intersection. However, it also seeks a low speed to reduce the size of Zone 2 and the critical decision period. If the intentions of other vehicles are known, the stochasticity in Zone 1 would be eliminated, transforming the problem into a scheduling task aimed at creating a velocity profile that minimizes the time required to cross. However, since the intentions of other vehicles are inherently stochastic, the next chapter offers a promising approach by accounting for this uncertainty and optimizing

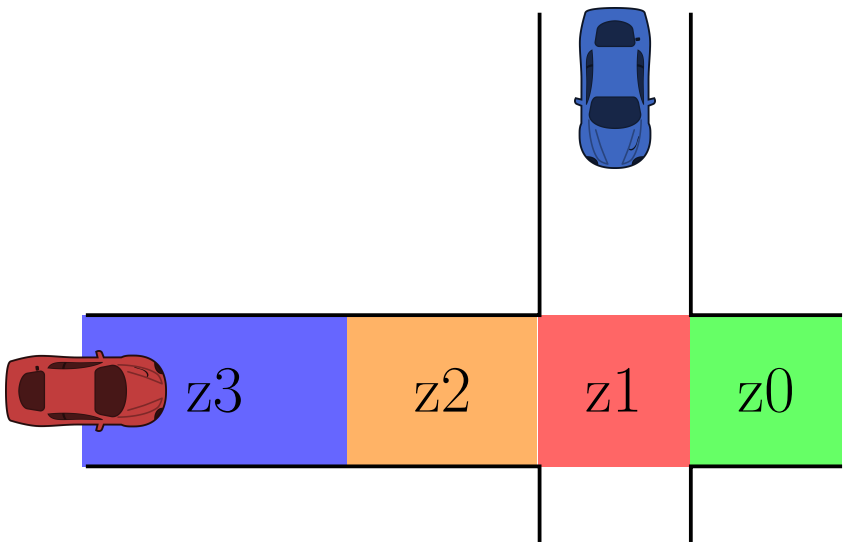


Figure 4.4: Intersection scenario divided into zones describing what is required of the decision maker in different zones

decision-making in dynamic traffic scenarios.

CHAPTER 5

Accounting for the uncertainty

The previous chapter formulated the POMDP and introduced how deep Q-learning methods can be utilized to create decision-making agents capable of navigating intersections. A significant advantage of RL methods is their scalability to different scenarios through appropriate training. However, a drawback of deep Q-learning methods, is the use of neural networks, which provide a black-box solution without indicating any confidence or uncertainty in their decisions [40].

This chapter presents two approaches to estimating the uncertainty: Paper C addresses the uncertainty in the output of the DQN, and Paper D addresses the uncertainty in the intention estimation that is fed as an input to the DQN. To demonstrate the effect of accounting for uncertainty, the results include two types of experiments. In the first type, the trained agent is evaluated on scenarios within the training set. In the second type, the agent is evaluated on scenarios outside the training set. Both types of experiments are conducted using the same simulation environment described in Section 4.2.

5.1 Estimating the uncertainty of the Q-value

To estimate the uncertainty in the Q-value of the trained agent, Paper C employed statistical bootstrapping to train an ensemble of neural networks on different subsets of the available experience. This ensemble provides a distribution over the estimated Q-values for any input. A better Bayesian posterior is obtained by adding different Randomized Prior Functions (RPF) to each ensemble member [41]. The Q-values of each ensemble member k is then calculated as the sum of two neural networks, f and p, with the same architecture but different values on the weights θ_k and $\hat{\theta}_k$, i.e.,

$$Q_k(s,a) = f(s,a;\theta_k) + \beta p(s,a;\hat{\theta}_k). \tag{5.1}$$

Here, the weights θ_k of network f are trainable, and the weights $\hat{\theta}_k$ of the prior network p are fixed to the randomly initialized values. A parameter β scales the importance of the networks. With the two networks, the loss function in Eq. 3.8 becomes

$$L(\theta_k) = \mathbb{E}_M\Big[(r + \gamma \max_{a'}(f_{\theta_k^-} + \beta p_{\hat{\theta}_k})(s',a') - (f_{\theta_k} + \beta p_{\hat{\theta}_k})(s,a))^2\Big]. \tag{5.2}$$

The training process of an ensemble RPF is described by Algorithm 1. An ensemble of K DQN are first initialized randomly. Each ensemble member is also assigned a separate experience replay memory buffer m_k. For each new episode, a random ensemble member ν is selected and used to take greedy actions throughout the episode, which corresponds to an approximate Thompson sampling approach to the exploration vs. exploration dilemma. Each new experience $e = (s_i, a_i, r_i, s_{i+1})$ is then added to the separate replay buffers m_k with probability p_{add}. The trainable weights of each ensemble member are then updated by uniformly sample a mini-batch M of experiences and using SGD.

The agent's uncertainty in choosing different actions can be defined as the coefficient of variation $c_{\text{v}}(s,a)$ of the Q-values of the ensemble members [42]. A threshold $c_{\text{v}}(s,a)$ can then be used to determine whether an action has an acceptable level of uncertainty.

A high uncertainty, where $c_{\text{v}}(s,a) > c_{\text{v}}^{\text{safe}}$, indicates that (s,a) is outside the training distribution. When the agent is fully trained, the policy π chooses

Algorithm 1 Ensemble RPF training process

for $k \leftarrow 1$ to K **do**
    Initialize θ_k and $\hat{\theta}_k$ randomly
    $m_k \leftarrow \{\}$
$i \leftarrow 0$
while networks not converged **do**
    $s_i \leftarrow$ initial random state
    $\nu \sim \mathcal{U}\{1, K\}$
    while episode not finished **do**
        $a_i \leftarrow \text{argmax}_a \, Q_\nu(s_i, a)$
        $s_{i+1}, r_i \leftarrow \textsc{StepEnvironment}(s_i, a_i)$
        for $k \leftarrow 1$ to K **do**
            if $p \sim \mathcal{U}(0,1) < p_{\text{add}}$ **then**
                $m_k \leftarrow m_k \cup \{(s_i, a_i, r_i, s_{i+1})\}$
            $M \leftarrow$ sample mini-batch from m_k
            update θ_k with SGD and loss $L(\theta_k)$
        $i \leftarrow i + 1$

actions by maximizing the mean of the Q-values of the ensemble members, with the restriction $c_{\text{v}}(s,a) < c_{\text{v}}^{\text{safe}}$, i.e.,

$$\begin{aligned} \pi = \underset{a}{\text{argmax}} \, \frac{1}{K} \sum_{k=1}^{K} Q_k(s,a), \\ \text{s.t.} \quad c_{\text{v}}(s,a) < c_{\text{v}}^{\text{safe}}. \end{aligned} \tag{5.3}$$

If no action within the action space satisfies the restriction, a predefined backup action a_{safe} is chosen instead. This a_{safe} can be something like a collision avoidance measure, such as emergency braking.

5.1.1 Results for scenarios within the training set

This section shows the results of the ensemble RPF method compared to the DQN method used in Paper A, but without the LSTM layer. The ensemble RPF method outperforms the DQN method when the agents are tested on scenarios that are similar to the training scenarios. When the fully trained ensemble RPF agent is exposed to situations that are outside the training

distribution, the agent indicates a high uncertainty and chooses safe actions, whereas the DQN agent sometimes collides with other vehicles.

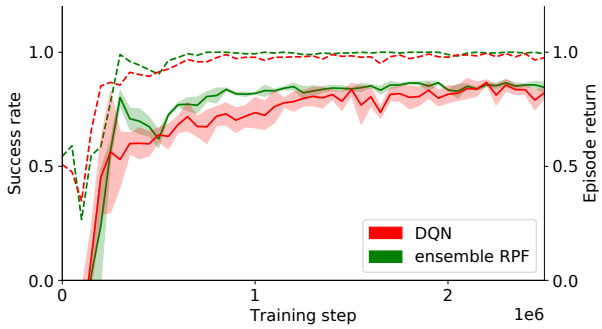


Figure 5.1: Proportion of test episodes where the ego vehicle reached its goal (dashed), and episode return (solid), over training steps for the ensemble RPF and DQN methods. The shaded areas show the standard deviation for 5 random seeds.

The average return and the average proportion of episodes where the ego vehicle reached the goal, as a function of number of training steps, is shown in Figure 5.1, for the test episodes. The figure also shows the standard deviation for 5 random seeds, which generates different sets of initial parameters of the networks and different training episodes, whereas the test episodes are kept fixed. The results show that the ensemble RPF method both learns faster, yields a higher return, and causes fewer collisions than the DQN method.

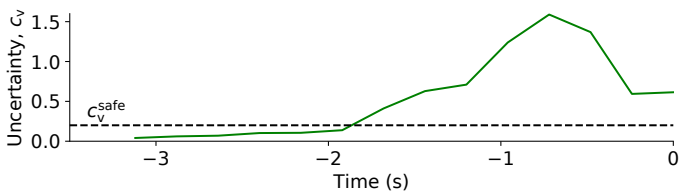
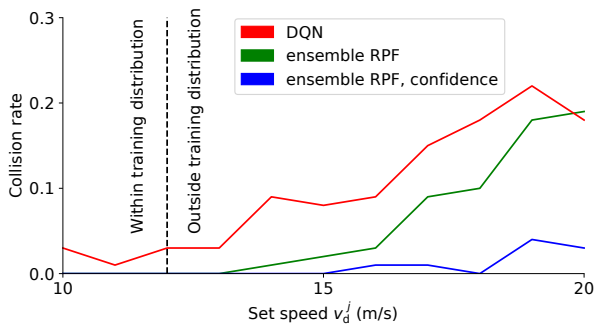


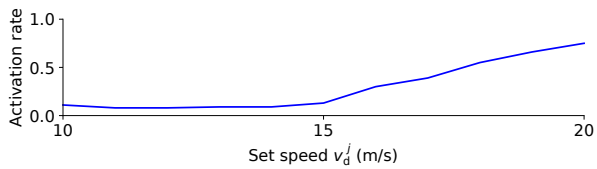
Figure 5.2: Uncertainty c_v during the time steps before one of the collisions in the test episodes, within the training distribution. The collision occurs at $t = 0$ s.

Upon analyzing the scenarios with collisions using the DQN agent, it was observed that in one particular example, the agent fails to brake sufficiently in zone 3 (from Figure 4.4) and takes an incorrect action in zone 2, resulting in a collision in zone 1. In contrast, in the same scenario with the ensemble RPF agent, Figure 5.2 show that the estimated uncertainty increases significantly during the time before the collision (zone 2), when the incorrect actions was taken. By applying the confidence criterion, the agent, aware of high uncertainty, brakes early enough in zone 2, thus avoiding collisions. This confidence criterion was applied to all test episodes, effectively eliminating all collisions for scenarios within the training set.

5.1.2 Results for scenarios outside the training set



(a) Proportion of collisions.



(b) Proportion of episodes where a_{safe} was used at least once.

Figure 5.3: Performance of the ensemble RPF agent, with and without the confidence criterion, and the DQN agent, in test episodes with different velocities for the surrounding vehicles.

To evaluate the agents' ability to detect unseen situations, a fully trained

ensemble RPF agent was tested in scenarios outside the training distribution. The same testing scenarios as in the previous section were used, with the exception that the speed of the surrounding vehicles was set to a single deterministic value, which was varied during different runs in the range $v_n \in [10, 20]$ m/s. The proportion of collisions as a function of set speed of the surrounding vehicles is shown in Figure 5.3, along with the proportion of episodes where the confidence criterion was violated at least once. The figure demonstrates that by accounting for the uncertainty of the Q-value and the use of the confidence criterion some collisions can be avoided.

5.2 Estimating the uncertainty of the intention state

The ensemble RPF agent avoided collisions by not taking a bad action in zone 2, by accounting for the uncertainty of the Q-value estimate. However, can further reductions in collisions be achieved by accounting for uncertainty in the input? In Paper D, the uncertainty regarding the intention state ζ_n is managed using a belief state represented as a probability distribution over possible states. This distribution, referred to as intention distributions, is generated using a particle filter. Based on these distributions, two methods were proposed: QMDP-IE and QID.

QMDP-IE is derived from the QMDP algorithm [29]. This derivation leverages the assumption that the intention state ζ_n can be determined after the ego vehicle has crossed the intersection analyzing the actions taken by the other vehicle, i.e., the intention can be determined by post-data analysis. This assumption allows the QMDP algorithm to blend elements of POMDPs and MDPs. The QMDP algorithm is comprised by the following steps:

1. Oracle DQN Training: The weights θ_{MDP} used in $Q(a, s; \theta_{\text{MDP}})$, are trained using the true state s, which includes the true intention state ζ_n.

2. Expected Q-value Computation: QMDP calculates the Q-value for each belief state, which accounts for uncertainty in the true state. It then computes the expected Q-value for each action by averaging the Q-values of the potential states, weighted by their probabilities in the belief state.

QMDP-IE modifies the QMDP approach:

1. Utilization of Oracle DQN: QMDP-IE uses the same Q-values $Q(a, s; \theta_{\text{MDP}})$ as calculated by the Oracle DQN.

2. Intention State Estimation $D^{\text{IE}}(b)$: Instead of considering the full distribution of potential states (belief state b), QMDP-IE estimates a single intention state. This estimation $D^{\text{IE}}(b)$ is done using a threshold $\zeta_{\text{threshold}}$ on the intention distribution

$$D^{\text{IE}}(b) = \zeta_n^{\text{IE}} = \begin{cases} \text{yield} & \text{if } \zeta_n^{\text{yd}} > \zeta_{\text{threshold}}, \\ \text{take way} & \text{otherwise.} \end{cases} \tag{5.4}$$

   This ζ_n^{IE}, together with the observable state o, is then passed as to the NN to estimate the Q-value.

The threshold value $\zeta_{\text{threshold}}$ in the QMDP-IE method is a critical parameter that directly influences the agent's behavior, ranging from aggressive to passive. The agent is tested with different $\zeta_{\text{threshold}}$ values, and the outcomes are observed in terms of success rate and success time shown in 5.4. A lower threshold value ($\zeta_{\text{threshold}} = 0,5$) means the agent relies on the most likely estimation of the intention state, resulting in aggressive behavior. This can lead to quicker success times but might increase the risk of collisions or other negative outcomes. A higher threshold values ($\zeta_{\text{threshold}} = 0.9$) makes the agent more cautious, potentially leading to safer but slower success times. In summary, while both QMDP and QMDP-IE rely on the Oracle DQN to predict Q-values, QMDP-IE reduces the complexity of decision-making by focusing on a single estimated intention state rather than a full belief state distribution reducing the computational complexity.

In the QID method, the intention state ζ_n^{ID} is derived from the belief state b using an approximator $D^{\text{ID}}(b)$. The derived intention state ζ_n^{ID} is obtained as the marginal distribution of intention from the set of particles $\{x_m\}_{m=1}^{M}$:

$$D^{\text{ID}}(b) = \zeta_n^{\text{ID}} = [\zeta_{\text{tw}}^{\text{ID}}\ \zeta_{\text{yd}}^{\text{ID}}] = \sum_{m=1}^{M} w_{n[m]}[x(\zeta)]_{n[m]}, \tag{5.5}$$

where each particle x_m represents a possible state, and each particle has an associated weight $w_{n[m]}$, indicating its probability out of M number of total particles. The intention state used by the particle $[x(\zeta)]_{n[m]}$ is specified using a one-hot vector, which represents the intention state in a binary format where

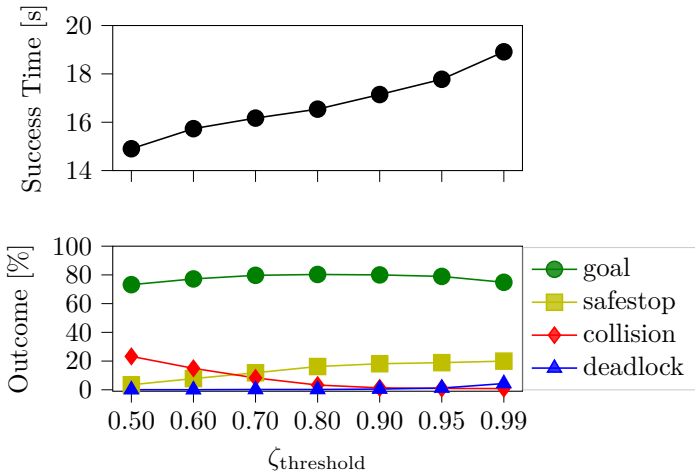


Figure 5.4: The success time and outcome percentages of QMDP-IE for different values of $\zeta_{\text{threshold}}$ values. The top figure show that increasing $\zeta_{\text{threshold}}$ result in slower success times, whereas the lower figure indicates that higher $\zeta_{\text{threshold}}$ reduces the collision rate while simultaneously increasing the safe stop rate.

one element is '1' (indicating the active intention) and the others are '0'. The intention state for "take-way" is then $\zeta_{\text{tw}}^{\text{ID}} \in [0, 1]$ and consequently, "yield" becomes $\zeta_{\text{yd}}^{\text{ID}} = 1 - \zeta_{\text{tw}}^{\text{ID}}$. This process allows the QID method to derive a clear and actionable intention state from a distribution of possible states.

Both QMDP-IE and QID were evaluated on the intersection scenario introduced in Section 4.2, specifically designed to include at least one car on the intersecting lane with the same time to intersection as the ego vehicle. This scenario increases the likelihood of a collision if the intentions of the other vehicles are not accurately considered, thereby encouraging more safe stops.

5.2.1 Results within the training set

This section presents the experimental results, starting with a comparison of the results of QMDP-IE and QID. QMDP is used as a baseline algorithm for QMDP-IE, while QPF, an algorithm trained on the whole belief state b is used

Table 5.1: Result summary for scenarios within the training set

Experiments	Goal reached	Safe stop	Collision	Deadlock	Success Time
	%	%	%	%	s
Oracle DQN	84.50	14.45	1.05	0.00	15.49
QMDP-IE	80.00	18.15	1.35	0.50	17.14
QMDP	71.95	15.05	5.70	7.30	20.56
QID	85.60	10.40	3.70	0.30	16.64
QPF	63.50	21.80	12.15	2.55	16.61

as the baseline for QID. Unlike QMDP-IE, the QID algorithm does not require access to the true intent during training.

As shown in Table 5.1, both QMDP-IE and QID beat their respective baseline algorithm in both higher goal reached and lowest collision percentage. QMDP-IE collision rate 1.35 % was very close to the oracle DQN 1.05 % making it as good as knowing the intention in these test cases. QID experienced more collisions than QMDP-IE but fewer than QMDP. This indicates that while training on ground truth is preferable, QID is a suitable alternative when ground truth data is not available.

5.2.2 Results for scenarios outside the training set

To investigate how the algorithms perform when exposed to behaviors they are not trained for, a scenario is introduced where a car is allowed to overtake another car in front, thereby violating the assumption of the IDM. In the previous experimental scenario, the order of the other cars was structured such that if one car yielded, all following cars would stop behind it. This makes it possible to collide with a car that started behind a yielding car, but it also creates larger gap in between cars, that was not possible before.

In this overtake scenario, both QID and QPF exhibit relatively low collision rates at 2.53 % and 3.7 % respectively. QMDP-IE and QMDP show slightly higher collision rates of 4.6 % and 5.95 % respectively, which is higher than QID and QPF and slightly higher than their own performance in the trained scenarios, detailed in Table 5.1. Because QMDP-IE uses the network from the Oracle DQN, it performs well when the intention is within the training

Table 5.2: Results with an overtake agent

Experiments	Goal reached %	Safe stop %	Collision %	Deadlock %	Time s
Oracle DQN	89.80	0.15	10.05	0.00	16.01
QMDP-IE	94.55	0.20	4.60	0.65	16.62
QMDP	79.70	10.20	5.95	4.15	19.84
QID	96.89	0.11	2.53	0.47	16.38
QPF	96.15	0.10	3.70	0.05	17.62

set, while QID and QPF was trained on the uncertainty making the policy more cautious in zone 3 which lead to an opening in zone 2 leading to the goal. This suggests that QID is slightly better at taking passive actions in zone 3, which can change the intention distribution and reduce the uncertainty for the next state. This makes QID more effective at handling scenarios outside the training set than QMDP-IE.

Overall, QID proves to be better than QMDP-IE in scenarios outside the training set, offering more robustness to untrained behaviors. Meanwhile, QMDP-IE provides a structured approach with moderate collision rates and higher computational efficiency, performing well when the intentions are within the training set.

CHAPTER 6

Generalizing over different scenarios

In the previous chapter, the experiments demonstrated how an agent can utilize the uncertainty measurement to reduce the number of collisions. This chapter explores how to identify when an agent is acting in a scenario outside its training set.

Let's consider that a company is designing Level 5 AD agents. As mentioned in Section 1.1, L5 requires the AV to be able to drive anywhere. The company has already designed two RL agents that have learned to drive at L4 in the USA and UK. Now, the company wants to deploy a new RL agent in India. Though all the RL agents are concerned with the same task, i.e. driving, the models encompassing driver behaviors, traffic rules, signs, etc., can differ for each. For example, UK and India have left-handed traffic, while the USA has right-handed traffic. However, learning a new controller specifically for every new geographic location is computationally expensive and time-consuming, as both data collection and learning take time. Thus, the company might use the models learned for UK and USA, to estimate the model for India, and use it further to build a new RL agent. Hence, being able to transfer the source models to the target environment allows the company to use existing knowledge to build a new agent faster while using less resources. This problem

Algorithm 2 Maximum Likelihood Estimation for Model-based Transfer RL (MLEMTRL)

Input: weights $\boldsymbol{w}^0$, m source MDPs $\mathcal{M}_s$, data D_0, discount factor γ, iterations T.
for $t = 0, \dots, T$ **do**
    // Stage 1: Model Estimation //
    $\boldsymbol{w}^{t+1} \leftarrow \text{Optimiser}(\log \mathbb{P}(D_t \mid \Sigma_{i=1}^m w_i \mu_i), \boldsymbol{w}^t)$
    Estimate the MDP: $\mu^{t+1} = \sum_{i=1}^m w_i \mu_i$
    // Stage 2: Model-based Planning //
    Compute the policy: $\pi^{t+1} \in \arg\max_{\pi} V^{\pi}_{\mu^{t+1}}$
    // Control //
    Observe $s_{t+1}, r_{t+1} \sim \mu^*(s_t, a_t), a_t \sim \pi^{t+1}(s_t)$
    Update the dataset $D_{t+1} = D_t \cup \{s_t, a_t, s_{t+1}, r_{t+1}\}$
return An estimated MDP model μ^T and a policy π^T

of model transfer from source models to a target environment to plan efficiently is address in Paper E.

6.1 Approach

In this example, the driving model for each country is defined by its own MDP. A set of source MDPs $\mathcal{M}_s \triangleq \{\mu_i\}_{i=1}^m$ is used to create a convex hull of MDPs $\mathcal{C}(\mathcal{M}_s)$, where each MDP μ_i acts as a boundary. The proposed approach, Maximum Likelihood Estimation for Model-based Transfer RL (MLEMTRL), involves using model transfer RL to determine the unknown target MDPs $\mu^* \in \mathcal{C}(\mathcal{M}_s)$ position within this convex hull of MDPs and then find an optimal policy π^* for that MDP. This method relies on having a diverse set of MDPs to effectively span the convex hull of models, thereby facilitating the discovery of optimal policies between MDPs. For example, downtown driving behavior in one country may closely resemble that in another. For the case when μ^* is close but outside $\mathcal{C}(\mathcal{M}_s)$, see Paper E.

The MLEMTRL algorithm, outlined in Algorithm 2, consists of a *model estimation* stage and a *planning* stage.

Model estimation: This stage estimates a MDP $\hat{\mu}$ from a compact subset

of m source MDPs, $\mu' \subset \mathcal{M}_s$, utilizing data from experience D_t gathered by taking actions in the environment. Let's define the convex hull as $\mathcal{C}(\mathcal{M}_s) \triangleq \{\mu_1 w_1 + \ldots + \mu_m w_m \,|\, \mu_i \in \mathcal{M}_s, w_i \geq 0, i = 1, \ldots, m, \sum_{i=1}^{m} w_i = 1\}$. Then, the corresponding MLE problem with the corresponding likelihood function is

$$\hat{\mu} \in \underset{\mu' \in \mathcal{C}(\mathcal{M}_s)}{\arg\max} \log \mathbb{P}(D_t \,|\, \mu'), D_t \sim \mu^*. \tag{6.1}$$

Since $\mathcal{C}(\mathcal{M}_s)$ induces a compact subset of model parameters $\mathcal{M}' \subset \mathcal{M}$, (6.1) leads to a constrained maximum likelihood estimation problem [43]. It implies that if the parameter corresponding to the target MDP is in $\mathcal{M}'$, it can be correctly identified. For continuous state-action MDPs, a linear-Gaussian likelihood is used. In this context, let d_s be the dimensionality of the state-space, $\boldsymbol{s} \in \mathbb{R}^{d_s}$ and d_a be the dimensionality of the action-space. Then, the mean function $\mathbf{M}$ is a $\mathbb{R}^{d_s} \times \mathbb{R}^{d_a + d_s}$ matrix. The mean visitation count to the successor state $\boldsymbol{s}'_t$ when an action $\boldsymbol{a}_t$ is taken at state $\boldsymbol{s}_t$ is given by $\mathbf{M}(\boldsymbol{a}_t, \boldsymbol{s}_t)$. The corresponding covariance matrix, denoted by $\mathbf{S}$, is of size $\mathbb{R}^{d_s} \times \mathbb{R}^{d_s}$. The log-likelihood is then expressed as follows:

$$\log \mathbb{P}(D_t \,|\, \mathbf{M}, \mathbf{S}) = \log \prod_{i=1}^{t} \frac{\exp\Big(-\frac{1}{2}\boldsymbol{v}^\top \mathbf{S}^{-1} \boldsymbol{v}\Big)}{(2\pi)^{d_s/2} |\mathbf{S}|^{1/2}},$$
$$\text{where } \boldsymbol{v} = \boldsymbol{s}'_i - \mathbf{M}(\boldsymbol{a}_i, \boldsymbol{s}_i).$$

As the optimization problem involves weighing multiple source models together, a weight vector $\boldsymbol{w} \in [0, 1]^m$ is introduced, with the usual property that $\boldsymbol{w}$ sum to 1.

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & \log \mathbb{P}(D_t \,|\, \Sigma_{i=1}^{m} w_i \mu_i), D_t \sim \mu^*, \mu_i \in \mathcal{M}_s, \\ \text{s.t.} \quad & \sum_{i=1}^{m} w_i = 1, w_i \geq 0. \end{aligned} \tag{6.2}$$

Because of the constraint on $\boldsymbol{w}$, this is a constrained nonlinear optimization problem. Any optimizer algorithm, denoted by Optimizer, can be used for this purpose. When an appropriate model μ^{t+1} has been identified at time step t, the next stage of the algorithm involves model-based planning in the estimated MDP.

Planning: This stage computes the policy π^{t+1} that maximizes the expected reward given the value function $V^{\pi}_{\mu^{t+1}}$ based on μ^{t+1}. After computing π^{t+1},

the agent executes actions in the environment to acquire new experiences D_{t+1}. These steps are repeated T times before producing a final estimated MDP model μ^T and a policy π^T is given.

Given the model, μ^t and the associated reward function $\mathcal{R}$, the optimal value function of μ^t can be computed iteratively as [21]:

$$V^*_{\mu^t}(s) = \max_a \sum_{s'} \mathcal{T}^a_{s,s'} \Big(\mathcal{R}(s,a) + \gamma V^*_{\mu^t}(s') \Big). \tag{6.3}$$

The fixed-point solution to Eq. (6.3) is the optimal value function, where $\mathcal{T}$ is the transition function introduced in Section. 4.1.3. When the optimal value function has been obtained, one can simply select the action maximizing the action-value function. Let π^{t+1} be the policy selecting the maximizing action for every state, then π^{t+1} is the policy the model-based planner will use at time step $t+1$.

6.2 Experiments and results

The proposed algorithm is evaluated on 10 different tasks in Deepmind Control Suite [44], and the results for two Linear Quadratic Regulator(LQR) tasks, *dm_LQR_2_1* and *dm_LQR_6_2*, are presented. These environments have continuous states and actions, where the tasks involve controlling a two-joint, one-actuator system and a six-joint, two-actuator system toward the center of the platform, respectively. They feature unbounded control inputs and rewards, with state spaces $s \in \mathbb{R}^4$ and $s \in \mathbb{R}^{12}$, respectively. In the Deepmind Control Suite, each task varies by seed, which determines the stiffness of the joints.

The performance is compared to baseline algorithms such as a posterior sampling RL method (PSRL), multi-task soft-actor critic (MT-SAC) [45], [46] and a modified MT-SAC-TRL that uses data from the novel task during learning. In PSRL, a new model is sampled from the prior at every round, learning in the target MDP from scratch. The aim of this experiment is to identify improvements in learning speed, jumpstart and asymptotic improvements. *Learning Speed Improvement:* A learning speed improvement would be indicated by the algorithm reaching its asymptotic convergence with less data. *Jumpstart Improvement:* A jumpstart improvement can be verified by the behavior of the algorithm during the early learning process. In particular, if

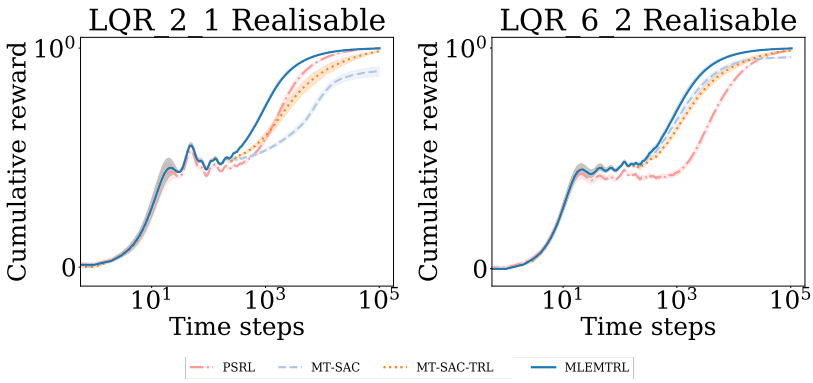


Figure 6.1: The average cumulative reward at every time step computed for two LQR tasks from Deepmind control suit. The shaded regions represent the standard error of the average cumulative reward at the time step.

the algorithm starts at a better solution than the baseline, or has a simpler optimization surface, it may more rapidly approach better solutions with much less data.*Asymptotic Improvement:* An asymptotic improvement would mean the algorithm converges asymptotically to a superior solution to that one of the baseline.

The results experiment are shown in Figure 6.1, where the performance metric is the average cumulative reward at each time step over 10^5 time steps, with the shaded region representing the standard deviation. In these experiments, MLEMTRL demonstrates a clear advantage over all baselines in terms of learning speed improvement and asymptotic performance when compared to MT-SAC, but shows negligible differences in jumpstart improvements.

This shows that MLEMTRL is able to construct a model for an RL agent using a set of source models, provided that the target model is sufficiently similar to the source models.

CHAPTER 7

Discussion

Chapter 4, 5 and 6 introduce different methods to create a decision-making agent for navigating through intersections using deep Q-learning, while considering the uncertainty of its own decisions and the prediction of the other drivers intentions. This chapter highlights some differences and synergies of the different methods.

7.1 Guaranteeing safety

Safety is the most important factor of a AD system, but since RL is a data driven approach it is very difficult to guarantee safety in the same way e.g., a MPC can. Safety validation in a data driven approach can be very costly as it is usually done by driving many miles and showing statistical confidence in the safety metrics [47], [48], e.g., number of interventions. That is why this section will clarify where the work in this thesis would fit in by first defining a system architecture for AD, its modules and a short summary of the ISO 26262 standard. Then, place itself within the system and motivate how safety can be guarantied.

The architecture of an autonomous driving system can be divided into

perception, planning and control [5], [49]. The perception module is responsible for sensing and mapping the environment with the use of sensors such as LIDARs, cameras, radars etc. The raw data from the sensors are then processed though various sensor fusion techniques to generate a representation of the environment, e.g., position, velocity of other traffic participants while also describing the road such as width and distance to the next intersection. This information is then used by the planner to create a driving strategy of how to transverse through the world. However, the information from the sensors are often noisy, with false positives and false negatives making it difficult for the planner.

Tactical planning can be divided into three categories, the proactive, active and reactive [50]. A proactive module would be something like a precautionary safety module that interprets the information about the environment and create constraints that is sent to the active planner, like driveable area, allowed speeds and actions [51]. These constraints are generated from a set of safety goals and rules, making this the first layer of protection that can ensure safety. The role of the active planner is to take this sets of allowed actions and prescribe the behavior of the vehicle through decisions such as drive, yield or stop. The goal of these high level decisions is to optimize metrics such as comfort, fuel consumption and time to goal. These decisions are then sent to a motion planner that generates a safe dynamically feasible path for the vehicle for a shorter planning horizon of around 0.1s. At the same time, a reactive, collision avoidance, module make sure that the chosen decision and path does lead to any collisions [52]. Unlike the decision maker, the collision avoidance module main goal is to identify imminent danger [53] and therefore has access to more aggressive actions like emergency braking to ensure safety.

In the industry today the main standard for functional safety in motorized vehicles is the ISO 26262 standard, titled "Road vehicles – Functional safety" [54]. It uses a Automotive Safety Integrity Level (ASIL) to classify the inherent safety risk in an automotive system and the functions or modules of such a system. The ASIL classification is used to express the level of risk reduction required to prevent a specific hazard, from ASIL D to ASIL A. ASIL D represents the highest hazard level and ASIL A the lowest. There is a level with no safety relevance and only standard Quality Management processes are required, this level is referred to as QM.

Although safety is the most important requirement for enabling autonomous

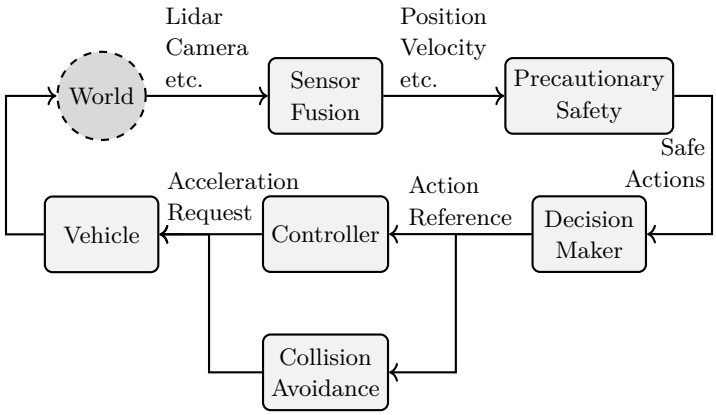


Figure 7.1: Representation of the system architecture.

driving, the work in this paper does not make any safety guarantees. Instead, it is proposed that the decision-making algorithms presented in this paper be used in the system architecture shown in Figure 7.1. This approach allows higher ASIL to be applied to the precautionary safety and collision avoidance modules, while the decision-making algorithms focus primarily on comfort. As a result, the ASIL classification for the decision-making components could be at lower levels, potentially even classified as QM in the best case.

7.2 Designing the reward function and terminal states

The reward function introduced in Chapter 4.1.5 is a crucial component that significantly influences the behavior and performance of a reinforcement learning agent. By carefully designing and tweaking the reward function in (4.9), the agent can be guided towards desirable behaviors and optimize its decision-making policy.

The results from Chapter 4 and 5 show a collision rate of $1 - 3\%$, which may seem high for AVs. However, within the proposed system architecture from Figure 7.1, this collision rate can be interpreted as interventions by a collision avoidance system, such as emergency braking. This interpretation can

be achieved by adjusting the simulation parameters, for example, increasing the size of the cars or redefining the collision state to represent a collision avoidance intervention state.

7.3 Modular models in autonomous vehicles

There are two common strategies for creating and deploying DQNs to the real world: training a comprehensive model that encapsulates everything or training smaller, specialized models and switching between them as needed. The proposed MLEMTRL algorithm from Chapter 6 is a step towards the latter approach of using smaller models. Combined with the uncertainty measurements in Chapter 5, instead of reverting to a default action when uncertainty is high, the agent can instead trigger a model change. MLEMTRL can then be used to determine which model to switch to, ensuring a more adaptive and robust decision-making process.

This approach allows for modularity and flexibility, enabling easier updates and maintenance since individual models can be refined or replaced without affecting the entire system. Additionally, smaller models are less likely to overfit to irrelevant details present in a larger, comprehensive dataset, leading to more generalizable and robust performance in their specific domains. They also require less computational power and memory, making them more suitable for deployment on AVs with limited resources. Smaller models are generally easier to interpret and debug, facilitating understanding of the decision-making process and identifying any issues or biases, which is an important property to have when developing AVs.

CHAPTER 8

Concluding remarks and future work

This thesis introduces how RL-based methods can be used to develop a decision-making agent and evaluates their efficacy in learning when to drive across intersections, with a focus on managing the uncertainty of other drivers' intentions. By answering the three research questions presented in Chapter 1.3.

In response to Q1: *How can RL techniques be used to develop a decision-making agent that effectively navigates intersections without explicitly estimating the intention state of other vehicles?* The intersection navigation problem is formulated as a POMDP, where observable states include positions and velocities of the vehicles in the scenario, and unobservable states is the intentions of surrounding drivers. In Chapter 4, a deep Q-learning approach is introduced to solve the POMDP, with short-term goals, as discreet actions, translated into reference points and constraints for a controller. Two controllers are implemented and compared: one using sliding mode and another using MPC. The results show that the DQN effectively generates a policy for driving across an intersection crossing with dynamic behavior of other traffic participants, and its performance improved when combined with a robust controller like MPC. Additionally, the hidden state in the LSTM layer of the RL agent incorporates estimations of driver intentions, which improved the performance of the RL

agent in dynamic traffic environments.

A significant advantage of RL methods is their scalability to different scenarios through appropriate training. However, a drawback of deep Q-learning methods is the use of neural networks, which provide a black-box solution without indicating any confidence or uncertainty in their decisions. This limitation is addressed by answering Q2: *How can an RL agent utilize the uncertainty in its predictions and actions to enhance decision-making in complex environments?* Chapter 5 presents two approaches to address the uncertainty: an ensemble method addresses the uncertainty in the output of the DQN, and a belief-based method addresses the uncertainty in the intention estimation that is fed as an input to the DQN. The results demonstrate that accounting for uncertainty can significantly improve the agent's performance, especially in scenarios outside the training set, by avoiding collisions and improving decision-making robustness.

Chapter 6 addresses Q3: *How can an RL agent handle situations it has not been trained on?* The MLEMTRL algorithm is introduced to address the challenge of deploying RL agents in new environments by leveraging knowledge from previously trained models. The approach involves constructing a convex hull of source MDPs and using model transfer RL to identify the most relevant model for the target environment. This method is evaluated in various tasks, including a continuous state-action MDPs, demonstrating improved learning speed and asymptotic performance. The results show that MLEMTRL can construct a new model for an RL agent using a set of source models, provided the target model is sufficiently similar to the source models. This capability can potentially enable the efficient deployment of AD agents in new environments, reducing the need for extensive data collection and training specific to each new geographic location.

Overall, the research presented in this thesis contributes to the advancement of RL techniques for autonomous driving in diverse and dynamic environments. The findings show the critical roles of accurate intention estimation, effective uncertainty management, and the strategic use of transfer learning to build robust and versatile autonomous driving systems. While RL methods alone may still have challenges in guaranteeing safety for autonomous driving, integrating them with MPC and other active safety systems can transform RL into a powerful tool for creating comfortable and enjoyable rides for the passengers, paving the way for safer and more efficient transportation solutions.

8.1 Future work

Future research should focus on transitioning from simulation environments to real-world implementations. While simulations offer a controlled setting for developing and testing algorithms, real-world driving presents unpredictable variables and complexities. Implementing and refining these models in actual driving scenarios will be crucial for validating their effectiveness and reliability. This step will involve rigorous testing, continuous learning, and adaptation to ensure the autonomous systems can handle diverse and dynamic real-world conditions, ultimately moving closer to the widespread adoption of safe and efficient autonomous vehicles.

Additionally, the integration of language prediction models, specifically using transformers and attention mechanisms, along with driver monitoring systems (DMS), can enhance the prediction of driver intentions. Transformers, with their powerful attention mechanisms, can effectively handle sequential data and capture complex dependencies. By leveraging these models, it may be possible to interpret and predict driver behaviors based on a broader range of contextual cues.

Driver monitoring systems can provide critical real-time data on driver behavior, including eye movement, head position, and other physiological indicators. Combining this data with sensor inputs and verbal communication or textual descriptions of driver actions can create a comprehensive understanding of driver intentions. Transformers can process and correlate these different data types, improving the accuracy of intention estimation, particularly in complex or ambiguous driving scenarios.

Moreover, employing transformers trained on extensive datasets could facilitate the development of more sophisticated algorithms capable of predicting and adapting to diverse driving behaviors. This approach could significantly enhance the overall safety and efficiency of autonomous driving systems by providing a more nuanced and dynamic understanding of the driving environment. Integrating DMS with advanced language models could also help in identifying potential risks and ensuring timely interventions, thus improving the overall robustness and reliability of autonomous vehicles.

CHAPTER 9

Summary of included papers

This chapter provides a summary of the included papers.

9.1 Paper A

Tommy Tram, Anton Jansson, Robin Grönberg, Mohammad Ali, and Jonas Sjöberg
Learning Negotiating Behavior Between Cars in Intersections using Deep Q-Learning
Published in 2018 21st International Conference on Intelligent Transportation Systems (ITSC),
pp. 3169–3174, Nov. 2018.
 DOI: 10.1109/ITSC.2018.8569316.

This paper concerns automated vehicles negotiating with other vehicles, typically human driven, in crossings with the goal to find a decision algorithm by learning typical behaviors of other vehicles. The vehicle observes distance and speed of vehicles on the intersecting road and use a policy that adapts its speed along its pre-defined trajectory to pass the crossing efficiently. Deep Q-

learning is used on simulated traffic with different predefined driver behaviors and intentions. The results show a policy that is able to cross the intersection avoiding collision with other vehicles 98% of the time, while at the same time not being too passive. Moreover, inferring information over time is important to distinguish between different intentions and is shown by comparing the collision rate between a Deep Recurrent Q-Network at 0.85% and a Deep Q-learning at 1.75%.

The thesis author contributed with the problem formulation, proposed algorithms and the writing of the paper.

9.2 Paper B

Tommy Tram, Ivo Batković, Mohammad Ali, and Jonas Sjöberg
Learning When to Drive in Intersections by Combining Reinforcement Learning and Model Predictive Control
Published in 2019 IEEE Intelligent Transportation Systems Conference (ITSC),
pp. 3263–3268, Oct. 2019.
 DOI: 10.1109/ITSC.2019.8916922.

In this paper, we propose a decision making algorithm intended for automated vehicles that negotiate with other possibly non-automated vehicles in intersections. The decision algorithm is separated into two parts: a high-level decision module based on reinforcement learning, and a low-level planning module based on model predictive control. Traffic is simulated with numerous predefined driver behaviors and intentions, and the performance of the proposed decision algorithm was evaluated against another controller. The results show that the proposed decision algorithm yields shorter training episodes and an increased performance in success rate compared to the other controller.

The thesis author contributed with the problem formulation, proposed algorithms, implementation of the simulation and reinforcement learning algorithm, and the writing of the paper.

9.3 Paper C

Carl-Johan Hoel, Tommy Tram, and Jonas Sjöberg
Reinforcement Learning with Uncertainty Estimation for
Tactical Decision-Making in Intersections
Published in 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC),
pp. 1-7, Sep. 2020.
 DOI: 10.1109/ITSC45102.2020.9294407.

This paper investigates how a Bayesian reinforcement learning method can be used to create a tactical decision-making agent for autonomous driving in an intersection scenario, where the agent can estimate the confidence of its decisions. An ensemble of neural networks, with additional randomized prior functions (RPF), are trained by using a bootstrapped experience replay memory. The coefficient of variation in the estimated Q-values of the ensemble members is used to approximate the uncertainty, and a criterion that determines if the agent is sufficiently confident to make a particular decision is introduced. The performance of the ensemble RPF method is evaluated in an intersection scenario and compared to a standard Deep Q-Network method, which does not estimate the uncertainty. It is shown that the trained ensemble RPF agent can detect cases with high uncertainty, both in situations that are far from the training distribution, and in situations that seldom occur within the training distribution. This work demonstrates one possible application of such a confidence estimate, by using this information to choose safe actions in unknown situations, which removes all collisions from within the training distribution, and most collisions outside of the distribution.

The thesis author contributed with the problem formulation, proposed algorithms, simulation implementation and the writing of the paper.

9.4 Paper D

Tommy Tram, Maxime Bouton, Jonas Fredriksson, Jonas Sjöberg, and Mykel Kochenderfer
Belief State Reinforcement Learning for Autonomous Vehicles in Intersections
Submitted to IEEE Transactions on Intelligent Transportation Systems,

This paper investigates different approaches to find a safe and efficient driving strategy through an intersection with other drivers. Because the intentions of the other drivers to yield, stop, or go are not observable, we use a particle filter to maintain a belief state. We study how a reinforcement learning agent can use these representations efficiently during training and evaluation. This paper shows that an agent trained without any consideration of the intentions of others is both slower at reaching the goal and results in more collisions. Four algorithms that use a belief state generated by a particle filter are compared. Two of the algorithms have access to the intention only during training while the others do not. The results show that explicitly trying to predict the intention gave the best performance in terms of safety and efficiency.

The thesis author contributed with the problem formulation, proposed algorithms, simulation and experimental implementations, and the writing of the paper.

9.5 Paper E

Hannes Eriksson, Tommy Tram, Debabrota Basu, Mina Alibeigi, and Christos Dimitrakakis
Reinforcement Learning in the Wild with Maximum Likelihood-based Model Transfer
Published in 2024 International Conference on Adaptive Agents and Multi-Agent Systems (AAMAS),
pp. 516–524, May. 2024.

For decision-problems with insufficient data, it is imperative to take into account not only what you know but also what you do not know. In this work, ways of transferring knowledge from known, existing tasks to a new setting is studied. In particular, for tasks such as autonomous driving, the optimal controller is conditional on things such as, the physical properties of the vehicle, the local and regional traffic rules and regulations and also on the specific scenario trying to be solved. Having separate controllers for every combination of these conditions is intractable. By assuming problems with similar structure, we are able to leverage knowledge attained from similar tasks to guide learning for new tasks. We introduce a maximum likelihood estimation procedure

for solving Transfer Reinforcement Learning (TRL) of different types. This procedure is then evaluated over a set of autonomous driving settings, each of which constitutes an interesting scenario for autonomous driving agents to make use of external information. We prove asymptotic regret bounds for proposed method for general structured probability matrices in a specific setting of interest.

The thesis author contributed with the problem formulation and experimental analysis of the paper.

References

[1] O. Y. I-Cheng Lin and C. Joe-Wong, "Mixed-autonomy era of transportation," Traffic21, Tech. Rep., 2022.

[2] K. Heineke, N. Laverty, T. Möller, and F. Ziegler, *The future of mobility: Mobility evolves*, Apr. 2023.

[3] D. J. Fagnant and K. Kockelman, "Preparing a nation for autonomous vehicles: Opportunities, barriers and policy recommendations," *Transportation Research Part A: Policy and Practice*, vol. 77, pp. 167–181, 2015, ISSN: 0965-8564.

[4] J. Janai, F. Güney, A. Behl, and A. Geiger, *Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art*. Now Publishers Inc., 2020.

[5] 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.

[6] "2019 traffic safety culture index," AAA Foundation for Traffic Safety, Washington DC, Tech. Rep. 202-638-5944, 2019.

[7] "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," On-Road Automated Driving (ORAD) Committee, Tech. Rep., 2021.

[8] L. Fletcher *et al.*, "The mit–cornell collision and why it happened," *Journal of Field Robotics*, vol. 25, pp. 775–807, Oct. 2008.

[9] S. Kammel *et al.*, "Team annieway's autonomous system for the 2007 darpa urban challenge," *Journal of Field Robotics*, vol. 25, pp. 615–639, Sep. 2008.

[10] L. Gressenbuch and M. Althoff, "Predictive monitoring of traffic rules," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2021.

[11] M. Liebner, M. Baumann, F. Klanner, and C. Stiller, "Driver intent inference at urban intersections using the intelligent driver model," in *IEEE Intelligent Vehicles Symposium (IV)*, 2012.

[12] S. Hoermann, D. Stumper, and K. Dietmayer, "Probabilistic long-term prediction for autonomous vehicles," in *IEEE Intelligent Vehicles Symposium (IV)*, 2017.

[13] C. B. Browne *et al.*, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[14] C. Hubmann, M. Becker, D. Althoff, D. Lenz, and C. Stiller, "Decision making for autonomous driving considering interaction and uncertain prediction of surrounding vehicles," in *IEEE Intelligent Vehicles Symposium (IV)*, 2017.

[15] 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.

[16] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a Frenét frame," in *IEEE Int. Conf. on Robot. and Automat.*, 2010.

[17] F. Damerow and J. Eggert, "Risk-aversive behavior planning under multiple situations with uncertainty," in *IEEE Int. Conf. on Intell. Transp. Syst. (ITSC)*, 2015.

[18] I. Batkovic, M. Zanon, M. Ali, and P. Falcone, "Real-time constrained trajectory planning and vehicle control for proactive autonomous driving with road users," in *European Control Conference (ECC)*, 2019.

[19] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, 2016.

[20] B. Paden, M. Čáp, S. Z. Yong, D. 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.

[21] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.

[22] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, "Navigating occluded intersections with autonomous vehicles using deep reinforcement learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[23] V. Mnih *et al.*, *Playing atari with deep reinforcement learning*, 2013.

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

[25] M. J. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2015.

[26] P. Zhu, X. Li, P. Poupart, and G. Miao, *On improving deep reinforcement learning for pomdps*, 2018.

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

[28] A. Wang, A. C. Li, T. Q. Klassen, R. T. Icarte, and S. A. Mcilraith, "Learning belief representations for partially observable deep RL," vol. 202, Jul. 2023.

[29] 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.

[30] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

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

[32] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 2004.

[33] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems (NIPS)*, vol. 12, 1999.

[34] A. R. Cassandra, M. L. Littman, and N. L. Zhang, "Incremental pruning: A simple, fast, exact method for POMDPs," in *Conference on Uncertainty in Artificial Intelligence (UAI)*, 1997.

[35] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[36] M. Hessel *et al.*, "Rainbow: Combining improvements in deep reinforcement learning," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[37] 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.

[38] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical Review E*, vol. 62, pp. 1805–1824, 2 2000.

[39] H. V. Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver, "Learning values across many orders of magnitude," in *Advances in Neural Information Processing Systems (NIPS)*, 2016.

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

[41] I. Osband, J. Aslanides, and A. Cassirer, "Randomized prior functions for deep reinforcement learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[42] C.-J. Hoel, K. Wolff, and L. Laine, "Tactical decision-making in autonomous driving by reinforcement learning with uncertainty estimation," in *IEEE Intelligent Vehicles Symposium (IV)*, 2020.

[43] J. Aitchison and S. Silvey, "Maximum-likelihood estimation of parameters subject to restraints," *The annals of mathematical Statistics*, vol. 29, no. 3, pp. 813–828, 1958.

[44] Y. Tassa *et al.*, "Deepmind control suite," *arXiv preprint:1801.00690*, 2018.

[45] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, 2018.

[46] T. Yu *et al.*, "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Conference on robot learning*, 2020.

[47] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," in *SAE International Journal of Transportation Safety*, 1. 2016, vol. 4, pp. 15–24.

[48] D. Åsljung, J. Nilsson, and J. Fredriksson, "Using extreme value theory for vehicle level safety validation and implications for autonomous vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 2, no. 4, pp. 288–297, 2017.

[49] D. Kortenkamp, R. G. Simmons, and D. Brugali, "Robotic systems architectures and programming," in *Springer Handbook of Robotics, 2nd Ed.*, 2008.

[50] K. Berntorp, T. Hoang, R. Quirynen, and S. Di Cairano, "Control architecture design for autonomous vehicles," in *IEEE Conference on Control Technology and Applications (CCTA)*, 2018.

[51] Z. Shiller, F. Large, S. Sekhavat, and C. Laugier, "Motion planning in dynamic environments," in *Autonomous Navigation in Dynamic Environments*, C. Laugier and R. Chatila, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 107–119, ISBN: 978-3-540-73422-2.

[52] M. Brännström, E. Coelingh, and J. Sjöberg, "Model-based threat assessment for avoiding arbitrary vehicle collisions," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 3, pp. 658–669, 2010.

[53] M. Brännström, E. Coelingh, and J. Sjöberg, "Decision-making on when to brake and when to steer to avoid a collision," *International Journal of Vehicle Safety*, vol. 7, no. 1, pp. 87–106, 2014.

[54] "Road vehicles — functional safety," International Organization for Standardization, Standard, Dec. 2018.

Part II

Papers

PAPER A

Learning Negotiating Behavior Between Cars in Intersections using Deep Q-Learning

Tommy Tram, Anton Jansson, Robin Grönberg, Mohammad Ali, and Jonas Sjöberg

Published in 2018 21st International Conference on Intelligent Transportation Systems (ITSC),
pp. 3169–3174, Nov. 2018.
 DOI: 10.1109/ITSC.2018.8569316.

The layout has been revised.

Abstract

This paper concerns automated vehicles negotiating with other vehicles, typically human driven, in crossings with the goal to find a decision algorithm based on learning typical behavior of other vehicles. The vehicle observes distance and speed of vehicles on the intersecting road and use a policy that adapts its speed along its pre-defined trajectory to pass the crossing efficiently. Deep Q-learning is used on simulated traffic and the results show that policies can be trained to successfully drive comfortably through an intersection, avoiding collision with other cars and not being too passive. The policies generalize over different types driver behaviors and intentions. Moreover, to enable inferring information over time, a Deep Recurrent Q-Network is tested and compared to the Deep Q-learning. The results show that a Deep Recurrent Q-Network succeeds in three out of four attempts where a Deep Q-Network fails.

1 Introduction

The development of autonomous driving vehicles is fast and there are regularly news and demonstrations of impressive technological progress, see eg [1]. However, one of the largest challenges does not have to do with the autonomous vehicle itself but with the human driven vehicles in mixed traffic situations. Human drivers are expected to follow traffic rules strictly, but in addition they also interact with each other in a way which is not captured by the traffic rules, [2], [3]. This *informal* traffic behavior is important, since the traffic rules alone may not always be enough to give the safest behavior. This motivates the development of control algorithms for autonomous vehicles which behave in a "human-like" way, and in this paper we investigate the possibilities to develop such behavior by training on simulated vehicles.

In [4] they raises two concerns when using Machine learning, specially Reinforcement learning, for autonomous driving applications: ensuring functional safety of the Driving Policy and that the Markov Decision Process model is problematic, because of unpredictable behavior of other drivers. In the real world, intentions of other drivers are not always deterministic or predefined.

Depending on their intention, different actions can be chosen to give the most comfortable and safe passage through an intersection. They also noted that in the context of autonomous driving, the dynamics of vehicles is Markovian but the behavior of other road users may not necessarily be Markovian. In this paper we solve these two concerns using a Partially Observable Markov Decision Process (POMDP) as a model and Short Term Goals (STG) as actions. With a POMDP the unknown intentions can be estimated using observations and that has shown promising results for other driving scenarios [5]. The POMDP is solved using a model-free approach called Deep (Recurrent) Q-Learning. With this approach a driving policy can be found using only observations without defining the states. Since we do not train on human driven vehicles, the results presented here cannot be considered human-like, but the general approach, to train the algorithms using traffic data, is shown working, and a possible next step could be to start with the pre-tuned policies from this work, and to continue the training in real traffic crossings.

2 Overview

This paper starts by introducing the system architecture and defining the actions, observations and POMDP in Section 3. The final strategy of what action to take at a given situation is called a policy and is described in Section 4. The method used for finding this policy is called Q-learning, which uses a neural network to approximate a Q-value and is described in Section 5 together with techniques used to improve the learning, such as Experience replay, Dropout and a recurrent layer called Long short term memory (LSTM). We then present the simulation, reward function and neural network configurations in Section 6. The results are then presented in Section 7 comparing the effect of the methods mentioned in Section 5. Finally the conclusion and brief discussion is presented in Section 8.

3 Problem formulation

The objective is to drive along a main road that has one or two intersections with crossing traffic and control the acceleration in a way that avoids collisions in a comfortable way. All vehicles are assumed to drive along predefined paths on the road where they can either speed up or slow down to avoid collisions in

the crossings. In this section the system architecture is defined along with the environment, observation and actions.

3.1 System architecture

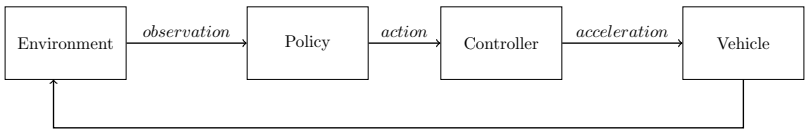


Figure 1: Representation of the architecture

Environment is defined as the world around the ego vehicle, including all vehicles of interest and the shape/type of the intersection. The environment can vary in different ways, e.g. number of vehicles and intersection or the distance to intersections. The environment is defined by the simulation explained in section 6.1. We assume that the ego vehicle receives observations from this environment at each sampling instant, as shown in Fig. 1. A policy then takes these observations and chooses a high level action that is defined in more detail in section 3.3. These actions are sent to a controller that calculates the appropriate acceleration request given to the ego vehicle, which will influence the environment and impact how other cars behave.

3.2 Actions as Short Term Goals

Motivated by the insight that the ego vehicle has to drive before or after other vehicles when passing the intersection, decisions on the velocity profile is modeled by simply keeping a distance to other vehicles until they pass. This is done by defining the actions as Short Term Goals (STG), eg. keep set speed or yield for crossing car. This allows the properties of comfort on actuation and safety to be tuned separately, making the decision a classification problem. The actions are then as followed:

- *Keep set speed*: Aims to keep a specified maximum speed $v_{\max}$, using a simple P-controller

$$a_p^e = K(v_{max} - v^e) \tag{A.1}$$

where a_p^e is the acceleration request and v^e is the velocity of ego vehicle towards the center of the intersection, while K is a proportional constant.

- *Keep distance to vehicle N*: Will control the acceleration in a way that keeps a minimum distance to a chosen vehicle N, a *Target Vehicle*, and can be done using a sliding mode controller, where the acceleration request is:

$$a_{sm}^e = \frac{1}{c_2}(-c_1 x_2 + \mu sign(\sigma(x_1, x_2))) \tag{A.2}$$

$$\text{where} \begin{cases} x_1 = p^t - p^e \\ x_2 = v^t - v^e \end{cases}$$

where p^e and p^t is the position of ego and target vehicle respectively, shown in Fig. 2, and v^t is the velocity of target vehicle. c_1 together with c_2 are calibration parameters that can be set to achieve wanted performance with a surface plane σ

$$\sigma = c_1 x_1 + c_2 x_2 \tag{A.3}$$

The final acceleration request a^e is achieved by a min arbitration between eq. A.1 and A.2

$$a^e = \min(a_{sm}^e, a_p^e) \tag{A.4}$$

For more detailed information about sliding mode see [6]. To distinguish between different cars to follow, each other vehicle will have its own action.

- *Stop in front of intersection*: Stops the car at the next intersection. Using the same controller as eq. A.4 while setting $v^t = 0$ and p^t to start of intersection, the controller can bring ego vehicle to a comfortable stop before the intersection.

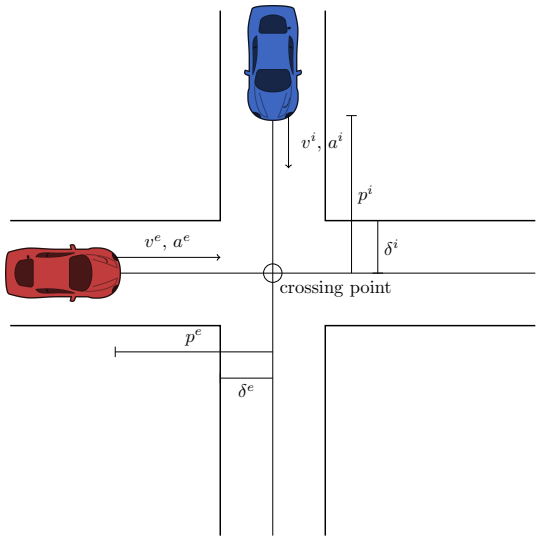


Figure 2: Observations that makes the state

3.3 Observations that make up the state

A human driver is, generally, good at assessing a scenario and it is hard to pin-point what information is used in their assessment. Therefore some assumptions are made on which features that are interesting to observe. The observation o_t at time t is defined as:

$$o_t = [\ p_t^e \quad v_t^e \quad a_t^e \quad \delta^e \quad p_t^i \quad v_t^i \quad a_t^i \quad \delta^i \quad a_{t+1}^{e,A}\ ]^T \tag{A.5}$$

With notations as follows: consider Fig. 2, position of ego p_t^e and other vehicle p_t^i are defined as distance to common reference point, called *crossing point*, where i is an index of the other vehicle. The start of intersection for ego δ^e and other vehicle δ^i also uses the crossing point as reference. These are relevant in case a driver would choose to yield for other vehicles, then they would most likely stop before the start of intersection. The velocity v^e and acceleration a^e of ego vehicle and velocity v^i and acceleration a^i of the other vehicles are observed to include the dynamics of different actors. The last feature in the observation, $a_{t+1}^{e,A}$, is the ego vehicle's predicted acceleration for each possible action A, which can be used to account for comfort in the

decision.

3.4 Partially Observable Markov Decision Processes

The decision making process in the intersection is modeled as a POMDP. A POMDP works like a Markov Decision Process (MDP) [7] in most aspects, but the full state is not observable.

At each time instant, an action, $a_t \in \mathcal{A}$, is taken, which will influence to which new state vector, s_{t+1}, the system evolves and changes the environment. Each action a_t from a state s_t has a value called the reward r_t, which is given by a reward function $\mathcal{R}_t$.

One of the unobservable states could be the intentions of other drivers approaching the intersection. The state can only be perceived partially through observations $o_t \in \Omega$ with the probability distribution of receiving observation o_t given an underlying hidden state $s_t : o_t \leftarrow \mathcal{O}(s_t)$, where $\mathcal{O}(s_t)$ is the probability distribution.

4 Finding the optimal policy

Assuming the states are not known, we want a model-free method of finding a policy, and for this we use reinforcement learning. The goal is to have an agent learn how to maximize the future reward by taking different actions in a simulated environment. Details on the simulation environment used is described in Section 6.1. The strategy of which action to take given a state is called a policy π and can be modeled in two ways:

- As stochastic policy $\pi(a|s) = \mathcal{P}[\mathcal{A} = a|\mathcal{S} = s]$
- As deterministic policy $a = \pi(s)$

The standard assumption is made that the future reward is discounted by a factor γ per time step, making the discounted future reward $\mathcal{R}_t = \sum_t^\tau \gamma^{t-1} r_t$, where τ is the time step where the simulation ends, e.g. when the agent crosses an intersection safely.

Similar to [8], the optimal action-value function $Q^*(s_t, a_t)$ is defined as the maximum expected reward achievable by following a policy π given the state s_t and taking an action a_t:

$$Q^*(s_t, a_t) = \max_{\pi} \mathbb{E}[\mathcal{R}_t | s_t, a_t, \pi] \tag{A.6}$$

Using the Bellman equation, $Q^*(s_t, a_t)$ can be defined recursively. If we know $Q^*(s_t, a)$ for all actions a that can be taken in state s_t, then the optimal policy will be one that takes the action a_t that gives the highest immediate and discounted expected future reward $r_t + \gamma Q^*(s_{t+1}, a_{t+1})$. This gives us:

$$Q^*(s_t, a_t) = \mathbb{E}[r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) | s_t, a_t] \tag{A.7}$$

The optimal policy π^* is then given by taking actions according to an optimal $Q^*(s_t, a_t)$ function:

$$\pi^*(s_t) = \arg\max_{a_t} Q^*(s_t, a_t) \tag{A.8}$$

5 Method

From eq. A.8, the optimal policy is defined by taking an action that has the highest expected Q-value. Because the Q-value is not known, a non linear function approximation, such as a neural network, is used to estimate the Q-function. The method is known as Deep Q-Learning [8] and in this section we will briefly describe Q-learning and methods used to improve the learning such as, Experience replay, dropout and Long-, Short- Term Memory (LSTM).

5.1 Deep Q-learning

Deep Q-Learning uses a neural network to approximate the Q-function. This neural network is called Deep Q-Network (DQN). The Q-function approximated by the DQN is denoted as $Q(s_t, a_t|\theta^\pi)$, where θ^π are the neural network parameters for policy π. The state s_t is the input to the DQN and the output is the Q-value for each action $\mathcal{A}$.

5.2 Experience Replay

A problem with Deep Q-Learning, looking at eq. A.8, is that with a small change in the Q-function, can effect the policy drastically. The distribution of future training samples will be greatly influenced by the updated policy. If the network only trains on recent experiences, a biased distribution of samples

is used. Such behavior can cause unwanted feedback loops which can lead to divergence of the trained network [9].

Proposed by [8] in order to make DQN more robust, all observations o, together with taken actions a and their rewards r, are stored in an experience memory E and the agent can sample experiences E' from the experience memory. The sampled experiences are fed to the gradient descent algorithm as a mini-batch. Thus, the agent learns on an average of the experiences in E and is likely to train on the same experiences multiple times, which can speed up the network's convergence and reduce oscillations [10].

5.3 Dropout

Overfitted neural networks have bad generalization performance [11] and to help reduce overfitting a technique called dropout was used. The idea with dropout is to temporarily remove random hidden neurons with their connections from the network, before each training iteration. This is done by, independently, for each neuron, setting its value to 0 with a probability p. For more details, see [12].

5.4 Long short term memory

The effect of changes over time is explored in this paper and is done by adding a recurrent layer to the DQN making it a Deep Recurrent Q-Network (DRQN). A regular Recurrent Neural Network struggle to remember longer sequences due to vanishing gradients, and in [13] LSTM is used to solve this problem. An LSTM is a recurrent network constructed for tasks with long-term dependencies. Instead of storing all information from previous time sample, LSTM stores information in a memory cell and modify this memory by using insert and forget gates. These gates decides if a memory cell should be kept or cleared, and during training, the network learns how to control these gates. As a result of this, both newly seen observations and observations seen a long time ago can be stored and used by the network. A sequence length of 4 is used when training the LSTM, where the first 3 observations are only used to build the internal memory state of the LSTM cells, as described in [14].

6 Implementation

In this section we go through the experiment implementation. A simulation environment was set up to model the interactions. From section 3, both the number of observations and actions are dependent on the maximum number of cars. In this paper we consider up to 4 cars. The Deep Q Network can then also be fully defined with the help of observations from section 3 and finally we go though the reward function that defines our behavior.

6.1 Simulation environment

The simulation environment is set up as an intersection described in section 3.3. The number of other cars that are observable at the same time can vary from 1-4, while their intentions can vary between an aggressive *take way*, passive *give way* or a cautious driver. The take way driver does not slow down or yield for crossing traffic in an intersection, while the give way driver will always yield for other vehicles before continuing through the intersection. The cautious driver on the other hand, will slow down for crossing traffic but not come down to a full stop. With a maximum number of other cars set to 4 all possible actions the ego vehicle can take are:

- α_1: Keep set speed.
- α_2: Stop in front of intersection.
- α_3: Keep distance to vehicle 1.
- α_4: Keep distance to vehicle 2.
- α_5: Keep distance to vehicle 3.
- α_6: Keep distance to vehicle 4.

At the start of an episode, the ego vehicle's position and velocity, the number of other vehicles and their intentions are randomly generated. The episode only end when the ego vehicle fulfills one out of three conditions: 1. Crossing the intersection and reaching the other side, 2. Colliding with another vehicle or 3. Running out of time τ_m. Each car follows the control law from eq. A.4,

trying to keep a set speed while keeping a set distance to the vehicle in front of its own lane.

All cars including the ego vehicle in these scenarios have a maximum acceleration set to $5m/s^2$ and was set after comfort and normal driving conditions.

6.2 Reward function tuning

When using a DQN, the reward function is optimally distributed around $[-1, 1]$. If the defined reward values are too large, the Q_π-values can become large and cause the gradients to grow [15]. The reward function is defined as follows:

$$r_t = \hat{r}_t + \begin{cases} 1 - \frac{\tau}{\tau_m} & \text{on success,} \\ -2 & \text{on collision} \\ -0.1 & \text{on timeout, i.e. } \tau \geq \tau_m \\ -\left(\frac{j_t^e}{j_{\max}}\right)^2 \frac{\Delta\tau}{\tau_m} & \text{on non-terminating updates} \end{cases}$$

$$\text{where } \hat{r}_t = \begin{cases} -1 & \text{if chosen } a_t \text{ is not valid} \\ 0 & \text{otherwise} \end{cases}$$

The actions $\alpha_3, \ldots, \alpha_6$ described should only be selected when a vehicle is visible and a valid target to follow. To learn when these actions are valid, the agent gets punished on invalid choices using $\hat{r}_t$. Accelerations returned by the controller for different STG can vary, which increases jerk and can make the ride uncomfortable. Therefore, the reward function also punishes the agent when acceleration jerk j_t^e is large, where τ is the elapsed time sense the episode started and τ_m is the maximum time has before a timeout.

6.3 Neural Network Setup

The DRQN structure is defined in Fig. 3. Where $\boldsymbol{h}$ are the hidden layers of the network with weights $\boldsymbol{W}$. Because the observations o_t from section 2, are used as input to the DRQN, the number of features must be fixed. With up to four other cars the input vectors $\boldsymbol{\xi}$ are as follows:

- $\xi_1 = [\ p_t^e \quad v_t^e \quad a_t^e \quad \delta^e \quad p_t^1 \quad v_t^1 \quad a_t^1 \quad \delta^1\ ]^T$
- $\xi_2 = [\ p_t^e \quad v_t^e \quad a_t^e \quad \delta^e \quad p_t^2 \quad v_t^2 \quad a_t^2 \quad \delta^2\ ]^T$

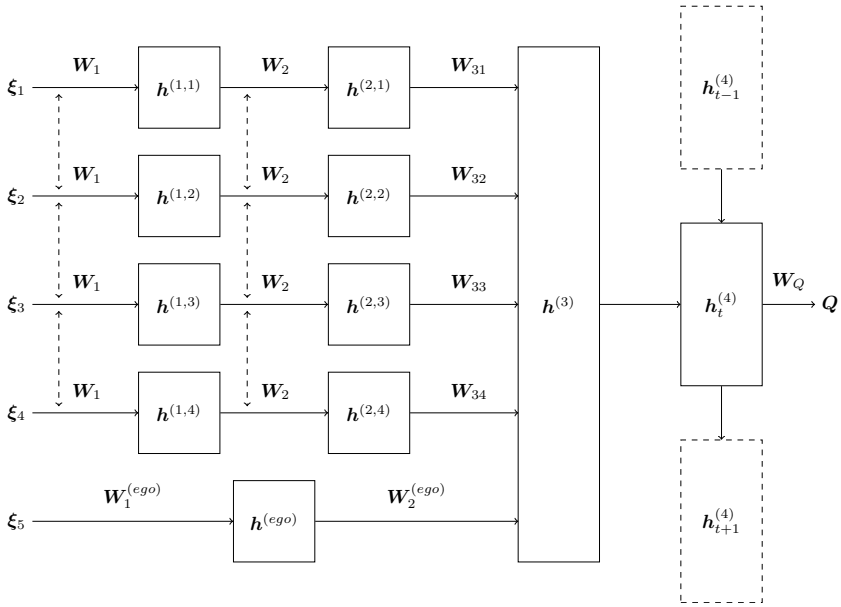


Figure 3: Deep Recurrent Q Network layout with shared weights and a LSTM

- $\xi_3 = [\ p_t^e \quad v_t^e \quad a_t^e \quad \delta^e \quad p_t^3 \quad v_t^3 \quad a_t^3 \quad \delta^3\ ]^T$
- $\xi_4 = [\ p_t^e \quad v_t^e \quad a_t^e \quad \delta^e \quad p_t^4 \quad v_t^4 \quad a_t^4 \quad \delta^4\ ]^T$
- $\xi_5 = [\ a_{t+1}^{e,1} \quad a_{t+1}^{e,2} \quad a_{t+1}^{e,3} \quad a_{t+1}^{e,4} \quad a_{t+1}^{e,5} \quad a_{t+1}^{e,6}\ ]^T$

In case a vehicle is not visible, the input vector $\boldsymbol{\xi}$ are set to -1. All vehicle features are scaled to be a value between or close to $[-1, 1]$ by using a car's sight range $p_{\max}$, maximum speed $v_{\max}$ and maximum acceleration $a_{\max}$.

The output $\boldsymbol{Q}$ should be independent of which order other vehicles was observed in the input ξ_i. In other words, whether a vehicle was fed into ξ_1 or into ξ_4, the network should optimally result in the same decision, only based on the features' values. The network is therefore structured such that input features of one car, for instance ξ_1, are used as input to a sub-network with two layers $\boldsymbol{h}^{(1,i)}$ and $\boldsymbol{h}^{(2,i)}$. Each other vehicle has a copy of this sub-network, resulting in them sharing weights ($\boldsymbol{W}_1$ and $\boldsymbol{W}_2$), as shown in Fig. 3. The first hidden layers are then given by:

$$\boldsymbol{h}^{(1,i)} = \tanh\left(\boldsymbol{W}_1\boldsymbol{\xi}_i + \boldsymbol{b}_1\right) \tag{A.9}$$

$$\boldsymbol{h}^{(2,i)} = \tanh\left(\boldsymbol{W}_2\boldsymbol{h}^{(1,i)} + \boldsymbol{b}_2\right) \tag{A.10}$$

$$\boldsymbol{h}^{(ego)} = \tanh\left(\boldsymbol{W}_1^{(ego)}\boldsymbol{\xi}_5 + \boldsymbol{b}^{(ego)}\right) \tag{A.11}$$

The output of each sub-network, $\boldsymbol{h}^{(2,i)}$ and $\boldsymbol{h}^{(ego)}$, is fed as input into a third hidden layer $\boldsymbol{h}^{(3)}$. The different sub-networks' $\boldsymbol{h}^{(2,i)}$ outputs are multiplied with different weights $\boldsymbol{W}_{31}, \ldots, \boldsymbol{W}_{34}$ in order to distinguish different cars for different follow car actions. The ego features are also fed into layer 3 with its own weights $\boldsymbol{W}_2^{(ego)}$. The neurons in layer $\boldsymbol{h}^{(3)}$ combine the inputs by adding them together:

$$\boldsymbol{h}^{(3)} = \tanh\left(\boldsymbol{W}_2^{(ego)}\,\boldsymbol{h}^{(ego)} + \sum_{i=1}^{4}\boldsymbol{W}_{3i}\,\boldsymbol{h}^{(2,i)} + \boldsymbol{b}_3\right) \tag{A.12}$$

The final layer $\boldsymbol{h}^{(4)}$ uses the LSTM, described in section 5. This layer handles the storage and usage of previous observations, making it the recurrent layer of the network.

$$\boldsymbol{h}_t^{(4)} = \text{LSTM}\left(\boldsymbol{h}^{(3)}|\boldsymbol{h}_{t-1}^{(4)}\right) \tag{A.13}$$

The output of the neural network is then the approximated $\boldsymbol{Q}$-value:

$$\boldsymbol{Q} = \boldsymbol{W}_Q\boldsymbol{h}^{(4)} + \boldsymbol{b}_4 \tag{A.14}$$

7 Results

Three metrics were selected for measurement of a training session: success rate, average episodic reward and collision to timeout ratio. With these metrics, the distribution between the three terminating states can be analyzed. Success rate is defined as the success to fail ratio averaged over the last 100 episodes, where both collisions and timeouts are considered as failures. Average episodic reward is the summed reward over a whole episode, then averaged over 100 episodes. The collision to fail ratio displays the ratio between the number of collisions and unsuccessful episodes for the last 100 episodes. From the success rate and collision to fail ratio, a final collision rate is computed, which

is the amount of episodes resulting in a collision, averaged over 100 episodes. The graphs presented are only using evaluation episodes, with a deterministic policy. Every 300 episode, the trained network is evaluated over 300 evaluation episodes.

The improvement of using Dropout and Experience replay, from Section 5, are clearly shown in Fig. 4 and 5. Studying the red curve in Fig. 4, with all methods included, the best policy had a success rate of 99%, average episodic reward 0.8 and collision to timeout ratio at 40%.

7.1 Effect of using Experience replay

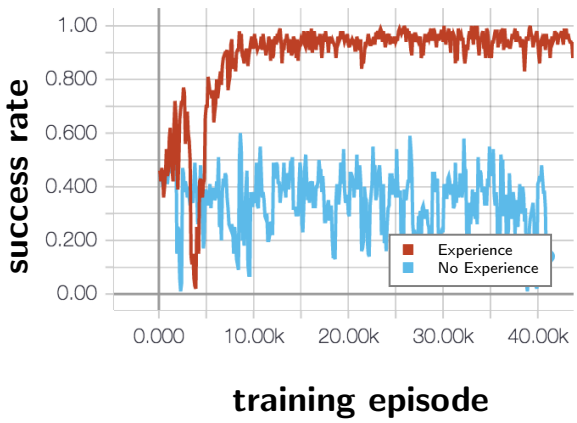


Figure 4: Success rate trend comparing using experience replay (red) and not using experience replay (blue)

Without either method the success rate does not converge to a value higher than 60%. When experience replay was not used, the highest success rate was 53%, average episodic reward -0.1 and collision to timeout ratio at 90%. Compared to not using dropout, not using experience replay has a higher lower variation on the success rate.

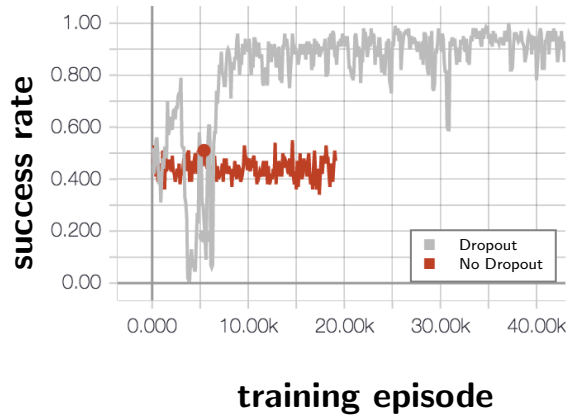


Figure 5: Success rate trend comparing using dropout (grey) and not using dropout (red)

7.2 Effect of using Dropout

In the case of not using Dropout, the best policy had a success rate of 58%, average episodic reward -0.7 and a collision to timeout ratio at 90%.

7.3 Comparing DQN and DRQN

In Fig. 6, we can see the effect off having a recurrent layer by comparing a DQN, without a LSTM layer, and with a DRQN, with LSTM. The faded colored line show actual sampled values and the thick line acts as a trend line which for DRQN converges towards a success rate of around 97.2% and a 0.85% collision rate, compared to a success rate of 87.5% and a collision rate of 1.75% for DQN.

7.4 Effect of sharing weights in the network

When introducing multiple cars in the scenario, the success rate converged considerably slower, as shown in Fig. 7. Using the network structure that share weights between cars, significantly improved how fast the network converged compared to a fully connected DRQN.

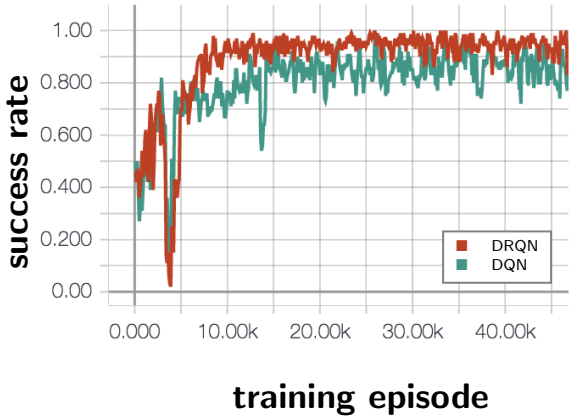


Figure 6: Graphs presenting the performance of a DRQN (red) compared to a DQN with a single observation (green), running on scenarios with cars that have different behaviors. When compared a DRQN succeeds in 3 out of 4 attempts, where a DQN fails.

8 Conclusion

In this paper, Deep Q-Learning was presented in the domain of autonomous vehicle control. The goal of the ego agent is to drive through an intersection, by adjusting longitudinal acceleration using short-term goals. Short-term goals also allowed a smoother and more human-like behavior by controlling the acceleration and comfort with a separate controller. Instead of finding a policy with a continuous control output the problem became a classification problem.

The results also show that trained policies can generalize over different types of driver behaviors. The same policy is able to respond to other vehicles' actions and handle traffic scenarios with a varied number of cars, without knowing traffic rules or the type of intersection it drives in. Multiple observations are needed in order to recognize cars' behaviors, and can be utilized by for instance a DRQN.

There was a significant performance improvement in using a DRQN instead of a DQN with a single observation. In other words, the environment for these scenarios is better modeled as a POMDP instead of an MDP and the agent needs multiple observations in order to draw enough conclusions about other

Figure 7: The figure shows that the success rate for a network with shared weights (brown line) converge faster than the fully connected network structure which do not share weights (turquoise line).

cars' behaviors. Convergence of the neural network was shown to be improved by sharing weights between the first layers to which the target car features are fed, compared to a fully connected neural network structure. The results are still limited to the tested traffic scenarios and driver behaviors, and expanding the domain beyond a simulator is a natural next step. The selected features are also limited to intersections.

The results show a success rate of around 99% for recognizing behaviors. However, the ego vehicle still collides. The ego vehicle is limited to drive comfortably, meaning that in some cases, the ego agent is not allowed to break hard enough. In a complete system, a collision avoidance procedure, which does not have comfort constraints, would need to take over the control to ensure a safe ride. A collision in this paper is defined by two areas overlapping, and in a real world implementation this does not have to mean an actual collision. Instead this could be interpreted as an intervention from a more safety critical system. This way, in the low chances a good action could not be found, the safety of the vehicles can still be guarantied.

In section 3.2, a sliding mode controller was chosen, but this can be replaced by any controller. One other option could be a Model Predictive Controller, where safer actuation can be achieved by using constraints. Also, the actions in this paper used the same controller tuning for all actions, this does not have to be the case. An action can have the same STG but only differ by the controller's tuning parameters. This way, the agent gains more flexibility while the comfort can maintain intact, possibly increasing the success rate.

References

[1] M. Bojarski *et al.*, "End to End Learning for Self-Driving Cars," 2016.

[2] M. Liebner, M. Baumann, F. Klanner, and C. Stiller, "Driver intent inference at urban intersections using the intelligent driver model," in *IEEE Intelligent Vehicles Symposium, Proceedings*, 2012.

[3] S. Lefevre, C. Laugier, and J. Ibanez-Guzman, "Evaluating risk at road intersections by detecting conflicting intentions," in *IEEE International Conference on Intelligent Robots and Systems*, 2012.

[4] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," arXiv:1610.03295, 2016.

[5] S. Brechtel, T. Gindele, and R. Dillmann, "Probabilistic decision-making under uncertainty for autonomous driving using continuous POMDPs," in *Proceedings of the 17th IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2014.

[6] J. J. Moskwa and J. K. Hedrick, "Sliding mode control of automotive engines," in *American Control Conference (ACC)*, 1989.

[7] R. Bellman, "A markovian decision process," English, *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957.

[8] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[9] J. N. Tsitsiklis and B. V. Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Transactions on Automatic Control*, vol. 42, no. 5, 1997.

[10] L. J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Journal of Machine Learning Research*, 1992.

[11] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," 2012, ISSN: 9781467394673.

[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[13] S. Hochreiter and J. Urgen Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] G. Lample and D. S. Chaplot, "Playing fps games with deep reinforcement learning," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2017.

[15] H. V. Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver, "Learning values across many orders of magnitude," in *Advances in Neural Information Processing Systems (NIPS)*, 2016.

PAPER B

Learning When to Drive in Intersections by Combining Reinforcement Learning and Model Predictive Control

Tommy Tram, Ivo Batković, Mohammad Ali, and Jonas Sjöberg

Published in 2019 IEEE Intelligent Transportation Systems Conference (ITSC),
pp. 3263–3268, Oct. 2019.
 DOI: 10.1109/ITSC.2019.8916922.

The layout has been revised.

Abstract

In this paper, we propose a decision making algorithm intended for automated vehicles that negotiate with other non-automated vehicles in crossings. The decision algorithm is separated into two parts: a high-level decision module based on reinforcement learning, and a low-level planning module based on model predictive control. Traffic is simulated with different predefined driver behaviors and intentions, and evaluate the performance of the proposed decision algorithm and benchmark it against using a sliding mode controller. The results show that the proposed decision algorithm yields faster training times and an increased performance compared to the sliding mode controller.

1 Introduction

Self driving cars is a fast advancing field with Advanced Driver-Assistance Systems becoming a requirement in modern day cars. Decision making for self drving cars can be difficult to solve with simple rule based system in complex scenarios like intersections, while human drivers have a good intuition about when to drive and how to drive comfortably. Sharing the road with other road users requires interaction, which can make rule based decision making complex [1]. Many advancements aim to bring self driving Level 4 to the market by trying to imitate human drivers [2] or predicting what other drivers in traffic are planning to do [3].

Previous research [4] showed that reinforcement learning (RL) can be used to learn a negotiation behavior between cars without vehicle to vehicle communication when driving in an intersection. The method found a policy that learned to drive through an intersection, with crossing traffic, where other vehicles have different intentions and avoided collision. The previous method could use the same algorithm, but trained on different type of intersections and still find a general policy that would get to the other side of the intersection while avoiding collision. By modeling the decision process as a partially observable Markov decision process, uncertainty in the environment or sensing can be accounted for [5] and still be safe [6].

Because the decision policy is separated from the control, high level decision

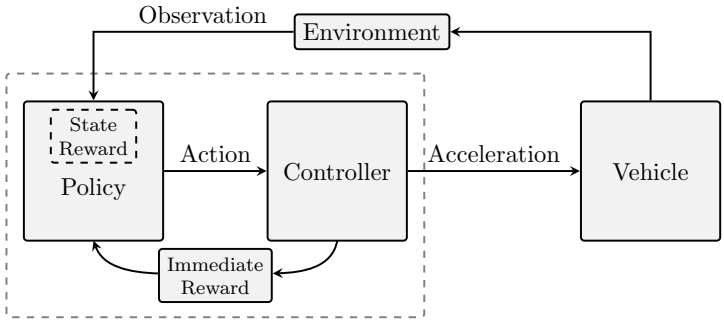


Figure 1: Representation of the decision making architecture

making can focus on the task, when to drive, while the low level control that handles the comfort of passengers in the car by generating a smooth acceleration profile. Previous work showed how this worked for intersections with a single crossing point, where Short Term Goal (STG) actions could choose one car to follow. This gives the RL policy a flexibility to choose actions that can safely transverse through the intersection by switching between different STG.

A simple controller holds well when intersection crossing points are far away from each other, but when there are several crossing points in close succession, a simple controller would have a hard time handling it. In this paper, we instead propose using MPC that can consider multiple vehicles at the same time and generate an optimal trajectory. In contrast to [7], where they prove stability and recursive feasibility using an MPC approach, and assuming that agents can cooperate, we restrict ourselves to non-cooperative scenarios. The MPC is used to plan trajectories around other vehicles using available predictions from other vehicles, and in the worst case, use the prediction as early detection whenever a dangerous situation, e.g., a collision, may appear.

Applying MPC directly to the problem, could lead to a growing complexity with the number of vehicles in the intersection, e.g., which vehicle do we yield for and which vehicle do we drive in front. Therefore, we propose to separate the problem into two parts: the first being a high-level decision maker, which structures the problem, and the second being a low level planner, which optimizes a trajectory given the traffic configuration.

For the high-level decision maker RL is used to generate decisions for how the

vehicle should drive through the intersection, and MPC is used as a low-level planner to optimize a safe trajectory. Compared to [8] where all vehicles are controlled using MPC to stay in a break-safe set based on a model of other vehicles future trajectory, this can be perceived as too conservative for a passenger. By combining RL and MPC, the decision policy will learn which action is optimal by using feedback from the MPC controller into the reward function. Since MPC uses predefined models, e.g. vehicle models and other obstacle prediction models, the performance relies on their accuracy and assumptions. To mitigate this, we use Q-learning, which is a model-free RL approach, to optimize the expected future reward based on its experience during an entire episode which is able to compensate to some extent for model errors, which is explained more in section 5.1.

This paper is structured as follows. Section 2 introduces the system architecture of our framework. The problem formulation, along with the two-layers of the decision algorithm is presented in Section 3. Section 4 present three different used agents for simulation and validation. Implementation details is presented in Section 5 and the results are shown in Section 6 followed by discussion in Section 7. Finally, conclusions and future research are presented in Section 8.

2 System

A full decision architecture system shown in Fig. 1, would include a precautionary safety layer that limits which acceleration values the system can actuate in order to stay safe. Followed by a decisions making system that makes a high level decision for when to drive in order to avoid collision and be comfortable. The policy maker later send that action to a controller that actuates the action turning it into a control signal, e.g., an acceleration request. The environment state, together with the new acceleration request, is sent though a collision avoidance system that checks if the current path has a collision risk, and mitigate if needed. With such a structure, the comfort that is experienced by the passenger is fully controlled by the low-level controller and partially affected by the decision. This paper focuses on the integration between policy and actuation, by having an MPC controller directly giving feedback to the decision maker through immediate actions. This allows the policy to know

how comfortably the controller can handle the action and give feedback sooner if the predicted outcome may be good or bad.

3 Problem formulation

The goal of the ego-vehicle is to drive along a predefined route that has one or two intersections with crossing traffic, where the intent of the other road users is unknown. Therefore, the ego-vehicle needs to assess the driving situation and drive comfortably, while avoiding collisions with any vehicle[1] that may cross. In this section, we define the underlying Partially Observable Markov Decision Process (POMDP) and present how the problem is decomposed using RL for decision making and MPC for planning and control.

3.1 Partially Observable Markov Decision Process

The decision making process is modeled as a Partially Observable Markov Decision Process (POMDP). A POMDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ an action space that is defined in section 4.1, $\mathcal{T}$ the transition function, $\mathcal{R}$ the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is defined in 5.4, Ω an observation space, $\mathcal{O}$ the probability of being in state s_t given the observation o_t, and γ the discount factor.

A POMDP is a generalization of the Markov Decision Process (MDP) [9] and therefore works in the same way in most aspects. At each time instant t, an action, $a_t \in \mathcal{A}$, is taken, which will change the environment state s_t to a new state s_{t+1}. Each transition to a state s_t with an action a_t has a reward r_t given by a reward function $\mathcal{R}$. The Key difference from a regular MDP is that the environment state s_t is not entirely observable because the intention of other vehicles are not known. In order to find the optimal solution for our problem, we need to know the future intention of other drivers. Instead we can only partially perceive the state though observations $o_t \in \Omega$.

3.2 Deep Q-Learning

In the reinforcement learning problem, an agent observes the state s_t of the environment, takes an action a_t, and receives a reward r_t at every time step t.

[1] Although our approach can be extended to other road users, for convenience of exposition we'll refer to vehicles.

Through experience, the agent learns a policy π in a way that maximizes the accumulated reward $\mathcal{R}$ in order to find the optimal policy π^*. In Q-learning, the policy is represented by a state action value function $Q(s_t, a_t)$. The optimal policy is given by the action that gives the highest Q-value.

$$\pi^*(s_t) = \arg\max_{a_t} Q^*(s_t, a_t) \tag{B.1}$$

Following the Bellman equation the optimal Q-function $Q^*(s_t, a_t)$ is given by:

$$Q^*(s_t, a_t) = \mathbb{E}[r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) | s_t, a_t] \tag{B.2}$$

Because Q-learning is a model-free algorithm and it does not make any assumptions on the environment, even though models are used to simulate the environment, this can be useful when the outcome does not match the prediction models.

4 Agents

This section explains the different agents types. MPC and sliding mode (SM) for the ego vehicle. Observed target vehicles has different intention agents based on SM agent.

4.1 MPC agent

We model the vehicle motion with states $\mathbf{x} \in \mathbb{R}^3$ and control $\mathbf{u} \in \mathbb{R}$, defined as

$$\mathbf{x} := [p^{\mathrm{e}} \quad v^{\mathrm{e}} \quad a^{\mathrm{e}}]^\top, \quad \mathbf{u} := j^{\mathrm{e}}, \tag{B.3}$$

where we denote the position along the driving path in an Frenet frame as p^{e}, the velocity as v^{e}, the acceleration as a^{e}, and the jerk as j^{e}, see Fig. 2. In addition, we assume that measurements of other vehicles are provided through an observation $\mathbf{o}$. We limit the scope of the problem to consider at most four vehicles, and define the observations as

$$\mathbf{o} := [p^1 \quad v^1 \quad p_{\mathrm{ego}}^{\mathrm{cross},1} \quad \cdots \quad p^4 \quad v^4 \quad p_{\mathrm{ego}}^{\mathrm{cross},4} \quad]^\top, \tag{B.4}$$

where we denote the position along its path as p, the velocity as v, and $p_{\mathrm{ego}}^{\mathrm{cross},j}$ for $j \in [1, 4]$, as the distance to the ego-vehicle from the intersection point, see Fig. 2.

In this paper, we assume that there exists a lateral controller that stabilizes the vehicle along the driving path. To that end, we only focus on the longitudinal control. Given the state representation, the dynamics of the vehicle is then modeled using a triple integrator with jerk as control input.

The objective of the agent is to safely track a reference, e.g. follow a path with a target speed, acceleration, and jerk profile, while driving comfortably and satisfying constraints that arise from physical limitations and other road users, e.g. not colliding in intersections with crossing vehicles. Hence, we formulate the problem as a finite horizon, constrained optimal control problem

$$
\begin{aligned}
J = \min_{\bar{\mathbf{x}},\bar{\mathbf{u}}} \sum_{k=0}^{N-1} & \begin{bmatrix} \bar{\mathbf{x}}_k - \mathbf{r}_k^{\mathbf{x}} \\ \bar{\mathbf{u}}_k - \mathbf{r}_k^{\mathbf{u}} \end{bmatrix}^\top \begin{bmatrix} Q & S^\top \\ S & R \end{bmatrix} \begin{bmatrix} \bar{\mathbf{x}}_k - \mathbf{r}_k^{\mathbf{x}} \\ \bar{\mathbf{u}}_k - \mathbf{r}_k^{\mathbf{u}} \end{bmatrix} && \text{(B.5a)} \\
& + \begin{bmatrix} \bar{\mathbf{x}}_N - \mathbf{r}_N^{\mathbf{x}} \end{bmatrix}^\top P \begin{bmatrix} \bar{\mathbf{x}}_N - \mathbf{r}_N^{\mathbf{x}} \end{bmatrix} \\
\text{s.t.} \quad & \bar{\mathbf{x}}_0 = \hat{\mathbf{x}}_0, && \text{(B.5b)} \\
& \bar{\mathbf{x}}_{k+1} = A\bar{\mathbf{x}}_k + B\bar{\mathbf{u}}_k, && \text{(B.5c)} \\
& h(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k, \bar{\mathbf{o}}_k, a_k) \leq 0, && \text{(B.5d)}
\end{aligned}
$$

where k is the prediction time index, N is the prediction horizon, Q, R, and S are the stage costs, P is the terminal cost, $\bar{\mathbf{x}}_k$ and $\bar{\mathbf{u}}_k$ are the predicted state and control inputs, $\mathbf{r}_k^{\mathbf{x}}$ and $\mathbf{r}_k^{\mathbf{u}}$ are the state and control input references, $\bar{\mathbf{o}}_k$ denotes the predicted state of vehicles in the environment which need to be avoided, and a is the action from the high-level decision maker. Constraint (B.5b) enforces that the prediction starts at the current state estimate $\hat{\mathbf{x}}_0$, (B.5c) enforces the system dynamics, and (B.5d) enforces constraints on the states, control inputs, and obstacle avoidance.

The reference points, $\mathbf{r}_k^{\mathbf{x}}$, $\mathbf{r}_k^{\mathbf{u}}$ are assumed to be set-points of a constant velocity trajectory, e.g. following the legal speed-limit on the road. Therefore, we set the velocity reference according to the driving limit, and the acceleration and jerk to zero.

Obstacle prediction

In order for the vehicle planner in (B.5) to be able to properly avoid collisions, it is necessary to provide information about the surrounding vehicles in the environment. Therefore, similarly to [10], we assume that a sensor system

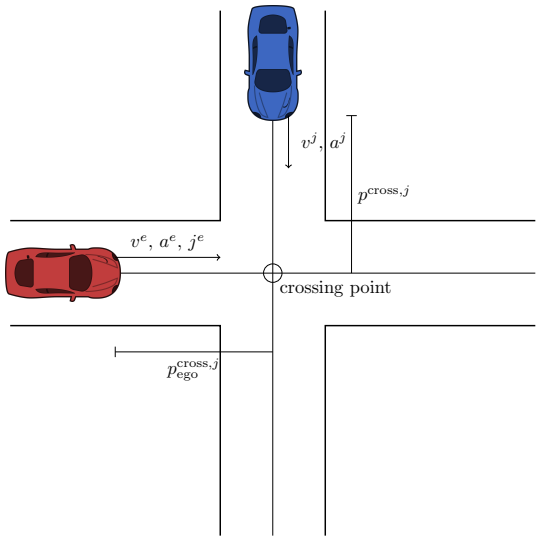


Figure 2: Observations of a scenario

provides information about the environment, and that there exists a prediction layer which generates future motions of other vehicles in the environment. The accuracy of the prediction layer will heavily affect the performance of the planner, hence, it is necessary to have computationally inexpensive and accurate prediction methods.

In this paper, for simplicity the future motion of other agents is estimated by a constant velocity prediction model. The motion is predicted at every time instant for prediction times $k \in [0, N]$, and is used to form the collision avoidance constraints, which we describe in the next section. Even though more accurate prediction methods do exist, e.g. [11], [12], we use this simple model to show the potential of the overall framework.

Collision avoidance

We denote a vehicle j with the following notation $\mathbf{x}^j := [p^j\, v^j\, a^j]^\top$, and an associated crossing point at position $p^{\text{cross},j}$ in its own vehicle frame, which translated into the ego-vehicle frame is denoted as $p_{\text{ego}}^{\text{cross},j}$. With a predefined road topology, we assume that the vehicles will travel along the assigned paths,

and that collisions may only occur at the crossing points $p^{\text{cross},j}$ between an obstacle and the ego vehicle. Hence, for collision avoidance, we use the predictions of the future obstacle states $\bar{\mathbf{x}}_k^j$ for times $k \in [0, N]$, provided by a prediction layer outside of the MPC framework. Given the obstacle measurements, the prediction layer will generate future states throughout the prediction horizon. With this information, it is possible to identify the time slots when an obstacle will enter the intersection.

Whenever an obstacle j is predicted to be within a threshold of $p^{\text{cross},j}$, e.g. the width of the intersecting area, the ego vehicle faces a constraint of the following form

$$\bar{p}_k^{\text{e}} \geq p_{\text{ego}}^{\text{cross},j} + \Delta, \quad \underline{p}_k^{\text{e}} \leq p_{\text{ego}}^{\text{cross},j} - \Delta,$$

where Δ ensures sufficient padding from the crossing point that does not cause a collision. The choice of Δ must be at least such that p_k together with the dimensions of the ego-vehicle does not overlap with the intersecting area.

Take way and give way constraint

Since the constraints from the surrounding obstacles become non-convex, we rely on the high-level policy maker to decide through action a how to construct the constraint (B.5d) for Problem (B.5). The take-way action implies that the ego-vehicle drives first through the intersection, i.e., it needs to pass the intersection before all other obstacles. This implies that for any vehicle j that reaches the intersection during prediction times $k \in [0, N]$, the generated constraint needs to lower bound the state p_k according to

$$\max_j p^{\text{cross},j} + \Delta \leq p_k^{\text{e}}. \tag{B.6}$$

Similarly, if the action is to give way, then the position needs to be upper bounded by the closest intersection point so that

$$p_k^{\text{e}} \leq \min_j p_{\text{ego}}^{\text{cross},j} - \Delta, \tag{B.7}$$

for all times k that the obstacle is predicted to be in the intersection.

Following an obstacle

For any action a that results in the following of an obstacle j, the ego-vehicle position is upper bounded by $p_k^{\text{e}} \leq p_{\text{ego}}^{\text{cross,j}}$. We construct constraints for obstacles $i \neq j$ according to

- if $p^{\text{cross,i}} < p^{\text{cross,j}}$ then $p^{\text{cross,i}} + \Delta \leq p_k^{\text{e}}$, which implies that the ego-vehicle should drive ahead of all obstacles i that are approaching the intersection;
- if $p^{\text{cross,i}} > p^{\text{cross,j}}$ then $p_k^{\text{e}} \leq p^{\text{cross,i}} - \Delta$, which implies that the ego-vehicle should wait to pass obstacle j and other obstacles i;
- if $p^{\text{cross,i}} = p^{\text{cross},j}$ then the constraints generated for obstacle i becomes an upper or lower bound depending on if obstacle i is ahead or behind the obstacle j into the intersection.

4.2 Sliding mode agent

To benchmark the performance of using MPC, a SM controller that was used in [4] is introduced.

$$a_{sm}^{e} = \frac{1}{c_2}(-c_1 x_2 + \mu sign(\sigma(x_1, x_2))), \tag{B.8a}$$

$$\text{where} \begin{cases} x_1 = p^t - p^e, \\ x_2 = v^t - v^e, \end{cases} \tag{B.8b}$$

$$\sigma = c_1 x_1 + c_2 x_2, \tag{B.8c}$$

$$a_{\text{p}}^{e} = K(v_{\max} - v^e), \tag{B.8d}$$

$$a^e = \min(a_{\text{sm}}^{e}, a_{\text{p}}^{e}). \tag{B.8e}$$

The SM controller aims to keep a minimum distance to a target car with a velocity of v^e, by controlling the acceleration a_{sm}^{e}. c_1, c_2, and μ are tuning parameters to control the comfort of the controller. In case there is no target car, the controller maintains a target velocity $v_{\max}$ with a regular p-controller from (B.8d) with a proportional constant K. The final acceleration is given by (B.8e). For more details about the SM agent see [4].

4.3 Intention agents

There are three different intention agents for crossing traffic with predetermined intentions, three velocity profiles are shown as an example in Fig. 3. All agents were implemented with a SM controller with different target values. The take way intention does not yield for the crossing traffic and simply aim to keep its

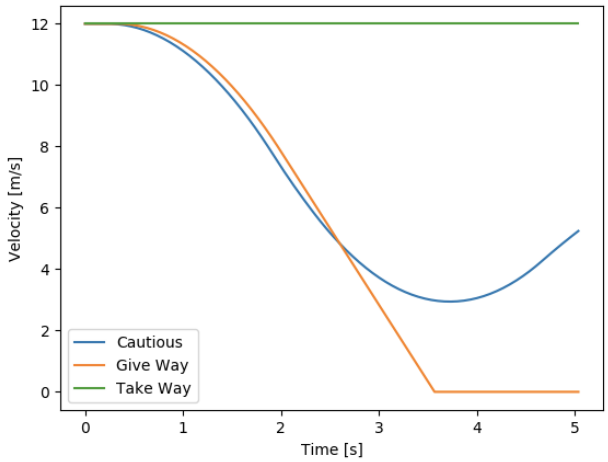


Figure 3: Example of how velocity profile of the different intention agents can look like. All agents has the same starting velocity of 12m/s and are approaching the same intersection

target reference speed. Give way intention however, slow down to a complete stop at the start of the intersection until crossing traffic has passed before continuing through. The third intention is cautious, slowing down but not to a full stop. This makes it difficult for a constant velocity or acceleration model to predict what other agents will do.

5 Implementation

5.1 Deep Q-Network

The deep Q-network is structured as a three layer neural network with shared weights and a Long Short-Term Memory based on previous work [4] and shown in Fig. 4. The input features ξ_n are composed of observations o_t, introduced in section 3.1 and shown in Fig. 2, with up to four observed vehicles:

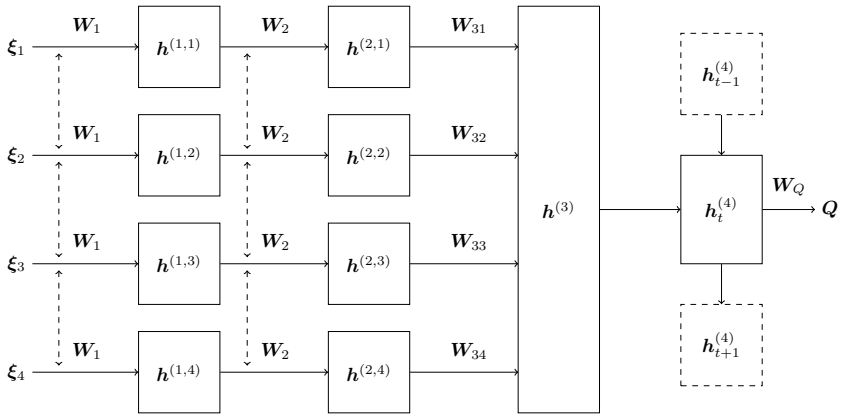


Figure 4: Representation of the network structure

$$\begin{aligned}
\xi_1 &= [\ p_t^e \quad v_t^e \quad a_t^e \quad \delta^e \quad p_t^1 \quad v_t^1 \quad a_t^1 \quad \delta^1\ ]^T \\
\xi_2 &= [\ p_t^e \quad v_t^e \quad a_t^e \quad \delta^e \quad p_t^2 \quad v_t^2 \quad a_t^2 \quad \delta^2\ ]^T \\
\xi_3 &= [\ p_t^e \quad v_t^e \quad a_t^e \quad \delta^e \quad p_t^3 \quad v_t^3 \quad a_t^3 \quad \delta^3\ ]^T \\
\xi_4 &= [\ p_t^e \quad v_t^e \quad a_t^e \quad \delta^e \quad p_t^4 \quad v_t^4 \quad a_t^4 \quad \delta^4\ ]^T
\end{aligned} \tag{B.9}$$

Normalization of the input features is done by scaling the features down to values between $[-1, 1]$ using the maximum speed $v_{\max}$, maximum acceleration $a_{\max}$ and a car's sight range $p_{\max}$. Empty observation of other vehicles $[p_t^n \quad v_t^n \quad a_t^n \quad \delta^n]$ has a default value of $-\mathbf{1}$. The input vectors are sent though two hidden layers $\boldsymbol{h}^{(1,i)}$ and $\boldsymbol{h}^{(2,i)}$ with shared weights $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ respectively

$$\boldsymbol{h}^{(1,i)} = \tanh\left(\boldsymbol{W}_1 \boldsymbol{\xi}_i + \boldsymbol{b}_1\right) \tag{B.10}$$

$$\boldsymbol{h}^{(2,i)} = \tanh\left(\boldsymbol{W}_2 \boldsymbol{h}^{(1,i)} + \boldsymbol{b}_2\right). \tag{B.11}$$

A similar study for lane changes on a highway confirmed the importance of having equals weights for inputs that describe the state of interchangeable objects [13]. The output of each sub-network is then sent though a fully connecting layer

$$\boldsymbol{h}^{(3)} = \tanh\left(\sum_{i=1}^{4} \boldsymbol{W}_{3i}\, \boldsymbol{h}^{(2,i)} + \boldsymbol{b}_3\right). \tag{B.12}$$

That is then connected to a Long Short-Term Memory (LSTM) [14] that can store and use previous features

$$\boldsymbol{h}_t^{(4)} = \text{LSTM}\left(\boldsymbol{h}^{(3)} | \boldsymbol{h}_{t-1}^{(4)}\right). \tag{B.13}$$

The approximated Q-value is then

$$\boldsymbol{Q}_{approx} = \boldsymbol{W}_Q \boldsymbol{h}^{(4)} + \boldsymbol{b}_4 \tag{B.14}$$

The Q-value is then masked using Q-masking, explain in section 5.2

$$\boldsymbol{Q} = \boldsymbol{Q}_{approx} \boldsymbol{Q}_{mask} \tag{B.15}$$

the optimal policy π^* is then given by taking the action that gives the highest Q-value

$$\pi^*(s_t) = \arg\max_{a_t} Q^*(s_t, a_t) \tag{B.16}$$

5.2 Q-masking

Q-masking [15] helps the learning process by reducing the actions space by disabling actions the agent does not need to explore. If there are less than N cars, it would then be meaningless to choose to follow a car that does not exist. Which motivates masking off cars that does not exist. In previous work [4], a high negative reward was given when an action to follow a car that did not exist was chosen, while the algorithm continued with a default action take way. The agent quickly learned to not choose cars that did not exist, but with Q-masking, the agent does not even have to explore these options. For other details about the training see [4].

5.3 Simulation environment

All agents are spawned with random initial speed $v_0 \in [10, 30]$m/s, position $p_0^i \in [10, 55]$m and intention. The cars dimensions are 2 m wide and 4 m long. The ego car operates within comfort bounds and therefore has a limited maximum acceleration and deceleration of 5 m/s^2. Two main types of crossing were investigated. One and two crossing points as shown in Fig. 5, where the distance between crossing points d_{cross} vary between $[4, 8, 12, 25, 30, 40]m$ with different scenarios.

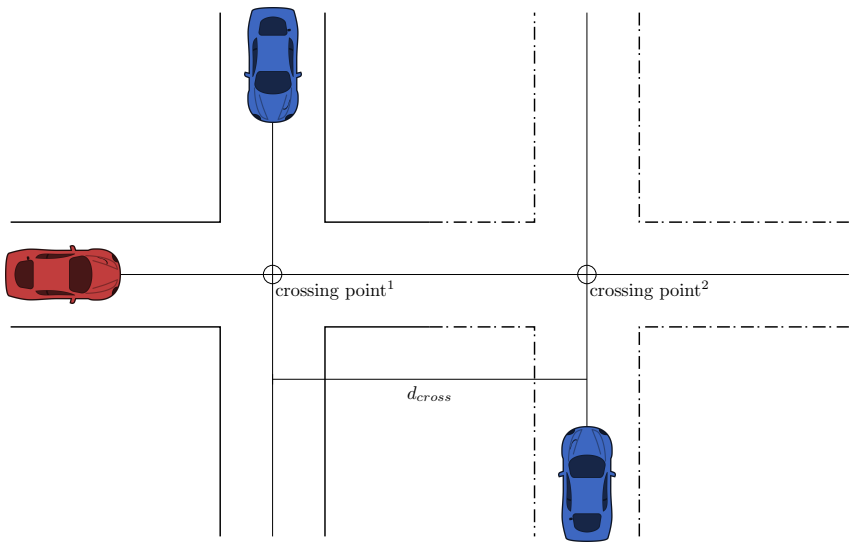


Figure 5: Illustration of a intersection scenario, where the solid line is a single crossing and together with the dashed line creates a double crossing.

The MPC agent was discretized at 30Hz, with a prediction horizon of $N = 100$ and cost tuning of

$$Q = \text{blockdiag}(0.0, 1.0, 1.0),\ R = 1,\ S = \mathbf{0}. \tag{B.17}$$

5.4 Reward function tuning

There are three states that terminates an episode; success, failure, and timeout. Success is when the ego agent reaches the of the road defined by the scenario. Failure is when the frame of the ego agent overlaps with another road users frame, e.g., in a collision, this frame can be the size of the vehicle or a safety boundary around a vehicle. The final terminating state is timeout and that is simply when the agent cant reach the two previous terminating states before the timeout time τ_m. According to [16], the Q_π values and gradient can grow to be very large if the total reward values are too large. All rewards are therefore scaled with the episode timeout time τ_m, which is set to $25s$, to keep

the total reward $r_t \in [-2, 1]$. The reward function is defined as follows:

$$r_t = \begin{cases} 1 & \text{on success,} \\ -1 & \text{on failure,} \\ 0.5 & \text{on timeout, i.e. } \tau \geq \tau_m, \\ f(p_{\text{crash}}, p_{\text{comf}}) & \text{on non-terminating updates,} \end{cases}$$

where $f(p_{\text{crash}}, p_{\text{comf}})$ consists of

$$f(p_{\text{crash}}, p_{\text{comf}}) = \alpha p_{\text{crash}} \frac{\tau_m}{\tau - t_{pred}} + \beta p_{\text{comf}} \frac{\tau_m}{\tau}, \tag{B.18}$$

with $\alpha \in [0, 1]$, $\beta \in [0, 1]$ being weight parameters, and $\alpha + \beta = 1$. The first term in the function corresponds to a feasibility check of Problem (B.5), which to a large extent depends on the validity of the accuracy of the prediction layer. The high-level decision from the policy-maker affects how the constraints are constructed, and may turn the control problem infeasible, e.g. if the decided action is to take way, while not being able to pass the intersection before all other obstacles. Therefore, whenever the MPC problem becomes infeasible we set $p_{\text{crash}} = 1$ to indicate that the selected action most likely will result in a collision with the surrounding environment.

The second term p_{comf} relates to the comfort of the planned trajectory, which is estimated by computing and weighting the acceleration and jerk profiles as

$$p_{\text{comf}} = \frac{1}{\sigma N} \left(\sum_{k=0}^{N-1} \bar{a}_k^2 Q^a + \bar{j}_k^2 R^j + a_N^2 Q^a \right),$$

where $\bar{a}$, and $\bar{j}$ are the acceleration and jerk components of the state and control input respectively, Q^a and R^j are the corresponding weights, and σ is a normalizing factor which ensures that $p_{\text{comf}} \in [0, 1]$. For the simulation we used $Q^a = 1$ and $R^j = 1$.

The timeout reward 0.5 was set to be higher than the average accumulated reward from p_{comf}, so that the total accumulated reward would be positive in case of timeouts. Because p_{crash} usually only triggers close to a potential collision, that is why t_{pred} is set to the first time a crash prediction is triggered. This will scale the negative reward higher in collision episodes.

Table 1: Average success rates and collision to timeout rates.

Controller	Success Rate		Timeout Ratio	
	Single	Double	Single	Double
SM	96.1%	90.9%	72%	93%
MPC	97.3%	95.2%	45%	76%

6 Results

For evaluation we compared the success rate of the decision-policy together with a collision to timeout ratio (CTR). The success rate is defined as the number of times the agent is able cross the intersections without colliding with other obstacles, or exceeding the time limit to cross. Since we define a time-out to be a failure, we use the CTR to separate potential collisions with the agent being too conservative.

Fig. 6 shows a comparison in success rate between the proposed MPC architecture and the previous SM agent for scenarios with only one crossing. In this scenario, the MPC agent converges after 10^4 training episodes, while the previous SM agent converges after $4 \cdot 10^4$ training episodes. In addition, comparing the CTR metric, Fig. 7 shows that the MPC agent has 0.45 CTR while the SM agent has 0.72 CTR. Evidently, it is visible that the MPC is able to leverage future information into its planning horizon in order to achieve faster training, and also avoiding collisions as a result.

We evaluate the performance of the MPC and SM agents for the more difficult double intersection problem, where we vary the distance between the intersection points. Table 1 shows the performance of the MPC and SM agent for both the single and double scenarios. The performance drops for both agents for the double crossing scenario. However, it is visible that the MPC agent suffers less performance degradation compared to the SM agent. The CTR however more than doubles for the MPC agent for the double crossing, while the already high CTR rate for the SM agent increases above rates of 0.9.

7 Discussion

The benefit of being able to use a prediction horizon for the MPC is shown to mostly impact the training time for the traffic scenarios compared to the

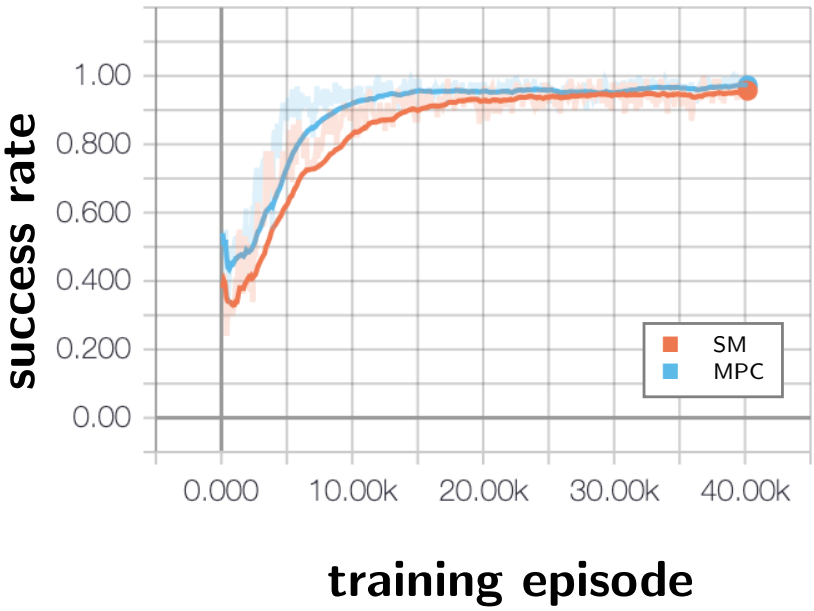


Figure 6: Average MPC and SM success rate for a single corssing after evaluating the policy 300 episodes.

SM agent. This allows the RL decision-policy to get feedback early in the training process to see whether an action most likely will lead to a collisions. In addition, the lower CTR also implies that the use of a prediction horizon also makes the decision-policy more conservative, since it rather times out than risk collisions.

It is important to note little effort was put into tuning the MPC agent, and that we used very primitive prediction methods that do not hold very well in crossing scenarios, e.g. the simulated agents did not keep constant speed profiles while approaching the intersections. However, under these circumstances, the decision algorithm still managed to obtain a success rate above 95% for the double crossings.

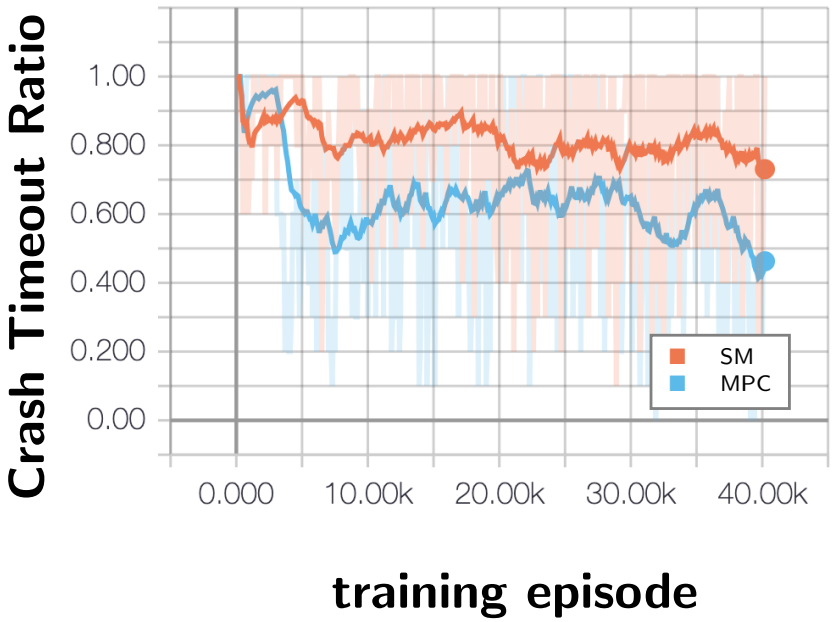


Figure 7: Average MPC and SM crash to timeout ratio for a single crossing after evaluating the policy in 300 episodes. A CTR of 0 means that all failures are timeouts, while a CTR of 1 means that all failures are collisions.

8 Conclusion

In this paper, we proposed a decision making algorithm for intersections which consists of two components: a high-level decision maker that uses Deep Q-learning to generate decisions for how the vehicle should drive through the intersection, and a low-level planner that uses MPC to optimize safe trajectories. We tested the framework in a traffic simulation with randomized intent of other road users for both single and double crossings. Results showed that the proposed MPC agent outperforms the previous SM agent by almost 5% in scenarios with double crossings and in cases of failure, more often timeout than colliding.

References

[1] M. Liebner, M. Baumann, F. Klanner, and C. Stiller, "Driver intent inference at urban intersections using the intelligent driver model," in *IEEE Intelligent Vehicles Symposium, Proceedings*, 2012.

[2] M. Bansal, A. Krizhevsky, and A. Ogale, "ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst," 2018.

[3] A. Zyner, S. Worrall, J. Ward, and E. Nebot, "Long short term memory for driver intent prediction," in *IEEE Intelligent Vehicles Symposium, Proceedings*, 2017.

[4] 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.

[5] S. Brechtel, T. Gindele, and R. Dillmann, "Probabilistic decision-making under uncertainty for autonomous driving using continuous POMDPs," in *Proceedings of the 17th IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2014.

[6] 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 Aritifical Intelligence (UAI)*, 2018.

[7] R. Hult, M. Zanon, S. Gros, and P. Falcone, "Optimal coordination of automated vehicles at intersections: Theory and experiments," *IEEE Transactions on Control Systems Technology*, pp. 1–16, 2018.

[8] X. Qian, J. Gregoire, A. de La Fortelle, and F. Moutarde, "Decentralized model predictive control for smooth coordination of automated vehicles at intersection," in *European Control Conference (ECC)*, Jul. 2015.

[9] R. Bellman, "A markovian decision process," English, *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957.

[10] I. Batkovic, M. Zanon, M. Ali, and P. Falcone, "Real-time constrained trajectory planning and vehicle control for proactive autonomous driving with road users," in *European Control Conference (ECC)*, 2019.

[11] S. Lefèvre, D. Vasquez, and C. Laugier, "A survey on motion prediction and risk assessment for intelligent vehicles," *ROBOMECH journal*, vol. 1, no. 1, p. 1, 2014.

[12] I. Batkovic, M. Zanon, N. Lubbe, and P. Falcone, "A computationally efficient model for pedestrian motion prediction," in *European Control Conference (ECC)*, Jun. 2018.

[13] C. J. Hoel, K. Wolff, and L. Laine, "Automated speed and lane change decision making using deep reinforcement learning," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2018.

[14] S. Hochreiter and J. Urgen Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] 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.

[16] H. V. Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver, "Learning values across many orders of magnitude," in *Advances in Neural Information Processing Systems (NIPS)*, 2016.

PAPER C

Reinforcement Learning with Uncertainty Estimation for Tactical Decision-Making in Intersections

Carl-Johan Hoel, Tommy Tram, and Jonas Sjöberg

Published in 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC),
pp. 1-7, Sep. 2020.
 DOI: 10.1109/ITSC45102.2020.9294407.

The layout has been revised.

Abstract

This paper investigates how a Bayesian reinforcement learning method can be used to create a tactical decision-making agent for autonomous driving in an intersection scenario, where the agent can estimate the confidence of its decisions. An ensemble of neural networks, with additional randomized prior functions (RPF), are trained by using a bootstrapped experience replay memory. The coefficient of variation in the estimated Q-values of the ensemble members is used to approximate the uncertainty, and a criterion that determines if the agent is sufficiently confident to make a particular decision is introduced. The performance of the ensemble RPF method is evaluated in an intersection scenario and compared to a standard Deep Q-Network method, which does not estimate the uncertainty. It is shown that the trained ensemble RPF agent can detect cases with high uncertainty, both in situations that are far from the training distribution, and in situations that seldom occur within the training distribution. This work demonstrates one possible application of such a confidence estimate, by using this information to choose safe actions in unknown situations, which removes all collisions from within the training distribution, and most collisions outside of the distribution.

1 Introduction

To make safe, efficient, and comfortable decisions in intersections is one of the challenges of autonomous driving. A decision-making agent needs to handle a diverse set of intersection types and layouts, interact with other traffic participants, and consider uncertainty in sensor information. The fact that around 40% of all traffic accidents during manual driving occur in intersections indicates that decision-making in intersections is a complex task [1]. To manually predict all situations that can occur and tailor a suitable behavior is not feasible. Therefore, a data-driven approach that can learn to make decisions from experience is a compelling approach. A desired property of such a machine learning approach is that it should also be able to indicate

how confident the resulting agent is about a particular decision.

Reinforcement learning (RL) provides a general approach to solve decision-making problems [2], and could potentially scale to all types of driving situations. Promising results have been achieved in simulation by applying a Deep Q-Network (DQN) agent to intersection scenarios [3], [4], and highway driving [5], [6], or a policy gradient method to a lane merging situation [7]. Some studies have trained an RL agent in a simulated environment and then deployed the agent in a real vehicle [8], [9], and for a limited case, trained the agent directly in a real vehicle [10].

Generally, a fundamental problem with the RL methods in previous work is that the trained agents do not provide any confidence measure of their decisions. For example, if an agent that was trained for a highway driving scenario would be exposed to an intersection situation, it would still output a decision, although it would likely not be a good one. A less extreme example involves an agent that has been trained in an intersection scenario with nominal traffic, and then faces a speeding driver. McAllister et al. further discuss the importance of estimating the uncertainty of decisions in autonomous driving [11].

A common way of estimating uncertainty is through Bayesian probability theory [12]. Bayesian deep learning has previously been used to estimate uncertainty in autonomous driving for image segmentation [10] and end-to-end learning [13]. Dearden et al. introduced Bayesian approaches to RL that balances the trade off between exploration and exploitation [14]. In recent work, this approach has been extended to deep RL, by using an ensemble of neural networks [15]. However, these studies focus on creating an efficient exploration method for RL, and do not provide a confidence measure for the agents' decisions.

This paper investigates an RL method that can estimate the uncertainty of the resulting agent's decisions, applied to decision-making in an intersection scenario. The RL method uses an ensemble of neural networks with randomized prior functions that are trained on a bootstrapped experience replay memory, which gives a distribution of estimated Q-values (Sect. 2). The distribution of Q-values is then used to estimate the uncertainty of the recommended action, and a criterion that determines the confidence level of the agent's decision is introduced (Sect. 2.3). The method is used to train a decision-making agent in different intersection scenarios (Sect. 3), in which the results show that the

introduced method outperforms a DQN agent within the training distribution. The results also show that the ensemble method can detect situations that were not present in the training process, and thereby choose safe fallback actions in such situations (Sect. 4). Further characteristics of the introduced method is discussed in Sect. 5. This work is an extension to a recent paper, where we introduced the mentioned method, but applied to a highway driving scenario [16].

2 Approach

This section gives a brief introduction to RL, describes how the uncertainty of an action can be estimated by an ensemble method, and introduces a measure of confidence for different actions. Further details on how this approach was applied to driving in an intersection scenario follows in Sect. 3.

2.1 Reinforcement learning

Reinforcement learning is a branch of machine learning, where an agents explores an environment and tries to learn a policy $\pi(s)$ that maximizes the future expected return, based on the agent's experiences [2]. The policy determines which action a to take in a given state s. The state of the environment will then transitions to a new state s' and the agent receives a reward r. A Markov Decision Process (MDP) is often used to model the reinforcement learning problem. An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, T is a state transition model, R is a reward model, and γ is a discount factor. At each time step t, the agent tries to maximize the future discounted return

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{C.1}$$

In a value-based branch of RL called Q-learning [17], the objective of the agent is to learn the optimal state-action value function $Q^*(s, a)$. This function is defined as the expected return when the agent takes action a from state s and then follow the optimal policy π^*, i.e.,

$$Q^*(s, a) = \max_{\pi} \mathbb{E}\left[R_t | s_t = s, a_t = a, \pi\right]. \tag{C.2}$$

The Q-function can be estimated by a neural network with weights θ, i.e., $Q(s,a) \approx Q(s,a;\theta)$. The weights are optimized by minimizing the loss function

$$L(\theta) = \mathbb{E}_M\Big[(r + \gamma \max_{a'} Q(s',a';\theta^-) - Q(s,a;\theta))^2\Big], \tag{C.3}$$

which is derived from the Bellman equation. The loss is obtained from a mini-batch M of training samples, and θ^- represents the weights of a target network that is updated regularly. More details on the DQN algorithm are presented by Mnih et al. [18].

2.2 Bayesian reinforcement learning

One limitation of the DQN algorithm is that only the maximum likelihood estimate of the Q-values is returned. The risk of taking a particular action can be approximated as the variance in the estimated Q-value [19]. One approach to obtain a variance estimation is through statistical bootstrapping [20], which has been applied to the DQN algorithm [21]. The basic idea is to train an ensemble of neural network on different subsets of the available replay memory. The ensemble will then provide a distribution of Q-values, which can be used to estimate the variance. Osband et al. extended the ensemble method by adding a randomized prior function (RPF) to each ensemble member, which gives a better Bayesian posterior [15]. The Q-values of each ensemble member k is then calculated as the sum of two neural networks, f and p, with equal architecture, i.e.,

$$Q_k(s,a) = f(s,a;\theta_k) + \beta p(s,a;\hat{\theta}_k). \tag{C.4}$$

Here, the weights θ_k of network f are trainable, and the weights $\hat{\theta}_k$ of the prior network p are fixed to the randomly initialized values. A parameter β scales the importance of the networks. With the two networks, the loss function in Eq. C.3 becomes

$$\begin{aligned} L(\theta_k) = \mathbb{E}_M\Big[&(r + \gamma \max_{a'} (f_{\theta_k^-} + \beta p_{\hat{\theta}_k})(s',a') \\ &- (f_{\theta_k} + \beta p_{\hat{\theta}_k})(s,a))^2\Big]. \end{aligned} \tag{C.5}$$

Algorithm 3 outlines the complete ensemble RPF method, which was used in this work. An ensemble of K trainable and prior neural networks are

Algorithm 3 Ensemble RPF training process

for $k \leftarrow 1$ to K **do**
    Initialize θ_k and $\hat{\theta}_k$ randomly
    $m_k \leftarrow \{\}$
$i \leftarrow 0$
while networks not converged **do**
    $s_i \leftarrow$ initial random state
    $\nu \sim \mathcal{U}\{1, K\}$
    while episode not finished **do**
        $a_i \leftarrow \text{argmax}_a \, Q_\nu(s_i, a)$
        $s_{i+1}, r_i \leftarrow \textsc{StepEnvironment}(s_i, a_i)$
        for $k \leftarrow 1$ to K **do**
            if $p \sim \mathcal{U}(0, 1) < p_{\text{add}}$ **then**
                $m_k \leftarrow m_k \cup \{(s_i, a_i, r_i, s_{i+1})\}$
            $M \leftarrow$ sample mini-batch from m_k
            update θ_k with SGD and loss $L(\theta_k)$
        $i \leftarrow i + 1$

first initialized randomly. Each ensemble member is also assigned a separate experience replay memory buffer m_k (although in a practical implementation, the replay memory can be designed in such a way that it uses negligible more memory than a shared buffer). For each new training episode, a uniformly sampled ensemble member, $\nu \sim \mathcal{U}\{1, K\}$, is used to greedily select the action with the highest Q-value. This procedure handles the exploration vs. exploitation trade-off and corresponds to a form of approximate Thompson sampling. Each new experience $e = (s_i, a_i, r_i, s_{i+1})$ is then added to the separate replay buffers m_k with probability p_{add}. Finally, the trainable weights of each ensemble member are updated by uniformly sample a mini-batch M of experiences and using stochastic gradient descent (SGD) to backpropagate the loss of Eq. C.5.

2.3 Confidence criterion

The agent's uncertainty in choosing different actions can be defined as the coefficient of variation[1] $c_{\text{v}}(s, a)$ of the Q-values of the ensemble members. In

[1] Ratio of the standard deviation to the mean.

previous work, we introduced a confidence criterion that disqualifies actions with $c_{\mathrm{v}}(s,a) > c_{\mathrm{v}}^{\mathrm{safe}}$, where c_{safe} is a hard threshold [16]. The value of the threshold should be set so that (s,a) combinations that are contained in the training distribution are accepted, and those which are not will be rejected. This value can be determined by observing values of c_{v} in testing episodes within the training distribution, see Sect. 4.1 for further details.

When the agent is fully trained (i.e., not during the training phase), the policy chooses actions by maximizing the mean of the Q-values of the ensemble members, with the restriction $c_{\mathrm{v}}(s,a) < c_{\mathrm{v}}^{\mathrm{safe}}$, i.e.,

$$\begin{aligned} &\underset{a}{\operatorname{argmax}} \frac{1}{K} \sum_{k=1}^{K} Q_k(s,a), \\ &\text{s.t.} \quad c_{\mathrm{v}}(s,a) < c_{\mathrm{v}}^{\mathrm{safe}}. \end{aligned} \tag{C.6}$$

In a situation where no possible action fulfills the confidence criterion, a fallback action a_{safe} is chosen.

3 Implementation

The ensemble RPF method, which can obtain an uncertainty estimation of different actions, is tested on different intersection scenarios. In this work, the uncertainty information is used to reject unsafe actions and reduce the number of collisions. This section describes how the simulation of the scenarios is set up, how the decision-making problem is formulated as an MDP, the architecture of the neural networks, and the details on how the training is performed.

3.1 Simulation setup

The simulated environment consists of different intersection scenarios, and is based on previous work [22]. For completeness, an overview is presented here. Each episode starts by randomly selecting a single or bi-directional intersection, shown in Fig. 1, and placing the ego vehicle to the left with a random distance $p_{\mathrm{e}}^{c,j}$ to the intersection and a speed of 10 m/s. A random number N of other vehicles are positioned along the top and bottom roads with a random distance $p_o^{c,j}$ to the intersection, and a random desired speed v_{d}^{j}. The other vehicles follow the Intelligent Driver Model (IDM) [23], with a set time gap of $t_{\mathrm{d}}^{j} = 1$ s.

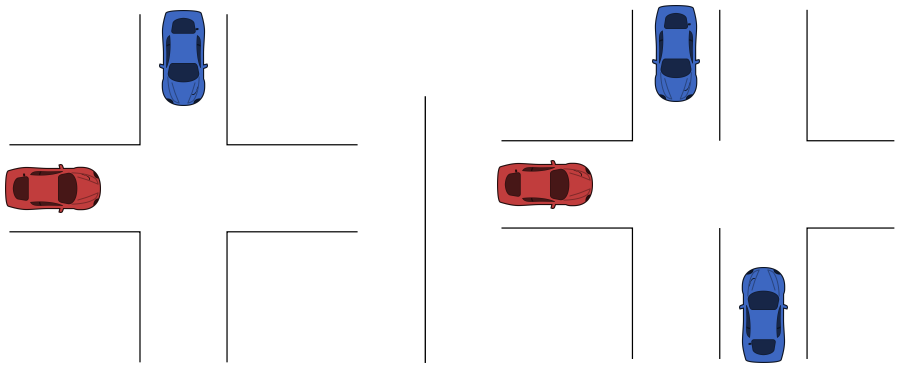


Figure 1: The two intersection scenarios considered in this work; single directional to the left and bidirectional to the right. The agent controls the red car.

Table 1: Parameters for simulator

Number of other vehicles, N	$\{1, 2, 3, 4\}$
Starting position ego, $p_{\mathrm{e}}^{\mathrm{c},j}$	$[50, 60]$ m
Starting position target, $p_{\mathrm{o}}^{\mathrm{c},j}$	$[10, 55]$ m
Desired velocity, v_{d}^{j}	$[8, 12]$ m/s

One quarter of the vehicles stop at the intersection and three quarters continue through the intersection, regardless of the behavior of the ego vehicle. When a vehicle has passed the intersection and reached the end of the road, it is moved back to the other side of the intersection, which creates a constant traffic flow. The simulator is updated at 25 Hz, and decisions are taken at 4 Hz. The goal of the ego vehicle is to reach a position that is located 10 m to the right of the last crossing point.

3.2 MDP formulation

The following Markov decision process is used to model the decision-making problem. The full state is not directly observable, since the intentions of the surrounding vehicles are not known to the agent. Therefore, the problem is a Partially Observable Markov Decision Process (POMDP) [24]. However,

by using a k-Markov approximation, where the state consists of the k last observations, the POMDP can be approximated as an MDP [18]. For the scenarios that were considered in this work, it proved sufficient to simply use the last observation.

State space, $\mathcal{S}$

The design of the state of the system,

$$s = (p_{\mathrm{e}}^{\mathrm{g}}, v_{\mathrm{e}}, a_{\mathrm{e}}, \{p_{\mathrm{e}}^{\mathrm{s},j}, p_{\mathrm{e}}^{\mathrm{c},j}, p_{\mathrm{o}}^{\mathrm{s},j}, p_{\mathrm{o}}^{\mathrm{c},j}, v_{\mathrm{o}}^{j}, a_{\mathrm{o}}^{j}\}_{j\in 0,\ldots,N}), \tag{C.7}$$

allows the description of intersections with different layouts [4]. The state, illustrated in Fig. 2, consists of the distance from the ego vehicle to the goal $p_{\mathrm{e}}^{\mathrm{g}}$, the velocity and acceleration of the ego vehicle, v_{e}, a_{e}, and the other vehicles, v_{o}^{j}, a_{o}^{j}, where j denotes the index of the other vehicles. Furthermore, $p_{\mathrm{e}}^{\mathrm{s},j}$ and $p_{\mathrm{e}}^{\mathrm{c},j}$ are the distances from the ego vehicle to the start of the intersection and crossing point, relative to target vehicle j respectively. The distances $p_{\mathrm{o}}^{\mathrm{s},j}$ and $p_{\mathrm{o}}^{\mathrm{c},j}$ are the distance from the other vehicles to the start of the intersection and the crossing point.

Action space, $\mathcal{A}$

The action space consists of six tactical decisions: {*'take way', 'give way', 'follow car* $\{1,\ldots,4\}$*'*}, which set the target of the IDM controller. The *'take way'* action treats the situation as an empty road, whereas the *'give way'* action sets a target distance of $p_{\mathrm{e}}^{\mathrm{s},j}$ and a target speed of 0 m/s. The *'follow car j'* actions sets the target distance to $p_{\mathrm{e}}^{\mathrm{c},j} - p_{\mathrm{o}}^{\mathrm{c},j}$ and target speed to v_{o}^{j}. In cases where $p_{\mathrm{o}}^{\mathrm{c},j} > p_{\mathrm{e}}^{\mathrm{c},j}$, the target distance is set to a value that corresponds to timegap 0.5 s. The output of the IDM model is further limited by a maximum jerk $j_{\max} = 5$ m/s^3 and maximum acceleration $a_{\max} = 5$ m/s^2. If less than four vehicles are present, the actions that correspond to choosing an absent vehicle are pruned by using Q-masking [25].

Reward model, R

The objective of the agent is to reach the goal on the other side of the intersection, without colliding with other vehicles and for comfort reasons, with as little jerk j_t as possible. Therefore, the reward at each time step r_t is

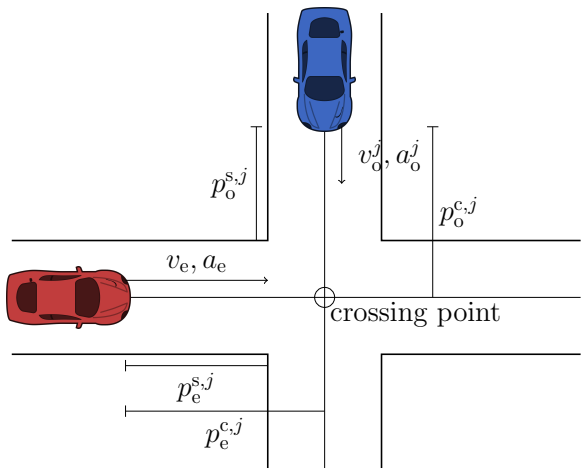


Figure 2: The state space definitions for a single crossing scenario, where subscript e and o denotes ego and other vehicle, respectively.

defined as

$$r_t = \begin{cases} 1 & \text{at reaching the goal,} \\ -1 & \text{at a collision,} \\ -\left(\frac{j_t}{j_{\max}}\right)^2 \frac{\Delta\tau}{\tau_{\max}} & \text{at non-terminating steps.} \end{cases}$$

The non-terminating reward is scaled with the maximum time of an episode, $\tau_{\max}$, and the step time $\Delta\tau = 0.04$ s, to ensure $\sum_{t=0}^{t=\tau_{\max}} \in [-1, 0]$. Further details about the reward function can be found in previous research [22].

Transition model, T

The state transition probabilities are not known to the agent. However, the true transition model is defined by the simulation model, described in Sect. 3.1.

3.3 Fallback action

As mentioned in Sect. 2.3, a fallback action a_{safe} is used when $c_{\mathrm{v}} > c_{\mathrm{v}}^{\mathrm{safe}}$ for all available actions. This fallback action is set to *'give way'*, with the

difference that no jerk limitation is applied and with a higher acceleration limit $a_{\max} = 10$ m/s^2.

3.4 Network architecture

In previous studies, we have showed that a network architecture that applies the same weights to the input that describes the surrounding vehicles results in a better performance and speeds up the training process [6], [4]. Such an architecture can be constructed by applying a one-dimensional convolutional neural network (CNN) structure to the surrounding vehicles' input. The network architecture that is used in this work is shown in Fig. 3. The first convolutional layer has 32 filters, with size and stride set to six, which equals the number of state inputs of each surrounding vehicle, and the second convolutional layers has 16 filter, with size and stride set to one. The fully connected (FC) layer that is connected to the ego vehicle input has 16 units, and the joint fully connected layer has 64 units. All layers use rectified linear units (ReLUs) as activation functions, except for the last layer, which has a linear activation function. The final dueling structure of the network separates the estimation of the state value $V(s)$ and the action advantage $A(s, a)$ [26]. The input vector is normalized to the range $[-1, 1]$. The input vector contains slots for four surrounding vehicles, and if less vehicles are present in the traffic scene, the empty input is set to -1.

3.5 Training process

Algorithm 3 is used to train the agent. The loss function of Double DQN is applied, which subtly modifies the maximization operation of Eq. C.3 to $\gamma Q(s', \text{argmax}_{a'} Q(s', a'; \theta_i); \theta_i^-)$ [27]. The Adam optimizer is used to update the weights [28], and K parallel workers are used for the backpropagation step. The hyperparameters of the training process are shown in Table 2, and the values were selected by an informal search, due to the computational complexity.

If the current policy of the agent decides to stop the ego vehicle, an episode could continue forever. Therefore, a timeout time is set to $\tau_{\max} = 20$ s, at which the episode terminates. The last experience of such an episode is not added to the replay memory. This trick prevents the agent to learn that an episode can end due to a timeout, and makes it seem like an episode can

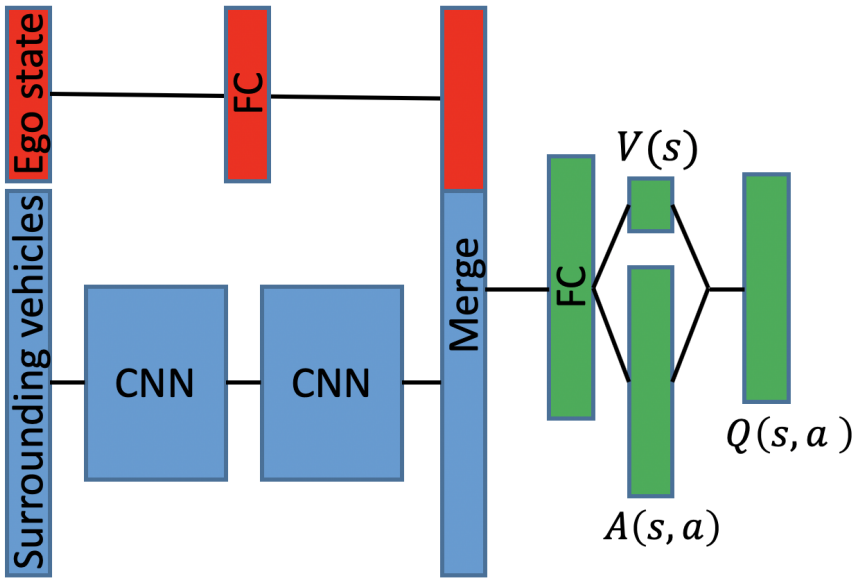


Figure 3: The neural network architecture that was used in this work.

continue forever, which is important, since the terminating state due to the time limit is not part of the MDP [6].

3.6 Baseline method

The Double DQN method, hereafter simply referred to as the DQN method, is used as a baseline. For a fair comparison, the same hyperparameters as for the ensemble RPF method is used, with the addition of an annealing ϵ-greedy exploration schedule, which is shown in Table 2. During test episodes, a greedy policy is used.

4 Results

The results show that the ensemble RPF method outperforms the DQN method, both in terms of training speed and final performance, when the resulting

Table 2: Hyperparameters of Algorithm 3 and baseline DQN.

Number of ensemble members, K	10
Prior scale factor, β	1
Experience adding probability, p_{add}	0.5
Discount factor, γ	0.99
Learning start iteration, N_{start}	50,000
Replay memory size, M_{replay}	500,000
Learning rate, η	0.0005
Mini-batch size, M_{mini}	32
Target network update frequency, N_{update}	20,000
Huber loss threshold, δ	10
Initial exploration constant, ϵ_{start}	1
Final exploration constant, ϵ_{end}	0.05
Final exploration iteration, $N_{\epsilon\text{-end}}$	1,000,000

agents are tested on scenarios that are similar to the training scenarios. When the fully trained ensemble RPF agent is exposed to situations that are outside of the training distribution, the agent indicates a high uncertainty and chooses safe actions, whereas the DQN agent collides with other vehicles. More details on the characteristics of the results are presented and briefly discussed in this section, whereas a more general discussion follows in Sect. 5.

The ensemble RPF and DQN agents were trained in the simulated environment that was described in Sect. 3. After every 50,000 training steps, the performance of the agents were evaluated on 100 random test episodes. These test episodes were randomly generated in the same way as the training episodes, but kept fixed for all the evaluation phases.

4.1 Within training distribution

The average return and the average proportion of episodes where the ego vehicle reached the goal, as a function of number of training steps, is shown in Fig. 4, for the test episodes. The figure also shows the standard deviation

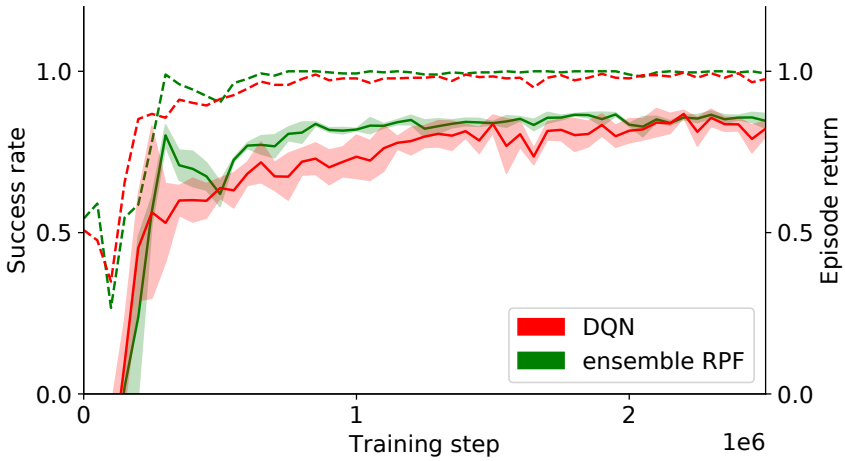


Figure 4: Proportion of test episodes where the ego vehicle reached its goal (dashed), and episode return (solid), over training steps for the ensemble RPF and DQN methods. The shaded areas show the standard deviation for 5 random seeds.

for 5 random seeds, which generates different sets of initial parameters of the networks and different training episodes, whereas the test episodes are kept fixed. The results show that the ensemble RPF method both learns faster, yields a higher return, and causes less collisions than the DQN method.

Fig. 5 shows how the coefficient of variation c_v of the chosen action varies during the testing episodes. Note that the uncertainty of actions that are not chosen can be higher, which is often the case. After around one million training steps, the average value of c_v settles at around 0.04, with a 99 percentile value of 0.15, which motivates the choice of setting $c_\mathrm{v}^\mathrm{safe} = 0.2$.

As shown in Fig. 4, occasional collisions still occur during the test episodes when deploying the fully trained ensemble RPF agent. The reasons for these collisions are further discussed in Sect. 5. In one particular example of a collision, the agent fails to brake early enough and ends up in an impossible situation, where it collides with another vehicle in the intersection. However, the estimated uncertainty increases significantly during the time before the collision, when the incorrect actions are taken, see Fig. 6. When applying the

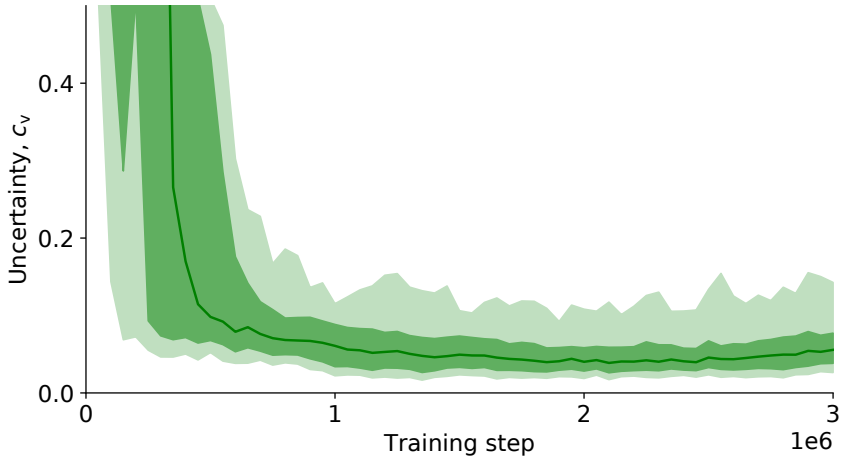


Figure 5: Mean coefficient of variation c_v for the chosen action during the test episodes. The dark shaded area shows percentiles 10 to 90, and the bright shaded area shows percentiles 1 to 99.

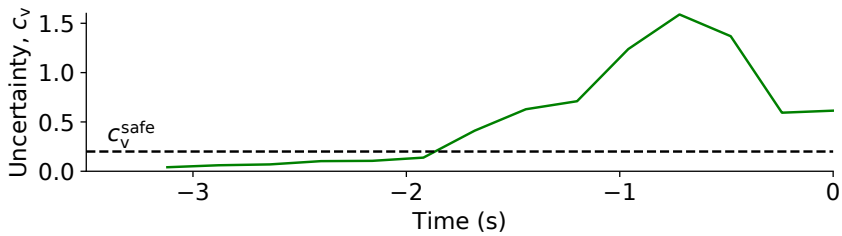


Figure 6: Uncertainty c_v during the time steps before one of the collisions in the test episodes, within the training distribution. The collision occurs at $t = 0$ s.

confidence criterion (Sect. 2.3), the agent instead brakes early enough, and can thereby avoid the collision. The confidence criterion was also applied to all the test episodes, which removed all collisions.

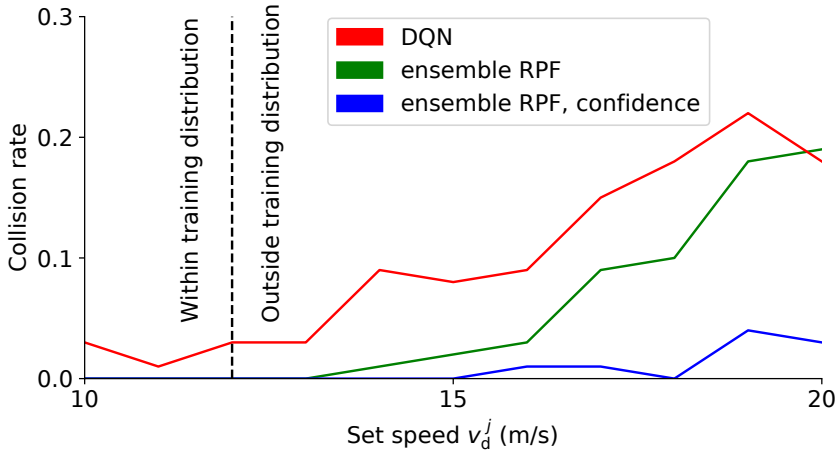
4.2 Outside training distribution

The ensemble RPF agent that was obtained after three million training steps was tested in scenarios outside of the training distribution, in order to evaluate the agent's ability to detect unseen situations. The same testing scenarios as for within the distribution was used, with the exception that the speed of the surrounding vehicles was set to a single deterministic value, which was varied during different runs in the range $v_{\mathrm{d}}^{j} = [10, 20]$ m/s. The proportion of collisions as a function of set speed of the surrounding vehicles is shown in Fig. 7, together with the proportion of episodes where the confidence criterion was violated at least once. The figure shows that when the confidence criterion is used, most of the collisions can be avoided. Furthermore, the violations of the criterion increase when the speed of the surrounding vehicles increase, i.e., the scenarios move further from the training distribution.

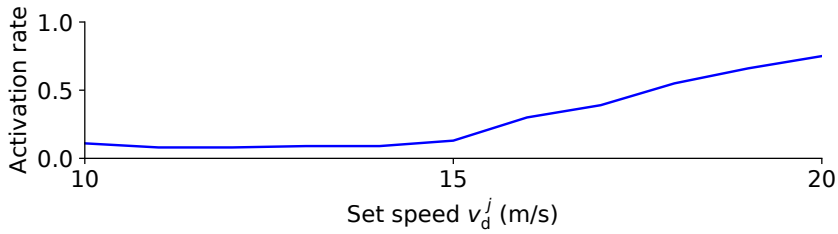
An example of a situation that causes a collision is shown in Fig. 8, where an approaching vehicle drives with a speed of 20 m/s. The Q-values of both the trained ensemble RPF and DQN agents indicate that the agents expect to make it over the crossing before the other vehicle. However, since the approaching vehicle drives faster than what the agents have seen during the training, a collision occurs. When the confidence criterion is applied, the uncertainty rises to $c_{\mathrm{v}} > c_{\mathrm{v}}^{\mathrm{safe}}$ for all actions when the ego vehicle approaches the critical region, where it has to brake in order to be able to stop, and a collision is avoided by choosing action a_{safe}.

5 Discussion

The results show that the ensemble RPF method can indicate an elevated uncertainty for situations that the agent has been insufficiently trained for, both within and outside of the training distribution. In previous work by the authors of this paper, we observed similar results when using the ensemble RPF method to estimate uncertainty outside of the training distribution in a highway driving scenario [16]. In contrast, this paper shows that, in some cases, the ensemble RPF method can even detect situations with high uncertainty within the training distribution. Such situations include rare events that seldom or never occur during the training process, which makes it hard for the agent to provide an accurate estimate of the Q-values for the corresponding states. Since these states are seldom used to update the neural networks of the



(a) Proportion of collisions.

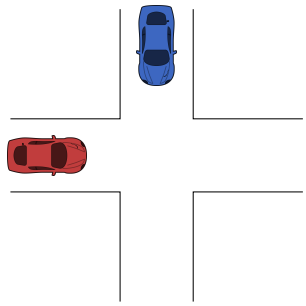


(b) Proportion of episodes where a_{safe} was used at least once.

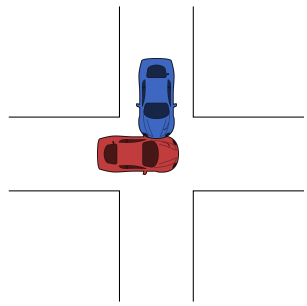
Figure 7: Performance of the ensemble RPF agent, with and without the confidence criterion, and the DQN agent, in test episodes with different set speeds v_{d}^j for the surrounding vehicles.

ensemble, the weights of the trainable networks will not adapt to the respective prior networks, and the uncertainty measure c_{v} will remain high for these rare events. This information is useful to detect edge cases within the training set and indicate when the decision of the trained agent is not fully reliable.

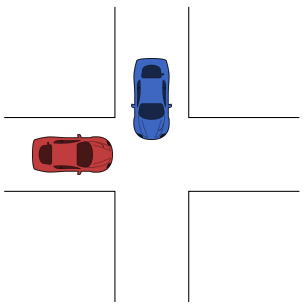
In this work, the estimated uncertainty is used to choose a safe fallback action if the uncertainty exceeds a threshold value. For the cases that are considered here, this confidence criterion removes all collisions within the



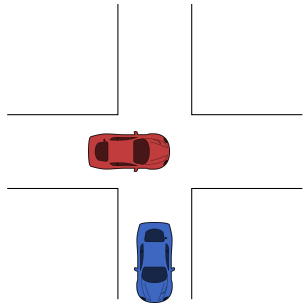
(a) $t = 0$, initial situation.



(b) $t = 1$, DQN and ensemble RPF without confidence criterion.



(c) $t = 1$, ensemble RPF with confidence criterion.



(d) $t = 1.5$, ensemble RPF with confidence criterion.

Figure 8: Example of a situation outside of the training distribution, where there would be a collision if the confidence criterion is not used. The vehicle at the top is here approaching the crossing at 20 m/s.

training distribution, and almost all collisions when the speed of the surrounding vehicles is increased to levels outside of the training distribution. However, to guarantee safety by using a learning-based method is challenging, and an underlying safety layer is often used [29]. The presented method could decrease the number of activations of such a safety layer, but possibly more importantly, the uncertainty measure could also be used to guide the training process to focus on situations that the current agent needs to explore further. Moreover, if an agent is trained in simulation and then deployed in real traffic, the uncertainty estimation of the agent could detect situations that should be added to the simulated world, in order to better match real-world driving.

The results show that the ensemble RPF method performs better and more stable than a standard DQN method within the training distribution. The main disadvantage is the increased computational complexity, since K neural networks need to be trained. This disadvantage is somewhat mitigated in practice, since the design of the algorithm allows an efficient parallelization. Furthermore, the tuning complexity of the ensemble RPF and DQN methods are similar. Hyperparameters for the number of ensemble members K and prior scale factor β are introduced, but the parameters that control the exploration of DQN are removed.

6 Conclusion

The results of this paper demonstrates the usefulness of using a Bayesian RL technique for tactical-decision making in an intersection scenario. The ensemble RPF method can be used to estimate the confidence of the recommended actions, and the results show that the trained agent indicates high uncertainty for situations that are outside of the training distribution. Importantly, the method also indicates high uncertainty for rare events within the training distribution. In this work, the confidence information was used to choose a safe action in situations with high uncertainty, which removed all collisions from within the training distribution, and most of the collisions in situations outside of the training distribution.

The uncertainty information could also be used to identify situations that are not known to the agent, and guide the training process accordingly. To investigate this further is a topic for future work. Another subject for future work involves how to set the parameter value c_v^{safe} in a more systematic way, and how to automatically update the value during training.

References

[1] "Traffic safety facts," National Highway Traffic Safety Administration, Tech. Rep. DOT HS 812 261, 2014.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.

[3] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, "Navigating occluded intersections with autonomous vehicles using deep reinforcement learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[4] 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.

[5] P. Wang, C. Chan, and A. d. L. Fortelle, "A reinforcement learning based approach for automated lane change maneuvers," in *IEEE Intelligent Vehicles Symposium (IV)*, 2018.

[6] C. J. Hoel, K. Wolff, and L. Laine, "Automated speed and lane change decision making using deep reinforcement learning," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2018.

[7] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," arXiv:1610.03295, 2016.

[8] X. Pan, Y. You, Z. Wang, and C. Lu, "Virtual to real reinforcement learning for autonomous driving," in *Proc. of the Brit. Machine Vision Conf. (BMVC)*, 2017.

[9] M. Bansal, A. Krizhevsky, and A. S. Ogale, "ChauffeurNet: Learning to drive by imitating the best and synthesizing the worst," *Robotics: Science and Systems*, 2019.

[10] A. Kendall, V. Badrinarayanan, and R. Cipolla, "Bayesian SegNet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding," in *Proc. of the Brit. Machine Vision Conf. (BMVC)*, 2017.

[11] R. McAllister *et al.*, "Concrete problems for autonomous vehicle safety: Advantages of Bayesian deep learning," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

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

[13] R. Michelmore, M. Kwiatkowska, and Y. Gal, "Evaluating uncertainty quantification in end-to-end autonomous driving control," *Computing Research Repository (CoRR)*, vol. 1811.06817, 2018.

[14] R. Dearden, N. Friedman, and S. Russell, "Bayesian Q-learning," in *AAAI Conference on Artificial Intelligence (AAAI)*, 1998.

[15] I. Osband, J. Aslanides, and A. Cassirer, "Randomized prior functions for deep reinforcement learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[16] C.-J. Hoel, K. Wolff, and L. Laine, "Tactical decision-making in autonomous driving by reinforcement learning with uncertainty estimation," in *IEEE Intelligent Vehicles Symposium (IV)*, 2020.

[17] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[18] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[19] J. García and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 42, pp. 1437–1480, 2015.

[20] B. Efron, *The Jackknife, the Bootstrap and Other Resampling Plans*. Society for Industrial and Applied Mathematics, 1982.

[21] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped DQN," in *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[22] T. Tram, I. Batkovic, M. Ali, and J. Sjöberg, "Learning when to drive in intersections by combining reinforcement learning and model predictive control," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, Oct. 2019.

[23] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical Review E*, vol. 62, pp. 1805–1824, 2 2000.

[24] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998, ISSN: 0004-3702.

[25] 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.

[26] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International Conference on Machine Learning (ICML)*, vol. 48, 2016.

[27] H. van Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver, "Learning values across many orders of magnitude," in *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations (ICLR)*, Dec. 2014.

[29] S. Underwood, D. Bartz, A. Kade, and M. Crawford, "Truck automation: Testing and trusting the virtual driver," in *Road Veh. Automat. 3*, G. Meyer and S. Beiker, Eds., Springer, 2016, pp. 91–109.

PAPER D

Belief State Reinforcement Learning for Autonomous Vehicles in Intersections

Tommy Tram, Maxime Bouton, Jonas Fredriksson, Jonas Sjöberg, and Mykel Kochenderfer

Submitted to IEEE Transactions on Intelligent Transportation Systems,
 DOI: TBD.

The layout has been revised.

Abstract

This paper investigates different approaches to find a safe and efficient driving strategy through an intersection with other drivers. Because the intentions of the other drivers to yield, stop, or go are not observable, we use a particle filter to maintain a belief state. We study how a reinforcement learning agent can use these representations efficiently during training and evaluation. This paper shows that an agent trained without any consideration of the intentions of others is both slower at reaching the goal and results in more collisions. Four algorithms that use a belief state generated by a particle filter are compared. Two of the algorithms have access to the intention only during training while the others do not. The results show that explicitly trying to predict the intention gave the best performance in terms of safety and efficiency.

1 Introduction

Advancements in Autonomous Vehicles (AV) technology is expected to create a paradigm shift in how we transport ourselves in the near future and even where we choose to live [1]. However, one of the major concerns is how to tackle the high social demand for traffic safety under various conditions and scenarios. Replacing the human driver with a computer to make all tactical and operational decisions is a difficult task to design. In particular, decision-making in scenarios that require interaction with other road participants, as summarized in a review paper [2].

In recent years, decision-making for AVs in structured scenarios like intersections has attracted a lot of attention in the research society and the automotive industry. A skilled engineer can sometimes solve the decision-making problem for structured traffic scenarios using rule-based methods, e.g., [3] and [4]. Rule-based decision-making is commonly implemented using hierarchical state machines to switch between predefined behaviors, as shown in [3]. However, it can be difficult for an engineer to anticipate every situation and design a suitable strategy that can solve all of them, in particular when drivers are not following the law [5], and statistics from the Insurance Institute for Highway

Safety estimated in 2019 that 143,000 people in the US were injured in red light running crashes and 846 of them were killed [6]. Hence, decision-making for AVs cannot only rely on traffic rules to be safe, but they also need to be prepared when other traffic participants do not follow them. Furthermore, rule-based approaches has difficulty generalizing to unknown situations and deal with uncertainties, such as the uncertainty of whether other traffic participants intend to follow the rules or not.

In order to handle the uncertainty of predicting other traffic participants' behaviors or intentions, the literature formulates the problem of driving under uncertainty as a Partially Observable Markov Decision Process (POMDP), e.g., with intentions as a non-observable state. POMDPs can be solved by simulating all possible future motions given different potential behaviors and/or intentions [7], [8], e.g. Monte Carlo Tree Search (MCTS). Unfortunately, MCTS requires extensive online computation and can be hard to scale in complex traffic situations with an increasing number of traffic participants, especially when a participant's actions are interdependent on all other participants' actions. Learning-based approaches, like Reinforcement Learning (RL) [9], [10], can help relieve the burden of designing hand-crafted solutions for all possible scenarios and Deep Recurrent Q-Network (DRQN) approaches, [11], [12], showed some promise solving POMDP with non-observable states by leveraging past observations or actions. One such approach was proposed in our previous work [13], where a decision-making policy that identifies and chooses a gap in-between cars in an intersection scenario by using a deep Q-learning [14] method and an Long Short-Term Memory (LSTM) network architecture [15]. The results showed that the policy solved the decision-making problem up to 97% of the time under perfect sensing conditions. The few cases where the AV ended up in collisions occurred when the AV and another vehicle had the same velocity and the same distance to the intersection, i.e., the policy had difficulties sorting out the situation because the estimation of the hidden state (intention) was embedded in the neural network architecture. The prediction of surrounding vehicles was especially difficult for low velocities.

To explicitly estimate the intention of other drivers is not an easy task [16]. Liebner *et al.* [17] proposed to estimate driver behavior using a driver model, the so-called Intelligent Driver Model (IDM) [18], to infer driver intent in urban intersections, and Hoermann *et al.* [19] used a particle filter to estimate the parameters of the IDM, both works showed promising results when evaluated on

real-world data. Another approach to estimating the intention is to introduce belief states to capture uncertainties in the environment [20]. The belief state can be used to model the probability distribution over the uncertain world states, e.g., the intention of other road users. Wang *et al.* [21] decoupled the belief state modeling (via unsupervised learning) from policy optimization (via RL) and claimed that having full observability at learning time, combined with knowing what will not be observable at deployment time, enables an RL agent to learn a policy that is more robust to its unobservable states. Another approach using full observability at learning time is QMDP [22], where a model is first trained on the full state space, including the unobservable states, and later evaluated the model on the belief state. Training on full observability can only be done when the unobservable state can be obtained, e.g., using ground truth data and labeling.

In this paper, we propose a decoupled approach, similar to [21], but model the belief state using a particle filter that is tailor-made for the intersection and use the IDM to model the possible behavior of different intentions similar to [17] and [19]. The policy optimization uses the deep Q-learning approach from [13], but without the LSTM layer. More specifically, we introduce two novel approaches that are safer than their respective baseline by maintaining a belief state using a particle filter and estimating the unobservable states from the belief state before feeding it to the Deep Q-network (DQN), which gives the final action. The first approach uses the probability distribution of the intention as input to the DQN and the second approach uses a likelihood estimate with a threshold value for the intention decision to make an estimate of the intention before feeding it to the DQN. The proposed approaches are compared with a naive approach that trains on the entire belief state, a standard approximation method from the literature, QMDP [22], and a DQN with full observability. The latter approach represents an upper bound of what a DQN is capable of if it has perfect sensing and is referred to as the Oracle DQN.

The main contributions of this paper are:

- A decoupled approach that separates belief state modeling, using a particle filter, and policy optimization, using deep Q-learning;
- a tailor-made particle filter implementation with a separate transition model for each intention state;
- representing the intention state as a probability distribution and two

approaches for using the distribution in DQN.

2 Mathematical preliminaries

This section describes the relevant parts of the POMDP framework and the notations used in the problem formulation. It also briefly introduces deep Q-learning, which is used to find the driving policy, and the approximation method QMDP used to handle the unknown intention of the other drivers. Finally, a driving model that describes the behaviors of other drivers is presented. The driver model is later used on multiple occasions in this paper, both as the transition model of all vehicles in the simulation environment and the prediction model in the particle filter.

2.1 Partially observable Markov decision process

A POMDP is a mathematical framework for modeling sequential decision making problems under uncertainty. It is a generalization of the Markov Decision Process (MDP) that satisfies the Markov property, which requires that the probability distribution of the next state only depends on the current state and the action taken by the agent, not the history of states. A POMDP consists of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a transition model T, a reward function R, a set of observations Ω, an observation model O, and a discount factor $\gamma \in [0, 1)$. Together they create the tuple $(\mathcal{S}, \mathcal{A}, T, R, \Omega, O, \gamma)$ that formally defines a POMDP [23]. The goal is to find a policy that maximizes the expected return.

In this paper, we denote the driving policy π, i.e., the driving decision, which in a POMDP is formally defined as a mapping from a belief state to an action, where the action corresponds to the decision. The probability of transitioning to a future state s', given a current state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, is described by the transition model $T(s' \mid s, a)$, while the reward r for transitioning to the new state s' is given by the reward function $R(s, a)$. Note that, in the POMDP framework, the agent only has access to partial information contained in its observation o distributed according to $\Pr(o \mid s', a) = O(o, s', a)$.

To accommodate for the missing information, the agent maintains a belief state. A belief state b is a probability distribution such that

$$b(s) = \Pr(s \mid o_{1:t}) \tag{D.1}$$

is the probability of being in state s at time t, given observations $o_{1:t} := \{o_1, o_2, ..., o_t\}$ up to time t. At each time step t, the agent updates its belief using a Bayesian filtering approach given the previous belief and the current observation as follows:

$$b'(s') \propto O(o \mid s', a) \int_{s \in S} T(s' \mid s, a) b(s). \tag{D.2}$$

To evaluate different actions a, an action-value function $Q(b, a)$ is used

$$\begin{aligned} Q(b,a)^{\pi} = R(b,a) + \\ \gamma \int_o \max_{a \in \mathcal{A}} \int_s b(s) \int_{s'} O(o \mid s', a) T(s' \mid s, a) \alpha(s'). \end{aligned} \tag{D.3}$$

The action-value function $Q(b, a)^{\pi}$ represents the expected discounted accumulated reward received by following policy π after taking action a from belief state b. The optimal policy π^* can be found by maximizing the reward

$$\pi^*(b) = \operatorname*{argmax}_a Q(b, a). \tag{D.4}$$

For real-world problems, finding the optimal policy from a belief state is intractable, especially when the state space is continuous [23]. In this case, the belief state can be represented as a collection of particles, which simply are samples from the state space, and an approximation referred to as QMDP can be used:

$$Q(b, a) \approx \sum_s b(s) Q_{\text{MDP}}(s, a) \tag{D.5}$$

where Q_{MDP} is the optimal value function of the MDP version of the problem that assumes that the agent has full observability.

While finding the optimal $Q^*(b, a)$, in (D.4), may be hard, finding $Q_{\text{MDP}}(s, a)$ is usually easier. When the transition function is explicitly defined and the state space is finite, dynamic programming can be used to compute $Q_{\text{MDP}}(s, a)$. However, the transition function is often only accessible in the form of a generative model (*e.g.* a simulator), and the state space is high-dimensional and continuous. In such settings, deep Q-learning to approximate $Q_{\text{MDP}}(s, a)$ can be used, [22].

In deep Q-learning, the action-value function is represented by a neural network. The function approximating the optimal value function is given by

minimizing the loss function derived from the Bellman equation:

$$J(\theta) = \mathbb{E}_{s'}[(r + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta))^2] \tag{D.6}$$

where θ represents the weights of the neural network and (s, a, r, s') is obtained from one simulation step in the environment.

2.2 Behavior model

The general behavior of surrounding vehicles is modeled using IDM. Besides describing the behavior of all vehicles, including the ego vehicle, the IDM is also used in the prediction model of the particle filter. The IDM models a vehicle n's position and velocity as

$$\dot{p}_n = v_n \tag{D.7}$$

$$\dot{v}_n = a_{\max}\Big(1 - \Big(\frac{v_n}{v_n^{\text{desired}}}\Big)^{\delta} - \Big(\frac{d^*(v_n, \Delta v_n)}{d_n}\Big)^2\Big) \tag{D.8}$$

$$\text{with } d^*(v_n, \Delta v_n) = d_0 + v_n T_{\text{gap}} + \frac{v_n \Delta v_n}{2\sqrt{a_{\max}\alpha_b}}$$

where v_n^{desired} is the desired velocity, d_0 is the minimum distance between cars, T_{gap} is the desired time gap to the vehicle in front, $a_{\max}$ is the maximum vehicle acceleration, d_n is the distance to the vehicle in front, $\Delta v_n = v_n - v_{n-1}$ is the velocity difference between vehicle n and the vehicle directly in front $n-1$, and α_b and δ are model parameters for comfortable deceleration or acceleration.

The acceleration can be simplified into two terms: an interaction term for when there is a vehicle in front

$$\begin{aligned} a_n^{\text{int}} &= -a_{max}\Big(\frac{d^*(v_n, \Delta v_n)}{d_n}\Big)^2 \\ &= -a_{max}\Big(\frac{d_0 + v_n T_{\text{gap}}}{d_n} + \frac{v_n \Delta v_n}{2\sqrt{a_{\max}\alpha_b} d_n}\Big)^2 \end{aligned} \tag{D.9}$$

and free road term, when there is no leading vehicle

$$a_n^{\text{free}} = a_{\max}\Big(1 - \Big(\frac{v_n}{v_n^{\text{desired}}}\Big)^{\delta}\Big). \tag{D.10}$$

3 Proposed approach

In this section, the intersection problem is modeled as a POMDP, and the particle filter used to estimate the belief state and its uncertainty is introduced. Finally, we present the RL algorithms that are used to train the agent, controlling the ego vehicle.

3.1 POMDP formulation

The problem setting considered in this work is an intersection shown in Figure 1, and here we outline a general overview of the POMDP formulation, whereas Section 4 specifies the numerical values. The ego vehicle approaches an intersection with at least one other vehicle on the perpendicular lane with about the same time-to-intersection assuming they keep a constant velocity. The goal for the ego vehicle is to reach the other side of the intersection as fast as possible without colliding with the other cars. With only onboard sensors on the ego vehicle, it can observe the physical state of other vehicles but not the drivers' intention.

Such an intersection problem can be described as a POMDP with $(\mathcal{S}, \mathcal{A}, T, R, \Omega, O, \gamma)$ as:

State space, $\mathcal{S}$

The states of the system,

$$s = (p_{\text{ego}}^{\text{goal}}, p_{\text{ego}}^{\text{int}}, v_{\text{ego}}, t_{\text{stop}}, \{p_n, v_n, \zeta_n\}_{n=1}^{N}), \tag{D.11}$$

consists of the ego vehicle state and the states of the surrounding vehicles. The ego vehicle state is described by the distance to the intersection $p_{\text{ego}}^{\text{int}}$, the distance to the goal $p_{\text{ego}}^{\text{goal}}$, and the velocity v_{ego}. Stop time t_{stop} is the time the ego vehicle is at a standstill $v_{\text{ego}} = 0$. This state tracks the amount of time the ego vehicle has been standing still at the intersection and how far it is from potentially reaching a terminal state, defined later in Section 4.1. The states of the surrounding vehicles, indexed by $n \in \{1, \dots, N\}$, are described by the distance to the intersection p_n, the velocity v_n, and the intention ζ_n. The intentions are defined as one-hot vectors and can either be yield, $\zeta_n^{\text{yd}} = [0\,1]$, which means that the vehicle will stop before the intersection, or take way, $\zeta_n^{\text{tw}} = [1\,0]$, which means that the vehicle will drive through the intersection. These intentions control the behavior of the surrounding vehicles.

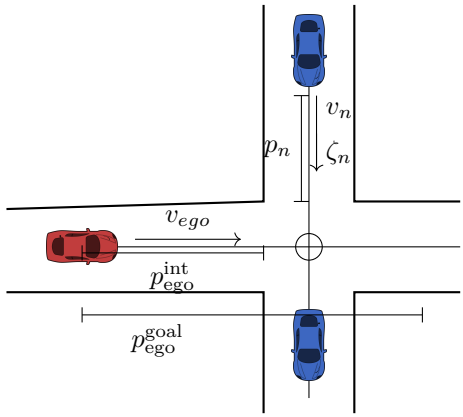


Figure 1: State definitions for the intersection. The red vehicle is the ego vehicle.

Action space, $\mathcal{A}$

The motion of the ego vehicle is controlled by changing the acceleration according to the IDM (D.8). This is done by two high-level actions *take way* and *yield*. The take way action sets the acceleration of the ego vehicle $a = a^{\text{free}}$ according to (D.10). While the yield action calculate a^{yield} by using (D.9) to slow down and stop at the intersection with $d_n = p_{\text{ego}}^{\text{int}}$ and $v_{n-1} = 0$. If there is a car in front, the acceleration becomes $a = \min(a^{\text{int}}, a^{\text{yield}})$.

Transition model, T

The state transition probabilities are defined by the dynamics and intentions of the crossing and ego vehicles. However, the dynamics of crossing vehicles are not known to the ego vehicle. The dynamics in this work are described by the IDM, and implemented as a simulation model defined in Section 4.1.

Reward function, R

The reward function is designed to encourage safety and time efficiency. The agent's main goal is to reach the other side of the intersection without colliding with other traffic participants. There are one non-terminal state and four terminal states: (i) goal, (ii) safe stop, (iii) collision, and (iv) deadlock. While goal and collision are self-explanatory, safe stop and deadlock are reached when

the agent chooses to stop at the intersection for T_{stop} consecutive seconds. To distinguish whether a stop was efficient, there are two different outcomes. If the other vehicles are at standstill and waiting for the ego vehicle, deadlock is reached, while a safe stop is reached if all other vehicles are in motion. All states other than the terminal state give a small negative reward to incentivize reaching a terminal state as quickly as possible.

Observation space, Ω

An observation o consists of the ego vehicle's state and the physical state of the surrounding vehicles, (p_n, v_n), while the intentions of the surrounding vehicles ζ_n are not observed. An observation is described by

$$o = (p_{\text{ego}}^{\text{goal}}, p_{\text{ego}}^{\text{int}}, v_{\text{ego}}, t_{\text{stop}}, \{\hat{p}_n, \hat{v}_n\}_{n=1}^{N}). \tag{D.12}$$

where the $\hat{}$ notation represents the observed states with noise.

Observation model, O

The ego vehicle observes its own states without noise, while it observers noisy measurements of the positions and speeds of the surrounding vehicles

$$\hat{p}_n = p_n + \epsilon_{\text{p}}, \tag{D.13}$$

$$\hat{v}_n = v_n + \epsilon_{\text{v}} \tag{D.14}$$

where, $\epsilon_{\text{p}} \sim \mathcal{N}(0, \sigma_p^2)$ and $\epsilon_{\text{v}} \sim \mathcal{N}(0, \sigma_v^2)$.

3.2 Belief state representation using a particle filter

This section describes how the belief state is estimated using a particle filter. The algorithm is described, along with the assumptions made on the interaction behaviors of the other vehicles in the intersection scenario. When the state space is continuous and not well approximated by a linear Gaussian model, as it is in (D.11), a sampling-based approach can be used to perform belief updates, where the belief state is represented as a collection of particles with weights [23].

The key idea is to initialize a set of particles with a uniform distribution of intention states ζ. Then, predict the future state of these particles, using a prediction model depending on ζ (D.9, D.10). Finally, sample a new set

of particles using the observation and the predicted states. The new set of particles will then have a better distribution of intentions.

Let s_n be the three-dimensional state, (p_n, v_n, ζ_n), of an observed car n from the state space in (D.11). Then one particle $x_{n[m]} = (p_{n[m]}, v_{n[m]}, \zeta_{n[m]})$ represents one *belief* of s_n, i.e., one sample of the estimated distribution, where $p_{n[m]}$ is the particle position, $\tilde{v}_{n[m]}$ the particle velocity, $\zeta_{n[m]}$ the particle intention and m the particle index out to the M total number of particles.

Algorithm 4 Initialize particle state for new car

Input: $(\hat{p}_n, \hat{v}_n) \in o$
Output: $\{x_{n[m]}, w_{n[m]}\}_{m=1}^{M}$

for $m = 1, \ldots, M$ **do**
  $p_{n[m]} \sim \mathcal{N}(\hat{p}_n, \sigma_p^2)$,
  $v_{n[m]} \sim \mathcal{N}(\hat{v}_n, \sigma_v^2)$,
  $\zeta_{n[m]} \leftarrow \begin{cases} \zeta_{\text{tw}}, & \text{if } m \leq M/2 \\ \zeta_{\text{yd}}, & \text{otherwise} \end{cases}$
  $x_{n[m]} = (\tilde{p}_{n[m]}, \tilde{v}_{n[m]}, \tilde{\zeta}_{n[m]})$,
  $w_{n[m]} = 1/M$

When a car is observed for the first time, the particle position $p_{n[m]}$ and velocity $v_{n[m]}$ state are sampled around the first observation $o_n = (\hat{p}_n, \hat{v}_n)$ with variance σ_p^2 and σ_v^2. The particle intention states are initialized with 50% yield and 50% take way with the same particle weight $w_{n[m]}$. The initialization of the particles for a new car is summarized in Algorithm 4. The particles for each car are then combined into a joint particle state

$$x_m = \{x_{1[m]}, \ldots, x_{N[m]}\} \tag{D.15}$$

and as the observation on ego vehicle $s_{\text{ego}} = (p_{\text{ego}}^{\text{goal}}, p_{\text{ego}}^{\text{int}}, v_{\text{ego}}, t_{\text{stop}})$ has no noise, the belief state $b(s)$ then becomes

$$b(s) = (s_{\text{ego}}, \{x_m\}_{m=1}^{M}). \tag{D.16}$$

The order of cars is set to $n = 1, \ldots N$, and the motion of each car n is assumed to depend on the car directly in front $n-1$ and independent of all other cars, allowing us to factor the joint distribution transition model as:

$$\Pr(s'_{1:N} \mid s_{1:N}) = \Pr(s'_1 \mid s_1) \prod_{n=2}^{N} \Pr(s'_n \mid s_n, s_{n-1}). \tag{D.17}$$

In addition, the observations of each vehicle $o_n = (\hat{p}_n, \hat{v}_n)$ are assumed to only depend on its own true state s_n and independent of the states of the other cars. Then the joint observation distribution is:

$$\Pr(o_{1:N} \mid s_{1:N}) = \prod_{n=1}^{N} \Pr(o_n \mid s_n). \tag{D.18}$$

Given these two assumptions, we can define the full particle filter procedure for updating the joint particle state.

At each update step, the standard particle filter operations of prediction and measurement updates are performed with a few modifications. For each particle, x_m, simulate one step forward using the transition model defined in (D.9) or (D.10) depending on the particle intention state $\zeta_{n[m]}$. Noise is added to the particle state to help prevent particle depletion. This is done in two ways. First, the intention of each particle has a slight probability ξ_i to change its intention state. Second, acceleration noise σ_a is added to the prediction model. The observation weights $w_{n[m]}$ for each individual particle are generated using a Gaussian sensor model for the position and velocity, as the observation noise is assumed to be normally distributed, according to (D.13) and (D.14). The weight of a joint particle w_m is then given by multiplying each of the individual vehicle weights $w_{n[m]}$ as described in D.18. The weights w are then normalized and together with the particle set $x' = \{x'_m\}_{m=1}^{M}$, a new set of particles is resampled, using sequential importance resampling [25]. The new set of joint particles x'_m are then used to represent the belief state b according to (D.16). The full particle update process is summarized in Algorithm 5.

3.3 Belief state reinforcement learning algorithms

Building upon the belief state $b(s)$ introduced in the previous chapter, our goal is to get an optimal policy π^* according to (D.4) by training a DQN using this belief. This section describes our two proposed approaches, QID (intention distribution) and QMDP-IE (intention estimation). The proposed approaches are compared with both a naive approach, henceforth referred to as QPF, and an established approximation approach QMDP [22]. To compare the different approaches fairly, a common DQN architecture is used for all four. The DQN uses a combination of improvement suggestions from [26], e.g., double DQN [27] and experience replay [28], but from here on out is referred

Algorithm 5 Update the joint particle state

Current particle state: $\{x_m, w_m\}_{m=1}^{M}$
New observation: o'
Updated particle state: $\{x'_m, w'_m\}_{m=1}^{M}$

$w_{\text{sum}} = 0$
for $m = 1, \ldots, M$ **do**
  for $n = 1, \ldots, N$ **do**
    if $x_{n[m]}$ is new **then**
      See Algorithm 4
    else
      if $\mathcal{U}(0,1) < \xi_i$ **then**

$$\zeta_{n[m]} = \begin{cases} \zeta^{\text{tw}}, & \text{if } \zeta_{n[m]} = \zeta^{\text{yd}} \\ \zeta^{\text{yd}}, & \text{otherwise} \end{cases}$$

$$v'_{n[m]} \leftarrow v_{n[m]} + \sigma_a + \begin{cases} a^{\text{int}} dt, & \text{if } \zeta_{n[m]} = \zeta^{\text{yd}} \\ a^{\text{free}} dt, & \text{otherwise} \end{cases}$$

      $p'_{n[m]} = p_{n[m]} + v_{n[m]} dt$
    $x'_{n[m]} = \{p'_{n[m]}, v'_{n[m]}, \zeta_{n[m]}\}$
    $w_{n[m]} \propto \mathcal{N}([p_n, v_n]^T; [\hat{p}_n, \hat{v}_n]^T, \text{diag}[\sigma_p^2, \sigma_v^2])$
  $x'_m = \{x'_{1,m}, \ldots, x'_{N,m}\}$
  $w_m = \prod_{n=1}^{N} w_{n[m]}$
  $w_{\text{sum}} = w_{\text{sum}} + w_m$
for $m = 1, \ldots, M$ **do** ▷ normalize weights
  $w_m = w_m / w_{\text{sum}}$
$M_{\text{eff}} = \frac{1}{\sum_{i=1}^{M} (w^i)^2}$ ▷ from [24]
if $M_{\text{eff}} < M_{\text{threshold}}$ **then**
  $x' = \text{Resample}(x, w)$ ▷ from [25]
  $w' = 1/M$ ▷ reset weights

to DQN for simplicity. The DQN used in this work is inspired by the network architecture from our previous work [29], shown in Figure 2.

First, the naive approach, QPF, is trained on the entire belief state b, as shown in Figure 3. The observation o, is divided into the ego state $s_{\text{ego}} = (p_{\text{ego}}^{\text{goal}}, p_{\text{ego}}^{\text{int}}, v_{\text{ego}}, t_{\text{stop}})$ and the observation of other vehicles $o_{1:N} = \{\hat{p}_n, \hat{v}_n\}_{n=1}^{N}$.

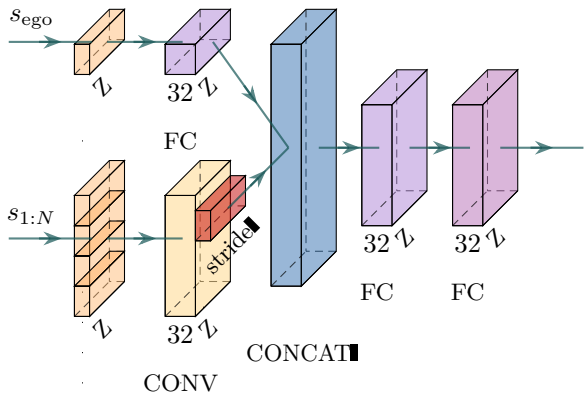


Figure 2: Common network architecture, where ego input s_{ego} represents the input features for the ego vehicle, and input of all other vehicles $s_{1:N}$. The layers consist of one convolution (Conv), three fully connected (FC), and one concatenation (CONCAT) layer. Finally, the depth of the layers Z represents the belief size. The network illustrations in this work are generated using [30]

.

The observation of the other vehicles $o_{1:N}$ is sent to the particle filter to update the belief state b and we get the updated particles x according to Algorithm 5. Both s_{ego} and x are sent to the DQN, where the number of particles M determines the depth Z of the DQN, which in turn increases the number of weights as a function of Z. The output of the DQN is then sent to a dimension-reducing layer, by calculating the mean value for each action and in turn outputs a single Q vector, denoted as Q_{PF}. As mentioned in the introduction, one challenge with this approach is that the number of particles representing the belief can be very large and this in turn requires more weights which leads to problems for training to converge.

This leads us to our first proposed approach, QID, shown in Figure 4. To reduce the input dimensions Z compared to QPF, we propose that the non-observable intention state ζ_n^{ID} is derived from the belief state using an approximator $D(b)$, while the observable states $\hat{p}_n$ and $\hat{v}_n$ are fed directly to the DQN. By doing this, the required depth of the DQN can be reduced to 1 and is no longer dependent on the number of particles M. For this method,

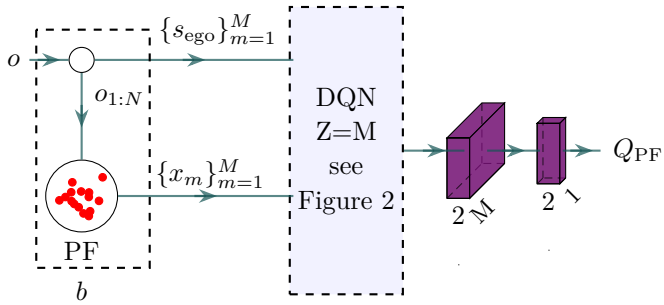


Figure 3: Block diagram of the QPF algorithm. The belief state b (D.16) consists of the observation o and the particles $\{x_m\}_{m=1}^{M}$ from the particle filer. From the belief state, the ego state s_{ego} is extracted separately from the state of the other vehicles $s_{1:N} = \{x_m\}_{m=1}^{M}$ for all cars $1:N$ and feed to the DQN, where the depth of the DQN is given by the number of particles M. After the DQN is a depth-reducing layer that reduces the depth M to 1.

$\tilde{\zeta}_n^{\text{ID}}$ is simply the marginal distribution of intention from the set of particles $\{x_m\}_{m=1}^{M}$:

$$D^{\text{ID}}(b) = \zeta_n^{\text{ID}} = \sum_{m=1}^{M} w_{n[m]}[\zeta_{\text{tw}}\ \zeta_{\text{yd}}]_{n[m]}, \tag{D.19}$$

where $w_{n[m]}$ is the weight of the particle m and $[\zeta_{\text{tw}}, \zeta_{\text{yd}}]$ is the one-hot vector specifying intention.

Another way to reduce the depth of the DQN and still consider the uncertainty of non-observable states is the approximation method QMDP [22]. Where the weights θ_{MDP} are trained in an environment with full observability. This can be used when the non-observable states can be identified offline e.g., using ground truth labeling. During evaluation time, each particle represents one belief state $b_m = \{s_{ego}, \{x_{n[m]}\}_{n=1}^{N}\}$ and is evaluated using θ_{MDP} and the approximated Q-value of the entire belief state is given by D.5

$$Q(b,a) \approx \frac{1}{M} \sum_{m} Q_{\text{MDP}}(b_m, a; \theta_{\text{MDP}}). \tag{D.20}$$

While QMDP reduces the required depth of the network, all M particles are still processed through the network and require M number of operations

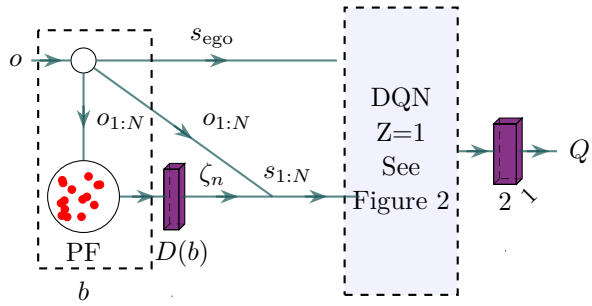


Figure 4: Our proposed approaches, QID and QMDP-IE. From an observation o, the observable states s_{ego} and $o_{1:N} = \{\hat{p}_n, \hat{v}_n\}_{n=1}^{N}$ are fed directly to the network. While an approximator $D(b)$ approximates the non-observable intention state $\hat{\zeta}_n$, so that $s_{1:N} = \{\hat{p}_n, \hat{v}_n, \hat{\zeta}_n\}_{n=1}^{N}$.

more than QID. This leads us to our final proposed approach, QMDP-IE. If we had access to θ_{MDP}, instead of averaging the Q-values of all the particles, a threshold $\zeta_{\text{threshold}}$ is set on the intention distribution from the particle filter and then use it as the true intention state. Similar to QID, the particle filter is only used to generate the intention and is represented as a one-hot vector

$$D^{\text{IE}}(b) = \hat{\zeta}_n^{\text{IE}} = \begin{cases} [0\ 1] & \text{if } \tilde{\zeta}_n^{\text{yd}} > \zeta_{\text{threshold}}, \\ [1\ 0] & \text{otherwise,} \end{cases} \tag{D.21}$$

where $\zeta_{\text{threshold}}$ is a design parameter that can be adjusted offline without any retraining of the DQN and directly correlates with the aggressiveness of the policy, e.g., a high value on $\zeta_{\text{threshold}}$ makes the agent more passive.

All four approaches are shown in Table 1 and the weights θ_{PF}, θ_{ID} and θ_{MDP} are obtained using double deep Q-learning [27], described in Algorithm 6, where step 10 is different between the algorithms. Step 12 is the simulator described in the next section while step 13 is the particle filter update described in Algorithm 5. $\mathcal{D}$ in step $14-15$, is an experience replay buffer that makes the convergence of DQN more robust [31]. The loss function in step 16 modifies (D.6) to

$$J(\theta, \theta^-) = \mathbb{E}[(r + \gamma Q(s, \underset{a}{\text{argmax}}\, Q(s, a; \theta); \theta^-)], \tag{D.22}$$

	Particles b	State Approximation $\mathrm{D}(b)$
Training weights θ	*QPF*, θ_{PF}	**QID**, θ_{ID}
Weights θ_{MDP} from ground truth	*QMDP*, θ_{MDP}	**QMDP-IE**, θ_{MDP}

Table 1: The difference between the **investigated algorithms**, while the other two are used as benchmarks. The columns show the algorithm's input to the neural network, where the first column contains the *baseline algorithms* that use the belief state b with all M particles and the second column contains algorithms that use an estimate of the intention. The first row shows the algorithms that train the weights with the given input and the second row shows algorithms that use weights trained on full observability.

where θ^- is the weight of the target network.

All algorithms are trained and evaluated on the same simulator and are described in the next section.

4 Experiments

The main goal of the experiments is to show the performance difference between the algorithms described in the previous chapter. In this section, we present the experiment setup in three parts. First, we describe how the simulator generates the different traffic scenarios. Second, the design of the reward function with some motivation to its values is described. Finally, we define the common network architecture, shown in Figure 2.

4.1 Simulator setup

Starting with the simulator, at the start of each episode, up to N vehicles are spawned with some initial values. The number of vehicles simultaneously in the other lane is the same thing as the traffic density of the lane. The initial positions p_n^0 are distributed along the intersecting lane with the furthest

Algorithm 6 double Q-learning training process

Initialize θ randomly
$\mathcal{D} \leftarrow \emptyset$
for nr episodes **do**
    $o \leftarrow$ initiate environment ▷ (D.12)
    $b \leftarrow$ InitializeBelief(o) ▷ Alg. 4
    while episode not finished **do**
        if $e \sim \mathcal{U}(0,1) < \epsilon$ **then** ▷ from [14]
            $a \leftarrow$ random action
        else
            $Q(b,a) \leftarrow$GetActionValues(b,θ) ▷ S. 3.3
            $a \leftarrow \text{argmax}_a\, Q(b,a)$ ▷ (D.4)
        $o', r \leftarrow$ StepEnvironment(a) ▷ S. 4.1
        $b' \leftarrow$ UpdateBelief$(b,\, o',\, a)$ ▷ Alg. 5
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(b,a,r,b')\}$
        $E \leftarrow$ sample from $\mathcal{D}$ ▷ from [28]
        update θ^- with SGD and loss $J(\theta,\theta^-)$ ▷ (D.22)
        for every N_{update} **do**
            update θ with θ^- ▷ from [27]

distance from the intersection being p_{lim}, a starting velocity v_n^0, and a desired velocity v_n^{desired}. Each vehicle is spawned with a deterministic policy that represents its intention ζ_n, which can either be ζ_n^{tw} *take way* or ζ_n^{yd} *yield*. These intentions follow the same transition model as the actions described in Section 3.1 for the ego vehicle and the time between simulation steps is $\text{d}t_{\text{decision}}$ seconds. The ego vehicle is spawned last with an initial speed v_{ego} and desired speed $v_{\text{ego}}^{\text{desired}}$, i.e., v_n^{desired} from (D.8).

Every time a vehicle in the perpendicular lane crosses the intersection, they are removed and a new vehicle is spawned at the start of the lane at a random time with new initial values $p_n^0, v_n^0, v_n^{\text{desired}}$ and intention ζ_n.

As mentioned in the introduction, the scenario from our previous work [29] that had the most difficulty solving was a *conflict scenario*, i.e., when another vehicle in the crossing lane has the same velocity and the same distance to the intersection. To create these conflict scenarios, the starting position of ego p_{ego}^0 is set based on the time to intersection τ_{tti} of one of the other vehicles.

This "other car" is referred to as the conflict car and is randomly chosen. This increases the probability that the ego will be in conflict with at least one other car.

Each episode is simulated until a terminal state, (i) goal, (ii) collision, (iii) safe stop, (iv) or deadlock, and a reward is given according to the reward function.

4.2 Designing the reward function

The reward function R is part of the POMDP defined in Section 3.1 and the reward for each terminal state determines the behavior of the optimal policy π^* from (D.4). According to [32], large reward values would result in large Q-values, which can cause the gradients to grow. With some trial and error, the rewards for each terminal state for this POMDP are defined as

$$r = \begin{cases} 8 & \text{reaching the goal,} \\ -10 & \text{collision,} \\ 0.4 & \text{safe stop,} \\ -0.6 & \text{deadlock,} \\ -0.01 & \text{non-terminal states.} \end{cases}$$

To get a policy that can cross the intersection when possible, a high reward of $r = 8$ is received whenever the agent reaches the goal. A high negative reward of $r = -10$ is received when the agent collides with another car to ensure that it avoids collisions.

When the agent instead chooses to yield and stop for longer than T_{stop} seconds, the received reward depends on what the other vehicle did. While the agent has stopped and the cars on the intersecting lane are crossing the intersection, the agent has made a safe stop and receives the reward $r = 0.4$. To disincentivize deadlock situations, where two cars are waiting for each other, a small negative reward is received $r = -0.6$.

Finally, to incentivize the agent to quickly reach a terminal state, a small negative reward is given for each decision step when the agent has not reached a terminal state combined with the discount factor γ.

Table 2: Hyperparameters of Simulator

decision time [s],	$\mathrm{d}t$	2
maximum initial distance [m],	p_{lim}	50
initial speed [m/s],	v_n^0	$2-7$
initial acceleration [m/s^2],	a_n^0	0
desired speed [m/s],	v_n^{desired}	$2-7$
time gap [s],	T_{gap}	1.5
Stop time limit [s],	T_{stop}	10
ego initial speed [m/s],	v_{ego}	5
ego desired speed [m/s],	$v_{\mathrm{ego}}^{\mathrm{desired}}$	5
noise position [m],	σ_{p}	2
noise velocity [m/s],	σ_{v}	1
max observed vehicles,	N	4
IDM max acceleration [m/s^2],	$\alpha^{\max}$	0.73
IDM deceleration [m/s^2],	α_b	$0.5-4.0$
IDM acceleration exponent,	δ	4
IDM minimum distance [m],	d_0	2
IDM vehicle length [m],	l_n	4
number of particles	M	100
min number of particles	$M_{\mathrm{threshold}}$	75
acceleration noise [m/s^2]	σ_a	0.1
intention switch probability [%]	ξ_i	5
intention probability threshold	$\zeta_{\mathrm{threshold}}$	0.8
Batch size	B	128
Learning rate	lr	0.0001
Discount factor	γ	0.95
Replay memory size	E_{replay}	20,000
Target network update frequency	N_{update}	1,000

5 Results and discussion

This section presents the evaluation metrics and results from the experiments, described in Section 4.1. Each algorithm is evaluated on 2000 episodes. The initial values, defined in Table 2, were randomized using the episode number

as a seed. The defined seed number ensures the same scenarios are used when comparing the different algorithms. The metrics of interest are the average time it takes for the agent to reach the success state and how often each terminal state is reached. A success state refers to either reaching the goal or a safe stop. An ideal agent would cause no collisions or deadlocks, and pass the junction with the smallest possible success time.

Another way to describe the algorithms' performance is by using aggressive and passive, which are not completely complementary to each other. An agent is considered aggressive if it has a low success time but a higher collision percentage, when compared to the Oracle DQN. An agent that does not collide often but instead ends up in deadlock more often is considered passive. So an agent that has low success time and high no collisions is neither aggressive nor passive, just a good agent. We start with analyzing how well a DQN agent that does not consider intentions, performs in scenarios with different traffic densities.

5.1 Traffic density experiment

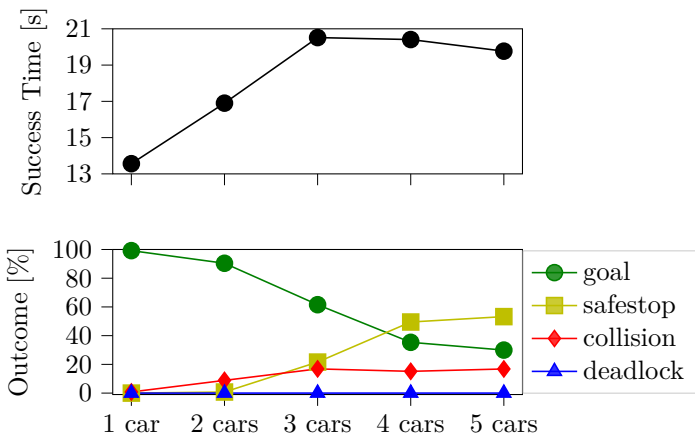


Figure 5: Performance results for DQN without intention state ζ, showing success time and outcome percentage for scenarios with increasing numbers of other cars.

Traffic density affects the problem at hand. Figure 5 shows the results from

an experiment with DQN trained agents that do not consider intentions in scenarios with different traffic densities. The 1-car scenario corresponds to traffic densities up to 1 car or vehicle per 50 m, or 20 vehicles/km; 2 cars correspond to 2 cars per 50 m, 40 vehicles/km, and so on. The results presented in Figure 5 clearly show that for low traffic densities, the problem can relatively easily be solved without needing intention estimation. The agent can solve the problem without collisions, deadlocks, or safe stops. As the traffic density increases, the performance significantly drops. For traffic densities larger than 60 vehicles/km (4 cars), the agent only reaches the goal in approximately 35 % of the cases, makes safe stops before the intersection in 50 % of the cases, and collides in 15 %, which makes them the scenarios that can be most improved upon with algorithms that consider intentions. This experiment also gives a sense of the limit of how a DQN can perform when not considering the intention state ζ.

As traffic densities in OECD countries according to [33] is around 60 vehicles/km, the rest of the experiments will focus on the 4 cars scenario.

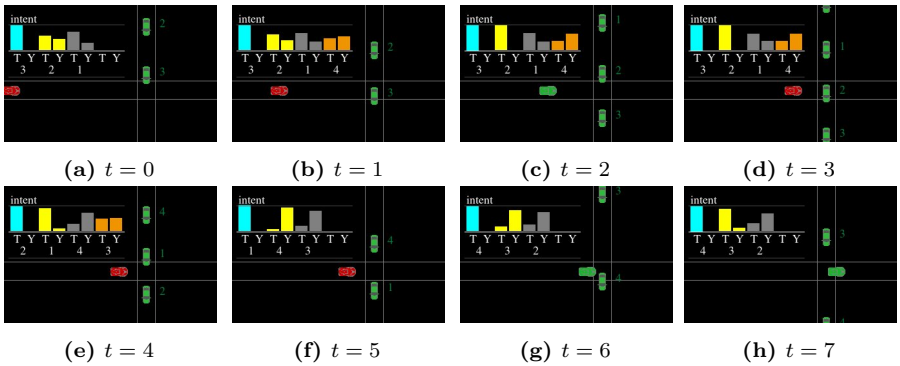
Table 3: Result summary for 4 cars scenario

Experiments	Goal reached %	Safe stop %	Collision %	Deadlock %	Success time s	Training time h
Oracle DQN	84.50	14.45	1.05	0.00	15.49	31h
QMDP-IE	**80.00**	**18.15**	**1.35**	**0.50**	**17.14**	**N/A**
QMDP	71.95	15.05	5.70	7.30	20.56	N/A
QID	**85.60**	**10.40**	**3.70**	**0.30**	**16.64**	**119**h
QPF	63.50	21.80	12.15	2.55	16.61	124h

5.2 Probability distribution of the intention state

The particle filter generates the probability distribution of the non-observable intention state ζ, which can be used to make the algorithms from Section 3.3 possible to implement.

In Figure 6, an example of a QMDP agent with the intention distribution from the particle filter is shown. Looking at the first car with id 3 at $t = 0 - 3$, we can observe that the particle filter correctly predicts the take way intention



(a) $t = 0$ **(b)** $t = 1$ **(c)** $t = 2$ **(d)** $t = 3$

(e) $t = 4$ **(f)** $t = 5$ **(g)** $t = 6$ **(h)** $t = 7$

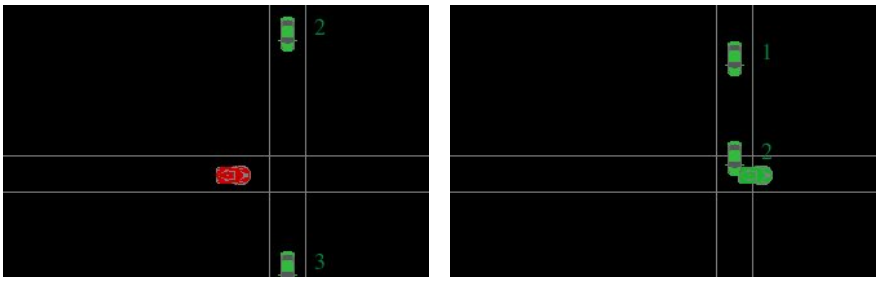
Figure 6: Example with a QMDP agent. The numbers on the right of the vertical cars are the cars id number n. The intention bar figure in each image shows the intention distribution for each other car. Each car has two bars grouped by the same color and the car id number, starting with the take way probability on the left and marked by a "T" and yield probability on the right marked with a "Y".

and has some uncertainty about the intention for car id 2 and 1 crossed the intersection. It is also interesting to observe car id 4, as it temporarily has a higher probability of yielding when following car id 1 for $t = 1 - 5$, but could quickly switch to take way once car 1 crossed the intersection at $t = 6$.

The example from Figure 6 shows that the proposed particle filter performs well at estimating the probability distribution of the intention. However, it relies on an accurate prediction model and could make some mistakes. Because we want to investigate if the approximation of the DQN could compensate for the noisy states of the particles and utilize the change of intention probability between states, the focus in this paper is not to make the particle filter prediction as correct as possible. It is possible to improve the estimate by increasing the number of particles at the expense of additional computation.

5.3 Oracle DQN

The results from the previous experiment gave us a sense of the lower limit of a DQN when not considering the intention state ζ. To give a sense of an upper limit, we trained a DQN agent on the true state (D.11), this is refereed to as the Oracle DQN. Table 3 shows results of the oracle DQN and all four other



(a) Ego car on the horizontal lane taking the yield action waiting for an opportunity to cross.

(b) Ego taking the take way action and colliding with another car on the vertical lane.

Figure 7: Example scenario of an Oracle DQN agent trying to drive through a gap between cars. The agent is on the horizontal lane and is red when choosing the yield action and green when choosing the take way action.

agents described in Section 3.3. Looking at the results, we see that even with full observability the oracle DQN has 1.05 % collisions. This shows that there are still some corner cases that the oracle DQN agent had difficulty navigating. One such case is shown in Figure 7, where the agent found a possible gap between cars in the picture on the left, but was hit on the tail end of ego in the figure to the right.

5.4 QMDP-IE and QMDP results

Starting with QMDP-IE, we first demonstrate how the $\zeta_{\text{threshold}}$ is chosen and then compare its results with QMDP. Both algorithms use the same network weights as the Oracle DQN, but QMPD-IE is less computational than QMDP since the threshold $\zeta_{\text{threshold}}$ transforms the set of particles into a point estimate of the intention state, as described in Section 3.3. Therefore, the choice of $\zeta_{\text{threshold}}$ is an important hyperparameter that directly correlates with the aggressiveness of the agent.

The threshold value $\zeta_{\text{threshold}}$, used in QMDP-IE, is determined by evaluating the agent with different $\zeta_{\text{threshold}}$ and observing the outcome percentage and success time. The results are shown in Figure 8. The lowest threshold $\zeta_{\text{threshold}} = 0.5$ is equivalent to just taking the most likely estimation of the intention and resulting in an aggressive agent. At the same time, threshold

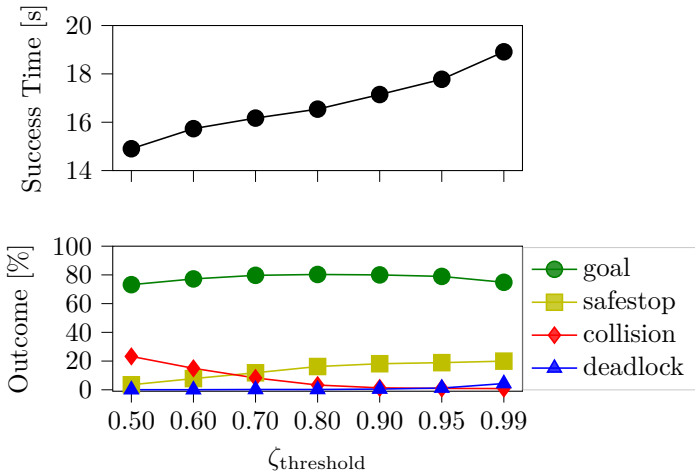


Figure 8: QMDP-IE performance (aggressiveness) for different $\zeta_{\text{threshold}}$ values. Increasing $\zeta_{\text{threshold}}$ lowers the collision rate while at the same time increasing the safe stop rate and the time it takes to reach a success state.

values that are too large result in a passive agent. In summary, the parameter $\zeta_{\text{threshold}}$ directly correlates with the agent's aggressiveness, and a good trade-off between aggressiveness and passiveness is around $\zeta_{\text{threshold}} = 0.9$. With $\zeta_{\text{threshold}} = 0.9$, QMDP-IE performs significantly better on most evaluation metrics than QMDP, see Table 3. Moreover, QMDP-IE performs almost the same as the Oracle DQN, particularly regarding collision avoidance and success time. The method is slightly more passive than the Oracle DQN but less passive than QMDP. A feature with QMDP-IE is that the aggressiveness/passiveness can be adjusted using threshold $\zeta_{\text{threshold}}$ and can be changed without re-training the network weights. However, the network weights need to be trained using the true states.

5.5 QPF and QID results

As opposed to QMDP-IE and QMDP, QID and QPF algorithms do not require access to the true intent during training time. While QPF trains a network using all particles, QID only uses the intention distribution from the particles,

which reduces the size of the network while keeping information about the uncertainty of the intention state. Looking at Table 3, QPF has the highest collision rate at 12.15 % and the second lowest success time out of all considered algorithms, making it the most aggressive policy out of all four algorithms. QID collided more than QMDP-IE but still less than QMDP. This shows that training on ground truth is preferred, but QID is a good algorithm to use when ground truth data is not available. One downside is that QID and QPF took much longer time to train than the Oracle DQN, with up to 5 days of training. Since QID and QPF have comparable training times, this hints that the particle filter used by both algorithms is the main explanation for the computational time and that the large state space of QPF is of minor importance.

5.6 Overtake agent

Table 4: Results with an overtake agent

Experiments	Goal reached %	Safe stop %	Collision %	Deadlock %	Success time s
Oracle DQN	89.80	0.15	10.05	0.00	16.01
QMDP-IE	**94.55**	**0.20**	**4.60**	**0.65**	**16.62**
QMDP	79.70	10.20	5.95	4.15	19.84
QID	**96.89**	**0.11**	**2.53**	**0.47**	**16.38**
QPF	96.15	0.10	3.70	0.05	17.62

To investigate how the algorithms perform when exposed to a behavior they are not trained for, we introduce a new scenario where the conflict car is allowed to overtake the car in front. This is done by only using the free road term (D.10) for the conflict car even when there are other cars in front. The overtake scenario aims to demonstrate how adaptable the agent is to situations outside of the training set. All algorithms are evaluated on the new overtake scenario without any retraining. The performance for all four algorithms, including the Oracle DQN, is summarized in Table 4. The collision rate for the Oracle DQN is the highest with 10.05 %, this is expected since the method is trained on intention states ζ_n outside of its training set.

In the overtake scenario, both QID and QPF have relatively low collision rates at 2.53 % and 3.7 % respectively, which is lower when compared to the trained scenarios in Table 3. While the collision rate for QMDP-IE and QMDP is 4.6 % and 5.95 % respectively, which is higher than what they got on the trained scenario but it is also higher than QID and QPF. This indicates that QID and QPF are better at handling some scenarios outside the training set than QMDP-IE and QMDP. QMDP also has the highest safe stop and deadlock rate at 10.2 % and 4.15 %, making it the most passive policy.

5.7 Discussion

When comparing QMDP-IE with QID, it is observed that the algorithms trained on ground truth (QMDP-IE) outperform the algorithms that were trained using the belief state (QPF) or even just an estimate of the belief distribution (QID).

The particle filter was used to create the belief state and filter the noisy observations. Hoping the flexibility of the neural network would be able to compensate for some of the inaccuracy when estimating the distribution. The higher collision rate for the QPF compared to the Oracle DQN shows that training on the full belief state can make finding a good policy difficult.

Figure 8 shows that the aggressiveness of the policy from QMDP-IE is correlated with $\zeta_{\text{threshold}}$ and by choosing a relatively high value, it could get a collision rate close to the Oracle DQN, and in this case could, to some extent, compensate for the not fully optimized particle filter. The ability to adjust the aggressiveness of the policy by changing $\zeta_{\text{threshold}}$ is also a strength of QMDP-IE compared to QMDP and our previous black box methods that use an LSTM to estimate the latent state [13]. The downside with QMDP-IE is that, in practice, they require the true intention during training, this can be obtained by either human labeling or auto labeling techniques, which can be either expensive or tedious to generate.

The other agents' possible actions were simplified to take way or yield. These actions could be expanded to left turn, right turn, or accelerate. In the road segment leading to the intersecting point, these actions would mainly affect the velocity profile of these cars coming into the intersection and slightly limit how fast ego can accelerate after a car that has turned into its lane. Looking at the overtake scenario, we can assume that these actions would not perform worse than using an overtake agent, shown in Table 4.

6 Conclusion

In this study, we explored the application of reinforcement learning for safe intersection navigation, emphasizing the importance of accounting for uncertainty and planning over the distribution of intentions. Specifically, we proposed two methods for representing belief over drivers' intentions using a particle-based approach and a compact parametric representation, which can manifest as a probability distribution (QID) or a point estimate (QMDP-IE).

Our proposed particle filter exhibited promising results in estimating intention probabilities. We leveraged these intention probability estimates in QMDP-IE and QID algorithms, which demonstrated superior performance compared to respective baseline approaches QMDP and QPF. QMDP-IE, in particular, offered adjustable aggressiveness post-training, yielding a collision rate close to the Oracle DQN with an optimal threshold value $\zeta_{\text{threshold}}$. Furthermore, QID policies collision rate was comparable to QMDP-IE for conflict scenarios. However, in an additional experiment involving a scenario with overtaking maneuvers, all algorithms exhibited higher collision rates compared to the standard scenario. Interestingly, QID and QPF demonstrated relatively lower collision rates, indicating adaptability to unforeseen behaviors, while QMDP displayed a more passive approach. A downside to QID and QPF is that the training times were considerably longer, highlighting the computational demands of the particle filter.

Overall, our findings underline the importance of accounting for uncertain intentions in autonomous driving scenarios and highlight the effectiveness of leveraging intention prediction for informed decision-making. Future research could improve the computational demand from the particle filter by focusing on network structure that can better learn the intention state and compare the results with the QMDP-IE and QID.

References

[1] K. Heineke, N. Laverty, T. Möller, and F. Ziegler, *The future of mobility: Mobility evolves*, Apr. 2023.

[2] 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.

[3] L. Fletcher *et al.*, "The mit–cornell collision and why it happened," *Journal of Field Robotics*, vol. 25, pp. 775–807, Oct. 2008.

[4] S. Kammel *et al.*, "Team annieway's autonomous system for the 2007 darpa urban challenge," *Journal of Field Robotics*, vol. 25, pp. 615–639, Sep. 2008.

[5] L. Gressenbuch and M. Althoff, "Predictive monitoring of traffic rules," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2021.

[6] "2019 traffic safety culture index," AAA Foundation for Traffic Safety, Washington DC, Tech. Rep. 202-638-5944, 2019.

[7] C. Hubmann, M. Becker, D. Althoff, D. Lenz, and C. Stiller, "Decision making for autonomous driving considering interaction and uncertain prediction of surrounding vehicles," in *IEEE Intelligent Vehicles Symposium (IV)*, 2017.

[8] 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.

[9] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.

[10] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, "Navigating occluded intersections with autonomous vehicles using deep reinforcement learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[11] M. J. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2015.

[12] P. Zhu, X. Li, P. Poupart, and G. Miao, *On improving deep reinforcement learning for pomdps*, 2018.

[13] 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.

[14] V. Mnih *et al.*, *Playing atari with deep reinforcement learning*, 2013.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[16] S. B. Amsalu, A. Homaifar, A. Karimoddini, and A. Kurt, "Driver intention estimation via discrete hidden markov model," in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2017.

[17] M. Liebner, M. Baumann, F. Klanner, and C. Stiller, "Driver intent inference at urban intersections using the intelligent driver model," in *IEEE Intelligent Vehicles Symposium (IV)*, 2012.

[18] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical Review E*, vol. 62, pp. 1805–1824, 2 2000.

[19] S. Hoermann, D. Stumper, and K. Dietmayer, "Probabilistic long-term prediction for autonomous vehicles," in *IEEE Intelligent Vehicles Symposium (IV)*, 2017.

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

[21] A. Wang, A. C. Li, T. Q. Klassen, R. T. Icarte, and S. A. Mcilraith, "Learning belief representations for partially observable deep RL," vol. 202, Jul. 2023.

[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] A. Kong, J. S. Liu, and W. H. Wong, "Sequential imputations and bayesian missing data problems," *Journal of the American Statistical Association*, vol. 89, no. 425, pp. 278–288, 1994.

[25] "Novel approach to nonlinear/non-gaussian bayesian state estimation," vol. 140, no. 2, pp. 107–113, 1993.

[26] M. Hessel *et al.*, "Rainbow: Combining improvements in deep reinforcement learning," *AAAI Conference on Artificial Intelligence (AAAI)*, vol. 32, no. 1, Apr. 2018.

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

[28] L.-J. Lin, "Reinforcement learning for robots using neural networks," UMI Order No. GAX93-22750, Ph.D. dissertation, USA, 1992.

[29] T. Tram, I. Batkovic, M. Ali, and J. Sjöberg, "Learning when to drive in intersections by combining reinforcement learning and model predictive control," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, Oct. 2019.

[30] H. Iqbal, *Harisiqbal88/plotneuralnet v1.0.0*, version v1.0.0, Dec. 2018.

[31] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[32] H. van Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver, "Learning values across many orders of magnitude," in *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[33] OECD, *Road traffic, vehicles and networks*. 2015.

PAPER E

Reinforcement Learning in the Wild with Maximum Likelihood-based Model Transfer

Hannes Eriksson, Tommy Tram, Debabrota Basu, Mina Alibeigi, and Christos Dimitrakakis

Published in 2024 International Conference on Adaptive Agents and Multi-Agent Systems (AAMAS),
pp. 516–524, May. 2024.

The layout has been revised.

Abstract

In this paper, we study the problem of transferring the available Markov Decision Process (MDP) models to learn and plan efficiently in an unknown but similar MDP. We refer to it as *Model Transfer Reinforcement Learning (MTRL)* problem. First, we formulate MTRL for discrete MDPs and Linear Quadratic Regulators (LQRs) with continuous state actions. Then, we propose a generic two-stage algorithm, MLEMTRL, to address the MTRL problem in discrete and continuous settings. In the first stage, MLEMTRL uses a *constrained Maximum Likelihood Estimation (MLE)*-based approach to estimate the target MDP model using a set of known MDP models. In the second stage, using the estimated target MDP model, MLEMTRL deploys a model-based planning algorithm appropriate for the MDP class. Theoretically, we prove worst-case regret bounds for MLEMTRL both in realisable and non-realisable settings. We empirically demonstrate that MLEMTRL allows faster learning in new MDPs than learning from scratch and achieves near-optimal performance depending on the similarity of the available MDPs and the target MDP.

1 Introduction

Deploying autonomous agents in the real world poses a wide variety of challenges. As in [1], we are often required to learn the real-world model with limited data, and use it to plan to achieve satisfactory performance in the real world. There might also be safety and reproducibility constraints, which require us to track a model of the real-world environment [2]. In light of these challenges, we attempt to construct a framework that can aptly deal with optimal decision making for a novel task, by leveraging external knowledge. As the novel task is unknown, we adopt the Reinforcement Learning (RL) [3] framework to guide an agent's learning process and to achieve near-optimal decisions.

An RL agent interacts directly with the environment to improve its performance. Specifically, in model-based RL, the agent tries to learn a model of the

environment and then use it to improve performance [4]. In many applications, the depreciation in performance due to sub-optimal model learning can be paramount. For example, if the agent interacts with living things or expensive equipment, decision-making with an imprecise model might incur significant cost [5]. In such instances, boosting the model learning by leveraging external knowledge from the existing models, such as simulators [6], physics-driven engines, etc., can be of great value [7]. A model trained on simulated data may perform reasonably well when deployed in a new environment, given the novel environment is *similar enough* to the simulated model. Also, RL algorithms running on different environments yield data and models that can be used to plan in another similar enough real-life environment. In this work, we study the problem where we have access to multiple source models built using simulators or data from other environments, and we want to transfer the source models to perform efficient model-based RL in a different real-life environment.

Let us consider that a company is designing autonomous driving agents for different countries in the world. The company has designed two RL agents that have learned to drive well in USA and UK. Now, the company wants to deploy a new RL agent in India. Though all the RL agents are concerned with the same task, i.e. driving, the models encompassing driver behaviors, traffic rules, signs, etc., can differ for each. For example, UK and India have left-handed traffic, while the USA has right-handed traffic. However, learning a new controller *specifically* for every new geographic location is computationally expensive and time-consuming, as both data collection and learning take time. Thus, the company might use the models learned for UK and USA, to estimate the model for India, and use it further to build a new autonomous driving agent (RL agent). Hence, being able to transfer the source models to the target environment allows the company to use existing knowledge to build an efficient agent faster and resource efficiently.

We address this problem of model transfer from source models to a target environment to plan efficiently. We observe that this problem falls at the juncture of *transfer learning* and *reinforcement learning* [8]–[10]. [9] enlists three approaches to transfer knowledge from the *source tasks* to a *target task*. (i) *Instance transfer:* data from the source tasks is used to guide decision-making in the novel task [7]. (ii) *Representation transfer:* a representation of the task, such as learned neural network features, are transferred to perform

the new task [11]. (iii) *Parameter transfer:* the parameters of the RL algorithm or *policy* are transferred [12]. In our paper, the source tasks are equivalent to the source models, and the target task is the target environment. Moreover, we adopt the **model transfer reinforcement learning** approach (MTRL), which encompasses both (i) and (ii) (Section 4).

[13] describes three possible benefits of transfer learning. The first is **learning speed improvement**, i.e. decreasing the amount of data required to learn the solution. Secondly, **asymptotic improvement**, where the solution results in better asymptotic performance. Lastly, **jumpstart improvement**, where the initial proxy model serves as a better starting solution than that of one learning the true model from scratch. In this work, we propose a new algorithm to transfer RL that achieves both learning speed improvement and jumpstart improvement (Section 8). However, we might not find an asymptotic improvement in performance if compared with the best and unbiased algorithm in the true setting. Rather, we aim to achieve a model estimate that allows us to plan accurately in the target MDP (Section 6).

Contributions. We aim to answer two central questions:

1. *How can we accurately construct a model using a set of source models for an RL agent deployed in the wild?*

2. *Does the constructed model allow efficient planning and yield improvements over learning from scratch?*

In this paper, we address these questions as follows:

1. *A Taxonomy of MTRL:* First, we formulate the problem with the Markov Decision Processes (MDPs). We further provide a taxonomy of the problem depending on a discrete or continuous set of source models, and whether the target model is realisable by the source models (Section 4).

2. *Algorithm Design with MLE:* Following that, we design a two-stage algorithm MLEMTRL to plan in an unknown target MDP (Section 5). In the first stage, MLEMTRL uses a Maximum Likelihood Estimation (MLE) approach to estimate the target MDP using the source MDPs. In the second stage, MLEMTRL uses the estimated model to perform model-based planning. We instantiate MLEMTRL for discrete state-action (tabular) MDPs and Linear Quadratic Regulators (LQRs). We also derive a generic bound on the goodness of the policy computed using MLEMTRL (Section 6). We further provide

a meta-algorithm, called Meta-MLEMTRL, to control the adaptation to the non-realisable setting (Section 7).

3. *Performance Analysis:* In Section 8, we empirically verify whether MLEMTRL improves the performance for unknown tabular MDPs and LQRs than learning from scratch. MLEMTRL exhibits learning speed improvement for tabular MDPs and LQRs. For LQRs, it incurs learning speed improvement and asymptotic improvement. We also observe that the more similar the target and source models are, the better the performance of MLEMTRL, as indicated by the theoretical analysis. An ablation study of Meta-MLEMTRL under realisable and non-realisable settings further shows provable improvements yielded in the asymptotic and non-realisable regimes.

Before elaborating on the contributions, we position this work in the existing literature (Section 2) and discuss the background knowledge of MDPs and MLEs (Section 3).

2 Related Work

Our work on Model Transfer Reinforcement Learning is situated in the field of Transfer RL (TRL) and also is closely related to the multi-task RL and Bayesian multi-task RL literature. In this section, we elaborate on these connections.

TRL is widely studied in Deep Reinforcement Learning. [14] introduces different ways of transferring knowledge, such as *policy transfer*, where the set of source MDPs $\mathcal{M}_s$ has a set of expert policies associated with them. The expert policies are used together with a new policy for the novel task by transferring knowledge from each policy. [12] uses this approach, where a student learner is combined with a set of teacher networks to guide learning in multi-task RL. [15] develops an actor-critic structure to learn ways to transfer its knowledge to new domains. [16] invokes generalisation across Q-functions by learning a master policy. Here, *we focus on model transfer instead of policy.*

Another seminal work in TRL, by [8] distinguishes between *multi-task learning* and *transfer learning.* Multi-task learning deals with problems where the agent aims to learn from a distribution over scenarios, whereas transfer learning makes no specific assumptions about the source and target tasks. Thus, in transfer learning, the tasks could involve different state and action spaces and different transition dynamics. Specifically, we focus on **model-transfer** [17]

approach to TRL, where the state-action spaces and also dynamics can be different. [17] performs model transfer for a target task with an identical transition model. Thus, the main consideration is to transfer knowledge to tasks with the same dynamics but varying rewards. [10] assumes a context similar to that of [17], where the model dynamics are identical across environments. In our work, we rather assume that the reward function is the same, but the transition models are different. We believe this is an interesting question as the harder part of learning an MDP is learning the transition model. These works explicate a deep connection between the fields of *multi-task learning* and *TRL*. In general, TRL can be viewed as an extension of multi-task RL, where multiple tasks can either be learned simultaneously or have been learned *a priori*. This flexibility allows us to learn even in settings where the state-actions and transition dynamics are different among tasks. [18] describes a multi-task Maximum Likelihood Estimation procedure for optimal control of an aircraft. They identify a mixture of Gaussians, where the mixture is over each of the tasks. Here, we adopt an MLE approach to TRL to optimise performance for the target MDP (or a target task) rather than restricting to a mixture of Gaussians.

The Bayesian approach to multi-task RL [19], [20] tackles the problem of learning jointly how to act in multiple environments. [20] handles the *open-world assumption*, i.e. the number of tasks is unknown. This allows them to transfer knowledge from existing tasks to a novel task, using value function transfer. However, this is significantly different from our setting, as we are considering model-based transfer. Further, *we adopt an MLE-based framework in lieu of the full Bayesian procedure described in their work.* In Bayesian RL, [21] also investigates a learning technique to generalise over multiple problem instances. By sampling a large number of instances, the method is expected to learn how to generalise from the existing tasks to a novel task. We do not assume access to such prior or posterior distributions to sample from.

There is another related line of work, namely multi-agent transfer RL [22]. For example, [23] develops a TRL framework for autonomous driving using federated learning. They accomplish this by aggregating knowledge for independent agents. This setting is different from general transfer learning but could be incorporated if the source tasks are learned simultaneously with the target task. This requires cooperation among agents, which is out of the scope.

3 Background

In this section, we introduce the important concepts on which this work is based upon. Firstly, we introduce the way we model the dynamics of the tasks. Secondly, we describe the Maximum Likelihood Estimation (MLE) framework used in this work.

3.1 Markov Decision Process

We study sequential decision-making problems that can be represented as Markov Decision Processes (MDPs) [24]. An MDP μ consists of a discrete or continuous state space denoted by $\mathcal{S}$, a discrete or continuous action-space $\mathcal{A}$, a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ which determines the quality of taking action a in state s, and a transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta\mathcal{S}$ inducing a probability distribution over the successor states s' given a current state s and action a. Finally, in the infinite-horizon formulation, a discount factor $\gamma \in [0, 1)$ is assigned. The overarching objective for the agent is to compute a decision-making policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ that maximises the expected sum of future discounted rewards up until the horizon T: $V_\mu^\pi(s) = \mathbb{E}\Big[\sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, a_t)\Big]$. $V_\mu^\pi(s)$ is called the value function of policy π for MDP μ. The optimal value function is denoted as μ is $V_\mu^* = V_\mu^{\pi^*}$. The technique used to obtain the the optimal policy $\pi^* = \sup_\pi V_\mu^\pi$ depends on the MDP class. The MDPs with discrete state-action spaces are referred to as tabular MDPs. In this paper, we also study a specific class of MDPs with continuous state-action spaces, namely the Linear Quadratic Regulators (LQRs) [25]. In tabular MDPs, we employ ValueIteration [24] for model-based planning whereas in the LQR setting we use RiccatiIteration [26].

The standard metric used to measure the performance of a policy π [27] for an MDP μ is *regret* $R(\mu, \pi)$. Regret is the difference between the optimal value function and the value function of π. In this work, we extend the definition of regret for MTRL, where the optimality is taken with respect to a policy class in the target MDP.

3.2 Maximum Likelihood Estimation

One of the most popular methods of constructing point estimators is the *Maximum Likelihood Estimation* (MLE) [28] framework. Given a density

function $f(x \mid \theta_1, \ldots, \theta_n)$ and associated i.i.d. data $X_1, \ldots, X_t$, the goal of the MLE scheme is to maximise,

$$\ell(\theta \mid x) \triangleq \ell(\theta_1, \ldots, \theta_n \mid x_1, \ldots, x_t) \triangleq \log \prod_{i=1}^{t} f(x_i \mid \theta_1, \ldots, \theta_n). \tag{E.1}$$

$\ell(\cdot)$ is called the log-likelihood function. The set of parameters maximising Equation E.1 is called the *maximum likelihood estimator* of θ given the data $X_1, \ldots, X_t$. Maximum likelihood estimation has many desirable properties that we leverage in this work. For example, the ML estimator satisfies *consistency*, i.e. under certain conditions, it achieves optimality even for *constrained* MLE. An estimator being consistent means that if the data $X_1, \ldots, X_t$ is generated by $f(\cdot \mid \theta)$ and as $t \to \infty$, the estimate almost surely converges to the true parameter θ. [29] shows that MLE admits the consistency property given the following assumptions hold. The model is *identifiable*, i.e. the densities at two parameter values must be different unless the two parameter values are identical. Further, the parameter space is *compact* and *continuous*. Finally, if the log-density is *dominated*, one can establish that MLE converges to the true parameter almost surely [30]. For problems where the likelihood is unbounded, flat or otherwise unstable one may introduce a penalty term in the objective function. This approach is called *penalised maximum likelihood estimation* [31], [32]. As we in our work are mixing over known parameters, we do not need to add regularisation to our objective to guarantee convergence.

In this work, we iteratively collect data and compute new point estimates of the parameters and use them in our decision-making procedure. In order to carry out MLE, a likelihood function has to be chosen. In this work, we investigate two such likelihood functions in Section 5, one for each respective model class.

4 A Taxonomy of Model Transfer RL

Now, we formally define the Model Transfer RL problem and derive a taxonomy of settings encountered in MTRL.

4.1 MTRL: Problem Formulation

Let us assume that we have access to a set of source MDPs $\mathcal{M}_s \triangleq \{\mu_i\}_{i=1}^m$. The individual MDPs can belong to a finite or infinite but compact set depending on the setting. For example, for tabular MDPs with finite state-actions, this is always a finite set. Whereas for MDPs with continuous state-actions, the transitions can be parameterised by real-valued vectors/matrices, corresponding to an infinite but compact set. Given access to $\mathcal{M}_s$, we want to find an optimal policy for an unknown target MDP μ^* that we encounter while deploying RL in the wild. At each step t, we use $\mathcal{M}_s$ and the data observed from the target MDP $D_{t-1} \triangleq \{s_0, a_0, s_1, \ldots, s_{t-1}, a_{t-1}, s_t\}$ to construct an estimate of μ^*, say $\hat{\mu}^t$. Now, we use $\hat{\mu}^t$ to run a model-based planner, such as ValueIteration or RiccatiIteration, that leads to a policy π^t. After completing this planning step, we interact with the target MDP using π_t that yields an action a_t, and leads to observing s_{t+1}, r_{t+1}. We update the dataset with these observations: $D_t \triangleq D_{t-1} \cup \{a_t, s_t\}$. Here, we assume that all the source and target MDPs share the same reward function $\mathcal{R}$. We do not restrict the state-action space of target and source MDPs.

Our goal is to compute a policy π^t that performs as close as possible with respect to the optimal policy π^* for the target MDP as the number of interactions with the target MDP $t \to \infty$. This allows us to define a notion of regret for MTRL: $R(\mu^*, \pi_t) \triangleq V_{\mu^*}^* - V_{\mu^*}^{\pi_t}$. Here, π_t is a function of the source models $\mathcal{M}_s$, the data collected from target MDP D_t, and the underlying MTRL algorithm. The goal of an MTRL algorithm is to minimise $R(\mu^*, \pi_t)$. For the parametric policies π_θ with $\theta \in \Theta \subset \mathbb{R}^d$, we can specialise the regret further for this parametric family: $R(\mu^*, \pi_{\theta_t}) = V_{\mu^*}^{\pi_{\theta^*}} - V_{\mu^*}^{\pi_{\theta_t}}$. For example, for LQRs, we by default work with linear policies. We use this notion of regret in our theoretical and experimental analysis.

4.2 Three Classes of MTRL Problems

We begin by illustrating MTRL using Figure 1. In the figure, the source MDPs $\mathcal{M}_s$ are depicted in red. This green area is the convex hull spanned by the source models $\mathcal{C}(\mathcal{M}_s)$. The target MDP μ^*, the best representative within the convex hull of the source models μ, and the estimated MDP $\hat{\mu}$ are shown in blue, yellow, and purple, respectively. If the target model is *inside* the convex hull, we call it a **realisable** setting. whereas If the target model is outside (as

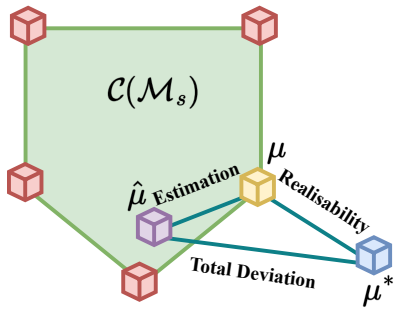


Figure 1: An illustration of the MTRL setting. The source models $\mathcal{M}_s$ are the red boxes. The green area is the convex hull $\mathcal{C}(\mathcal{M}_s)$ spanned by the source models. The target MDP μ^* is displayed in blue, and the best proxy model is contained in the convex hull μ in yellow. Finally, the estimator of the best proxy model $\hat{\mu}$ is shown in purple.

in Figure 1), then we have a **non-realisable** setting.

Figure 1 also shows that the total deviation of the estimated model from the target model depends on two sources of errors: (i) realisability, i.e. how far is the target MDP μ^* from the convex hull of the source models $\mathcal{C}(\mathcal{M}_s)$ available to us, and (ii) estimation, i.e. how close is the estimated MDP $\hat{\mu}$ to the best possible representation μ of the target MDP. In the realisable case, the realisability gap can be reduced to zero, but not otherwise. This approach allows us to decouple the effect of the expressibility of the source models and the goodness of the estimator.

Now, we further elaborate on these three classes and the corresponding implications of performing MLE.

I. Finite and Realisable Plausible Models. If the true model μ^* is one of the target models, i.e. $\hat{\mu} \in \mathcal{M}_s$, we have to identify the target MDP from a finite set of plausible MDPs. Thus, the corresponding MLE involves a finite set of parameters, i.e. the parameters of the source MDPs $\mathcal{M}_s$. We compute the MLE $\hat{\mu}$ by solving the optimisation problem:

$$\hat{\mu} \in \arg\max_{\mu' \in \mathcal{M}_s} \log \mathbb{P}(D_t \,|\, \mu'), D_t \sim \mu^*. \tag{E.2}$$

This method may serve as a reasonable heuristic for the TRL problem, where the target MDP is the same as or reasonably close to one of the source MDPs. However, this method will potentially be sub-optimal if the target MDP is too different from the source MDPs. Even if μ^* lies within the convex hull of the

source MDPs (the green area in Figure 1), this setting restricts the selection of a model to one of the red boxes. Thus, this setting fails to leverage the expressiveness of the source models as MLE allows us to accurately estimate models which are also in $\mathcal{C}(\mathcal{M}_s)$. Thus, we focus on the two settings described below.

II. Infinite and Realisable Plausible Models. In this setting, the target MDP μ^* is in the convex hull $\mu^* \in \mathcal{C}(\mathcal{M}_s)$ of the source MDPs. Thus, for Class I, we extend the parameter space considered in MLE to an infinite but compact parameter set.

Let us define the convex hull as $\mathcal{C}(\mathcal{M}_s) \triangleq \{\mu_1 w_1 + \ldots + \mu_m w_m \,|\, \mu_i \in \mathcal{M}_s, w_i \geq 0, i = 1, \ldots, m, \sum_{i=1}^{m} w_i = 1\}$. Then, the corresponding MLE problem with the corresponding likelihood function is

$$\hat{\mu} \in \underset{\mu' \in \mathcal{C}(\mathcal{M}_s)}{\arg\max} \; \log \mathbb{P}(D_t \,|\, \mu'), D_t \sim \mu^*. \tag{E.3}$$

Since $\mathcal{C}(\mathcal{M}_s)$ induces a compact subset of model parameters $\mathcal{M}' \subset \mathcal{M}$, Equation (E.3) leads to a *constrained maximum likelihood estimation problem* [33]. It implies that if the parameter corresponding to the target MDP is in $\mathcal{M}'$, it can be correctly identified. In the case where the optimum lies inside, we can use constrained MLE to accurately identify the true parameters given enough experience from μ^*. This approach allows us to leverage the expressibility of the source models completely. However, μ^* might lie outside or on the boundary. Either of them may pose problems for the optimiser.

III. Infinite and Non-realisable Plausible Models. This class is similar to Class II with the important difference that the true parameter μ^* is outside the convex hull of source MDPs $\mathcal{C}(\mathcal{M}_s)$, and thus, the corresponding parameter is not in the induced parameter subset $\mathcal{M}'$. This key difference means the true parameters cannot be correctly identified. Instead, the objective is to identify the best proxy model $\mu \in \mathcal{M}'$. The performance loss for using μ instead of μ^* is intimately related to the model dissimilarity $||\mu^* - \mu||_1$. This allows us to describe the limitation of expressivity of the source models by defining the *realisability gap*: $\epsilon_{\text{Realise}} \triangleq \min_{\mu \in \mathcal{C}(\mathcal{M}_s)} \|\mu^* - \mu\|_1$. The realisability gap becomes important while dealing with continuous state-action MDPs with parameterised dynamics, such as LQRs.

Algorithm 7 Maximum Likelihood Estimation for Model-based Transfer Reinforcement Learning (MLEMTRL)

Input: weights $\boldsymbol{w}^0$, m source MDPs $\mathcal{M}_s$, data D_0, discount factor γ, iterations T.
for $t = 0, \ldots, T$ **do**
    // Stage 1: Model Estimation //
    $\boldsymbol{w}^{t+1} \leftarrow \text{Optimiser}(\log \mathbb{P}(D_t \mid \Sigma_{i=1}^m w_i \mu_i), \boldsymbol{w}^t)$
    Estimate the MDP: $\mu^{t+1} = \sum_{i=1}^m w_i \mu_i$
    // Stage 2: Model-based Planning //
    Compute the policy: $\pi^{t+1} \in \arg\max_\pi V^\pi_{\mu^{t+1}}$
    // Control //
    Observe $s_{t+1}, r_{t+1} \sim \mu^*(s_t, a_t), a_t \sim \pi^{t+1}(s_t)$
    Update the dataset $D_{t+1} = D_t \cup \{s_t, a_t, s_{t+1}, r_{t+1}\}$
return An estimated MDP model μ^T and a policy π^T

5 MLEMTRL: MTRL with Maximum Likelihood Model Transfer

Now, we present the proposed algorithm, MLEMTRL. The algorithm consists of two stages, a *model estimation* stage, and a *planning* stage. After having obtained a plan, then the agent will carry out its decision-making in the environment to acquire new experiences. We sketch an overview of MLEMTRL in Algorithm 7. For completeness, we also provide an extension to MLEMTRL called Meta-MLEMLTRL. This extension combines the MLEMTRL estimated model with the empirical model of the target task. This allows us to identify the true model even in the non-realisable setting. The details of this algorithm are available in Section 7.

5.1 Stage 1: Model Estimation

The first stage of the proposed algorithm is *model estimation*. During this procedure, the likelihood of the data needs to be computed for the appropriate MDP class. In the tabular setting, we use a product of multinomial likelihoods, where the data likelihood is over the distribution of successor states s' for a given state-action pair (s, a). In the LQR setting, we use a linear-Gaussian likelihood, which is also expressed as a product over data observed from the

target MDP.

Likelihood for Tabular MDPs. The log-likelihood that we attempt to maximise in tabular MDPs is a product over $|\mathcal{S}| \times |\mathcal{A}|$ of pairs of multinomials, where p_i is the probability of event i, $n^{s,a}$ is the number of times the state-action pairs (s, a) appear in the data D_t, and $x_i^{s,a}$ is the number of times the state-action pair (s, a, s_i) occurs in the data. That is, $\sum_{i=1}^{|\mathcal{S}|} x_i^{s,a} = n^{s,a}$. Specifically,

$$\log \mathbb{P}(D_t \,|\, \boldsymbol{p}) = \log \left(\prod_{s,a} n^{s,a}! \prod_{i=1}^{|\mathcal{S}|} \frac{p_i^{x_i^{s,a}}}{x_i^{s,a}!} \right) \tag{E.4}$$

Likelihood for Linear-Gaussian MDPs. For continuous state-action MDPs, we use a linear-Gaussian likelihood. In this context, let d_s be the dimensionality of the state-space, $\boldsymbol{s} \in \mathbb{R}^{d_s}$ and d_a be the dimensionality of the action-space. Then, the mean function $\mathbf{M}$ is a $\mathbb{R}^{d_s} \times \mathbb{R}^{d_a + d_s}$ matrix. The mean visitation count to the successor state $\boldsymbol{s}'_t$ when an action $\boldsymbol{a}_t$ is taken at state $\boldsymbol{s}_t$ is given by $\mathbf{M}(\boldsymbol{a}_t, \boldsymbol{s}_t)$. We denote the corresponding covariance matrix of size $\mathbb{R}^{d_s} \times \mathbb{R}^{d_s}$ by $\mathbf{S}$. Thus, we express the log-likelihood by

$$\log \mathbb{P}(D_t \,|\, \mathbf{M}, \mathbf{S}) = \log \prod_{i=1}^{t} \frac{\exp\left(-\frac{1}{2} \boldsymbol{v}^\top \mathbf{S}^{-1} \boldsymbol{v} \right)}{(2\pi)^{d_s/2} |\mathbf{S}|^{1/2}},$$
$$\text{where } \boldsymbol{s}'_i - \mathbf{M}(\boldsymbol{a}_i, \boldsymbol{s}_i) = \boldsymbol{v}.$$

Model Estimation as a Mixture of Models. As the optimisation problem involves weighing multiple source models together, we add a weight vector $\boldsymbol{w} \in [0, 1]^m$ with the usual property that $\boldsymbol{w}$ sum to 1. This addition results in another outer product over the likelihoods shown above. Henceforth, μ will refer to either the parameters associated with the product-Multinomial likelihood or the linear-Gaussian likelihood, depending on the model class.

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & \log \mathbb{P}(D_t \,|\, \Sigma_{i=1}^m w_i \mu_i), D_t \sim \mu^*, \mu_i \in \mathcal{M}_s, \\ \text{s.t.} \quad & \sum_{i=1}^{m} w_i = 1, w_i \geq 0. \end{aligned} \tag{E.5}$$

Because of the constraint on $\boldsymbol{w}$, this is a constrained nonlinear optimisation problem. We can use any optimiser algorithm, denoted by OPTIMISER, for this purpose.

Optimiser. In our implementations, we use Sequential Least-Squares Quadratic Programming (SLSQP) [34] as the Optimiser. SLSQP is a quasi-Newton method solving a quadratic programming subproblem for the Lagrangian of the objective function and the constraints.

Specifically, in Line 4 of Algorithm 7, we compute the next weight vector $\boldsymbol{w}^{t+1}$ by solving the optimisation problem in Eq. (E.5). Let $f(\boldsymbol{w}) = \log \mathbb{P}(D_t \mid \Sigma_{i=1}^{m} w_i \mu_i)$. Further, let $\lambda = \{\lambda_1, \ldots, \lambda_m\}$ and κ be Lagrange multipliers. We then define the Lagrangian $\mathcal{L}$,

$$\mathcal{L}(\boldsymbol{w}, \lambda, \kappa) = f(\boldsymbol{w}) - \lambda^\top \boldsymbol{w} - \kappa(1 - \mathbf{1}^\top \boldsymbol{w}). \tag{E.6}$$

Here, $\boldsymbol{w}^k$ is the k-th iterate. Finally, taking the local approximation of Eq. (E.5), we define the optimisation problem as:

$$\begin{aligned} &\min_{\boldsymbol{d}} \frac{1}{2} \boldsymbol{d}^\top \nabla^2 \mathcal{L}(\boldsymbol{w}, \lambda, \kappa) \boldsymbol{d} + \nabla f(\boldsymbol{w}^k) \boldsymbol{d} + f(\boldsymbol{w}^k) \\ &\text{s.t.}\, \boldsymbol{d} + \boldsymbol{w}^k \geq 0, \mathbf{1}^\top \boldsymbol{w}^k = 1. \end{aligned} \tag{E.7}$$

This minimisation problem yields the search direction $\boldsymbol{d}_k$ for the k-th iteration. Applying this iteratively and using the construction above ensures that the constraints posed in Eq. (E.5) are adhered to at every step of MLEMTRL. At convergence, the k-th iterate, $\boldsymbol{w}^k$ is considered as the next $\boldsymbol{w}^{t+1}$ in Line 4 of Algorithm 7.

5.2 Stage 2: Model-based Planning

When an appropriate model μ^t has been identified at time step t, the next stage of the algorithm involves model-based planning in the estimated MDP. We describe two model-based planning techniques, ValueIteration and RiccatiIteration for tabular MDPs and LQRs, respectively.

ValueIteration. Given the model, μ^t and the associated reward function $\mathcal{R}$, the optimal value function of μ^t can be computed iteratively as [3]:

$$V^*_{\mu^t}(s) = \max_a \sum_{s'} \mathcal{T}^a_{s,s'} \Big(\mathcal{R}(s,a) + \gamma V^*_{\mu^t}(s') \Big). \tag{E.8}$$

The fixed-point solution to Eq. (E.8) is the optimal value function. When the optimal value function has been obtained, one can simply select the action maximising the action-value function. Let π^{t+1} be the policy selecting the

maximising action for every state, then π^{t+1} is the policy the model-based planner will use at time step $t+1$.

RiccatiIteration. A LQR-based control system, and thus, the corresponding MDP, is defined by four system matrices [25]: $\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R}$. The matrices $\mathbf{A}, \mathbf{B}$ are associated with the transition model $\boldsymbol{s}_{t+1} - \boldsymbol{s}_t = \mathbf{A}\boldsymbol{s}_t + \mathbf{B}\boldsymbol{a}_t$. The matrices $\mathbf{Q}, \mathbf{R}$ dictate the quadratic cost (or reward) of a policy π under an MDP μ is

$$V_\mu^\pi = \sum_{t=0}^{T} \boldsymbol{s}_t^\top \mathbf{Q}\boldsymbol{s}_t + \boldsymbol{a}_t^\top \mathbf{R}\boldsymbol{a}_t.$$

Optimal policy is identified following [26] that states $\boldsymbol{a}_t = -\mathbf{K}\boldsymbol{s}_t$ at time t, where $\mathbf{K}$ is computed using $\mathbf{A}, \mathbf{B}, \mathbf{Q}, \mathbf{R}$.

6 Theoretical Analysis of MLEMTRL

In this section, we further justify the use of our framework by deriving worst-case performance degradation bounds relative to the optimal controller. The performance loss is shown to be related to the realisability of μ^* under $\mathcal{C}(\mathcal{M}_s)$. In Figure 1, we visualise the model dissimilarities, where $||\mu - \hat{\mu}||_1$ is the model estimation error, $||\mu^* - \mu||_1$ is the realisability gap and $||\mu^* - \hat{\mu}||_1$ the total deviation of the estimated model. Note that by the norm on MDP, we always refer to the L_1 norm over transition matrices.

Theorem 1 (Performance Gap for Non-Realisable Models)**:** *Let $\mu^* = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}^*, \gamma)$ be the true underlying MDP. Further, let $\mu = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma)$ be the maximum log-likelihood*

$$\mu \in \arg\max_{\mu' \in \mathcal{C}(\mathcal{M}_s)} \; -\log \mathbb{P}(D_\infty \,|\, \mu'), D_\infty \sim \mu^*$$

and $\hat{\mu} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \hat{\mathcal{T}}, \gamma)$ be a maximum log-likelihood estimator of μ. In addition, let $\pi^*, \pi, \hat{\pi}$ be the optimal policies for the respective MDPs. Then, if $\mathcal{R}$ is a bounded reward function $\forall_{(s,a)}\, r(s,a) \in [0,1]$ and with ϵ_{Estim} being the estimation error and

$$\epsilon_{\text{Realise}} \triangleq \min_{\mu \in \mathcal{C}(\mathcal{M}_s)} \|\mu^* - \mu\|_1$$

the realisability gap. Then, the performance gap is given by,

$$||V_{\mu^*}^* - V_{\mu^*}^{\hat{\pi}}||_\infty \leq \frac{3(\epsilon_{\text{Estim}} + \epsilon_{\text{Realise}})}{(1-\gamma)^2}. \tag{E.9}$$

This result is comparable to recent results such as [35] but here with an explicit decomposition into model estimation error and realisability gap terms.

Remark (Bound on L_1 Norm Difference in the Realisable Setting)**:** It is known [36]–[38] that in the realisable setting, it is possible to bound the model estimation error term ϵ_{Estim} via the following argument. Let μ^* be the true underlying MDP, and $\hat{\mu}$ be an MLE estimate of μ^*, as defined in Theorem 1. If $\mathcal{R}$ is a bounded reward function, i.e. $r(s,a) \in [0,1], \forall (s,a)$, and ϵ_{Estim} is upper bound on the L_1 norm between $\mathcal{T}^*$ and $\hat{\mathcal{T}}$. If $n^{s,a}$ be the number of times (s,a) occur together, then with probability $1 - SA\delta$,

$$||\mathcal{T}^* - \hat{\mathcal{T}}||_1 \leq \epsilon_{\text{Estim}} \leq \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \sqrt{\frac{2 \log\left((2^S - 2)/\delta\right)}{n^{s,a}}}.$$

From this, it can be said that the total L_1 norm then scales on the order of $\mathcal{O}(SA\sqrt{S + \log(1/\delta)}/\sqrt{T})$.

This result is specific to tabular MDPs. In tabular MDPs, the maximum likelihood estimate coincides with the empirical mean model.

Remark (Performance Gap in the Realisable Setting)**:** A trivial worst-case bound for the realisable case (Section 4.2) can be obtained by setting $\epsilon_{\text{Realise}} = 0$ because by definition of the realisable case $\mu^* \in \mathcal{C}(\mathcal{M}_s)$.

7 A Meta-Algorithm for MLEMTRL under Non-realisability

To guarantee good performance even in the non-realisable setting, we can proceed in two steps. First, we can add the target task to the set of source tasks. Second, we can construct a meta-algorithm, combining the model estimated by MLEMTRL and the empirical estimation of the target task. In this section, we propose a meta-algorithm based on the latter. We illustrate it in Algorithm 8.

The main change in Algorithm 8 compared to Algorithm 7 is internally keeping track of the empirical model, and in Line 8, computing a posterior probability distribution over the respective models by weighting the two likelihoods together with their respective priors. The resulting algorithm then trades off its bias to the prior based on the choice of prior hyperparameter p. Since asymptotically the likelihood of the data under the empirical model is greater than the likelihood of the data under the MLEM-estimated model,

Algorithm 8 Meta-MLEMTRL

Input: prior p, weights $\boldsymbol{w}^0$, m source MDPs $\mathcal{M}_s$, data D_0, discount factor γ, iterations T.
for $t = 0, \dots, T$ **do**
    // Stage 1: Obtain Model Weights //
    $\boldsymbol{w}^{t+1} \leftarrow \text{MLEMTRL}(\boldsymbol{w}^t, \mathcal{M}_s, \mathcal{D}_t, \gamma, 1)$
    Estimate the MDP: $\mu^{t+1} = \sum_{i=1}^{m} w_i \mu_i$
    Compute log-likelihood $\ell^{t+1}_{\text{MLEM}} = \log \mathbb{P}(\mathcal{D}_t \,|\, \mu^{t+1})$
    Compute log-likelihood of empirical model $\ell^{t+1}_{\text{Empirical}} = \log \mathbb{P}(\mathcal{D}_t \,|\, \hat{\mu}^{t+1})$
    Sample $\tilde{\mu}^{t+1}$ as μ^{t+1} w.p. $\propto p \exp\left(\ell^{t+1}_{\text{MLEM}}\right)$ and $\hat{\mu}^{t+1}$ w.p. $\propto (1-p) \exp\left(\ell^{t+1}_{\text{Empirical}}\right)$.
    // Stage 2: Model-based Planning //
    Compute the policy: $\pi^{t+1} \in \arg\max_{\pi} V^{\pi}_{\tilde{\mu}^{t+1}}$
    // Control //
    Observe $s_{t+1}, r_{t+1} \sim \mu^*(s_t, a_t), a_t \sim \pi^{t+1}(s_t)$
    Update the dataset $D_{t+1} = D_t \cup \{s_t, a_t, s_{t+1}, r_{t+1}\}$
return An estimated MDP model $\tilde{\mu}^T$ and a policy π^T

as more and more data is collected the Meta-MLEMTRL algorithm performs similarly to the optimal planning using the empirical estimate. This is intended behaviour and allows for the algorithm to asymptotically plan optimally even in the non-realisable setting.

We experimentally study the behaviour of Meta-MLEMTRL and its dependence on the prior parameter p in the next section.

8 Experimental Analysis

To benchmark the performance of MLEMTRL, we compare ourselves to a posterior sampling method (**PSRL**) [39], equipped with a combination of product-Dirichlet and product-Normal Inverse Gamma priors for the tabular setting, and Bayesian Multivariate Regression prior [40] for the continuous setting. In PSRL, at every round, a new model is sampled from the prior, and it learns in the target MDP from scratch. Finally, for model-based planning, we use RiccatiIterations to obtain the optimal linear controller for the sampled model. In the continuous action setting, we compare the performance

to the baseline algorithm multi-task soft-actor critic (**MT-SAC**) [41], [42] and a modified **MT-SAC-TRL** using data from the novel task during learning. In the tabular MDP setting, we compare against multi-task proximal policy optimisation (**MT-PPO)** [42], [43] and similarly **MT-PPO-TRL**.

The objectives of our empirical study are three-fold:

1. How does MLEMTRL impact performance in terms of **learning speed**, **jumpstart improvement** and **asymptotic convergence** compared to our baseline?

2. What is the performance loss of MLEMTRL in the **non-realisable setting**?

3. How does Meta-MLEMTRL perform in the **non-realisable setting**?

We conduct two kinds of experiments to verify our hypotheses. Firstly, in the upper row of Figure 2, we consider the realisable setting, where the novel task μ^* is part of the convex hull $\mathcal{C}(\mathcal{M}_s)$. In this case, we are looking to identify an improvement in some or all of the aforementioned qualities compared to the baselines. Further, in the bottom row of Figure 2, we investigate whether the algorithm can generalise to the case beyond what is supported by the theory in Section 4.2. We begin by recalling the goals of the transfer learning problem [13].

1. *Learning Speed Improvement:* A learning speed improvement would be indicated by the algorithm reaching its asymptotic convergence with less data.
2. *Asymptotic Improvement:* An asymptotic improvement would mean the algorithm converges asymptotically to a superior solution to that one of the baseline.
3. *Jumpstart Improvement:* A jumpstart improvement can be verified by the behaviour of the algorithm during the early learning process. In particular, if the algorithm starts at a better solution than the baseline, or has a simpler optimisation surface, it may more rapidly approach better solutions with much less data.

RL Environments. We test the algorithms in a tabular MDP, i.e. Chain [44], CartPole [45], and two LQR tasks in *Deepmind Control Suite* [46]: *dm_LQR_2_1* and *dm_LQR_6_2*. Further details on experimental setups are deferred to Appendix C.1.

(1) Impacts of Model Transfer with MLEMTRL. We begin by evaluating the proposed algorithm in the Chain environment. The results of the said

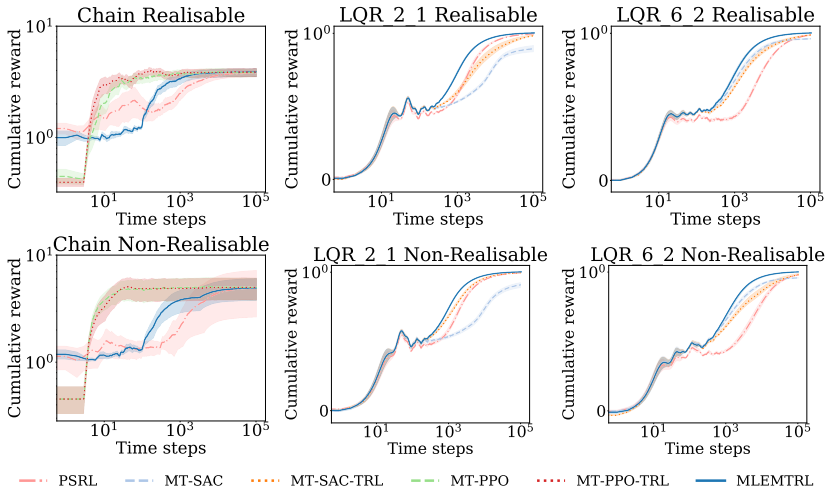


Figure 2: Depicted is the average cumulative reward at every time step computed over 10 novel tasks in the realisable/non-realisable setting. The shaded regions represent the standard error of the average cumulative reward at the time step.

experiment are available in the leftmost column of Figure 2. In it, we evaluate the performance of MLEMTRL against PSRL, MT-PPO, MT-PPO-TRL. The experiments are done by varying the slippage parameter $p \in [0.00, 0.50]$ and the results are computed for each different setup of Chain from scratch. In this experiment, we can see the baseline algorithms MT-PPO and MT-PPO-TRL perform very well. This could partially be explained by PSRL and MLEMTRL not only having to learn the transition distribution but also the reward function. The value function transfer in the PPO-based baselines implicitly transfers not only the empirical transition model but also the reward function. We can see that MLEMTRL has improved learning speed compared to PSRL in both realisable and non-realisable settings. An additional experiment with a known reward function across tasks is shown in Figure 7 in Appendix.

In the centre and rightmost columns of Figure 2, we can see the results of running the algorithms in the LQR settings with the baseline algorithms PSRL, MT-SAC and MT-SAC-TRL. The variation over tasks is given by the randomness over the stiffness of the joints in the problem. In these experiments, we can see a clear advantage of MLEMTRL compared to all baselines in terms

of learning speed improvements, and in some cases, asymptotic performance.

In Figure 2, the performance metric is the average cumulative reward at every time step, for 10^5 time steps and the shaded region represents the standard deviation, where the statistics are computed over 10 independent tasks.

(2) Impact of Realisability Gap on Regret. Now, we further illustrate the observed relation between model dissimilarity and degradation in performance. Figure 3 depicts the regret against the KL-divergence of the target model to the best proxy model in the convex set. We observe that model dissimilarity influences the performance gap in MLEMTRL. This is also justified in Section 6 where the bounds have an explicit dependency on the model difference. In this figure, only the non-zero regret experiments are shown. This is to have an idea of which models result in poor performance. As its shown, it is those models that are very dissimilar. Additional results in

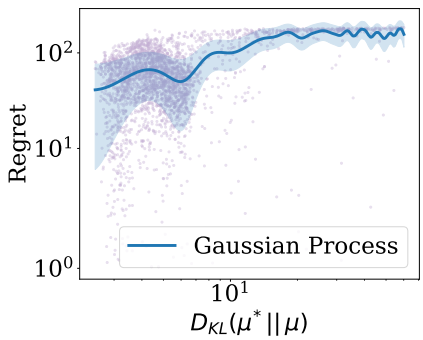


Figure 3: A log-log plot of regret vs. KL-divergence between the true MDP and the best proxy model in CartPole. The thick blue line is a Gaussian Process regression model fitted on observed data (in purple).

(3) Performance of Meta-MLEMTRL under Non-realisability. In order to validate the performance of the proposed meta algorithm Meta-MLEMTRL, we perform an ablation study over the prior hyperparameter p. In Figure 4, we illustrate the results of running the Meta-MLEMTRL algorithm in the Chain environment for both the realisable and non-realisable settings. The choice of p determines how much the algorithm should be biased towards the model estimated using MLEMTRL and in the case when $p = 1$, Meta-MLEMTRL reduces to MLEMTRL. Similarly, if $p = 0$ then the algorithm will forego

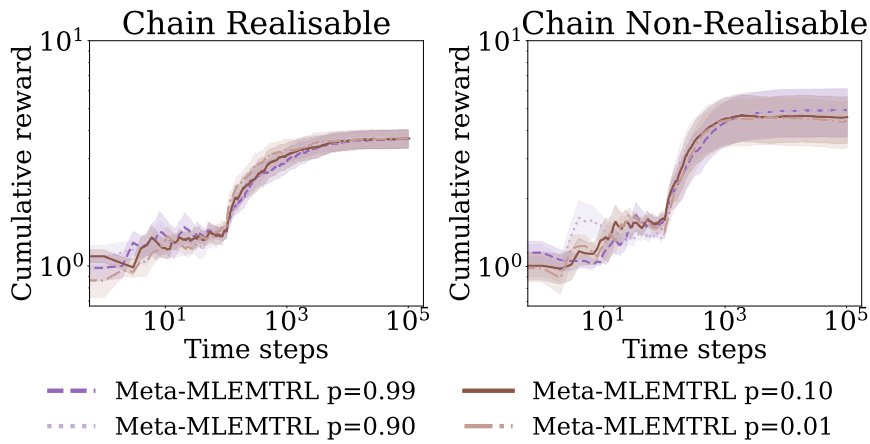


Figure 4: Figure depicting an ablation study of the prior parameter p in the Meta-MLEMTRL algorithm. The y-axis is the average cumulative reward at each time step computed over 10 novel tasks and the shaded region represents the standard error. When $p = 1$, the algorithm reduces to MLEMTRL and when $p = 0$ the algorithm reduces to standard maximum likelihood model estimation.

the MLEMTRL-estimate for the empirical estimate. Figure 4 shows that the performance of Meta-MLEMTRL is stable in long-run for different values of p. However, we identify that higher p values yield positive improvements in the cumulative reward over 10^5 steps, especially in the non-realisable setting. This indicates that the MLEMTRL-estimated model acts a good representation, while combined with the asymptotically converging empirical estimate obtained by Meta-MLEMTRL.

Summary of Results. In the experiments, we sought to identify whether the proposed algorithm shows superiority in terms of the transfer learning goals given by [13]. In the LQR-based environments, we can see a clear superiority of MLEMTRL in terms of learning speed compared to all baselines and in some cases, an asymptotic improvement. In the Chain environment, the proposed algorithm, MLEMTRL, outperforms PSRL in terms of learning speed. Also, we perform an ablation study of Meta-MLEMTRL under realisable and non-realisable settings, demonstrating provable improvements in the asymptotic and non-realisable regimes.

9 Discussions and Future Work

In this work, we aim to answer two central questions.

1. *How can we accurately construct a model using a set of source models for an RL agent deployed in the wild?*

2. *Does the constructed model allow us to perform efficient planning and yield improvements over learning from scratch?*

Our answer to the first question is by adopting the *Model Transfer Reinforcement Learning* framework and weighting existing knowledge together with data from the novel task. We accomplished this by following a maximum likelihood-based approach. This has led to a novel algorithm, MLEMTRL, consisting of a model identification stage and a model-based planning stage. The second question is answered by the empirical results in Section 8 and the theoretical results in Section 6. Further, the model allows generalising to novel tasks, given that the tasks are similar enough to the existing task(s).

We motivate the use of our framework in settings where an agent is to be deployed in a new domain that is similar to existing, known, domains. We verify the quick, near-optimal performance of the algorithm in the case where the new domain is similar and we prove worst-case performance bounds of the algorithm in both the realisable and non-realisable settings. As a future work, it would be interesting to study the MTRL framework under Bayesian setting [21] and to deploy it with a risk-sensitive value function [47], [48].

ACKNOWLEDGMENTS

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. D. Basu acknowledges the Inria-Kyoto University Associate Team “RELIANT” and the ANR JCJC grant for the REPUBLIC project (ANR22-CE23-0003-01).

References

[1] G. Dulac-Arnold *et al.*, "Challenges of real-world reinforcement learning: Definitions, benchmarks and analysis," *Machine Learning*, vol. 110, no. 9, pp. 2419–2468, 2021.

[2] J. Skirzyński, F. Becker, and F. Lieder, "Automatic discovery of interpretable planning strategies," *Machine Learning*, vol. 110, no. 9, pp. 2641–2683, 2021.

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

[4] T. M. Moerland, J. Broekens, A. Plaat, C. M. Jonker, *et al.*, "Model-based reinforcement learning: A survey," *Foundations and Trends® in Machine Learning*, vol. 16, no. 1, pp. 1–118, 2023.

[5] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 2, pp. 153–173, 2017.

[6] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*, 2018.

[7] M. E. Taylor, N. K. Jong, and P. Stone, "Transferring instances for model-based reinforcement learning," in *Joint European conference on machine learning and knowledge discovery in databases*, 2008.

[8] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey.," *Journal of Machine Learning Research*, vol. 10, no. 7, 2009.

[9] A. Lazaric, "Transfer in reinforcement learning: A framework and a survey," in *Reinforcement Learning*, Springer, 2012, pp. 143–173.

[10] R. Laroche and M. Barlier, "Transfer reinforcement learning with shared dynamics," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[11] A. Zhang, H. Satija, and J. Pineau, "Decoupling dynamics and reward for transfer learning," *arXiv preprint arXiv:1804.10689*, 2018.

[12] A. A. Rusu *et al.*, "Policy distillation," *arXiv preprint arXiv:1511.06295*, 2015.

[13] P. Langley, "Transfer of knowledge in cognitive systems," in *Talk, workshop on Structural Knowledge Transfer for Machine Learning at the Twenty-Third International Conference on Machine Learning*, 2006.

[14] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou, "Transfer learning in deep reinforcement learning: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[15] E. Parisotto, J. L. Ba, and R. Salakhutdinov, "Actor-mimic: Deep multitask and transfer reinforcement learning," *arXiv:1511.06342*, 2015.

[16] I. Arnekvist, D. Kragic, and J. A. Stork, "Vpe: Variational policy embedding for transfer reinforcement learning," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019.

[17] C. G. Atkeson and J. C. Santamaria, "A comparison of direct and model-based reinforcement learning," in *Proceedings of international conference on robotics and automation*, vol. 4, 1997.

[18] C. Rommel, J. F. Bonnans, B. Gregorutti, and P. Martinon, "Aircraft dynamics identification for optimal control," in *7th European Conference on Aeronautics and Space Sciences (EUCASS 2017)*, 2017.

[19] A. Wilson, A. Fern, S. Ray, and P. Tadepalli, "Multi-task reinforcement learning: A hierarchical bayesian approach," in *Proceedings of the 24th international conference on Machine learning*, 2007.

[20] A. Lazaric and M. Ghavamzadeh, "Bayesian multi-task reinforcement learning," in *ICML-27th International Conference on Machine Learning*, 2010.

[21] A. Tamar, D. Soudry, and E. Zisselman, "Regularization guarantees generalization in bayesian reinforcement learning through algorithmic stability," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, 2022.

[22] F. L. Da Silva and A. H. R. Costa, "A survey on transfer learning for multiagent reinforcement learning systems," *Journal of Artificial Intelligence Research*, vol. 64, pp. 645–703, 2019.

[23] X. Liang, Y. Liu, T. Chen, M. Liu, and Q. Yang, "Federated transfer reinforcement learning for autonomous driving," in *Federated and Transfer Learning*, Springer, 2023, pp. 357–371.

[24] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[25] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[26] J. Willems, "Least squares stationary optimal control and the algebraic riccati equation," *IEEE Transactions on automatic control*, vol. 16, no. 6, pp. 621–634, 1971.

[27] D. E. Bell, "Regret in decision making under uncertainty," *Operations research*, vol. 30, no. 5, pp. 961–981, 1982.

[28] G. Casella and R. L. Berger, *Statistical inference*. Cengage Learning, 2021.

[29] J. Kiefer and J. Wolfowitz, "Consistency of the maximum likelihood estimator in the presence of infinitely many incidental parameters," *The Annals of Mathematical Statistics*, pp. 887–906, 1956.

[30] W. K. Newey and J. L. Powell, "Asymmetric least squares estimation and testing," *Econometrica: Journal of the Econometric Society*, pp. 819–847, 1987.

[31] G. Ciuperca, A. Ridolfi, and J. Idier, "Penalized maximum likelihood estimator for normal mixtures," *Scandinavian Journal of Statistics*, vol. 30, no. 1, pp. 45–59, 2003.

[32] R. Ouhamma, D. Basu, and O. Maillard, "Bilinear exponential family of mdps: Frequentist regret bound with tractable exploration & planning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023.

[33] J. Aitchison and S. Silvey, "Maximum-likelihood estimation of parameters subject to restraints," *The annals of mathematical Statistics*, vol. 29, no. 3, pp. 813–828, 1958.

[34] D. Kraft, "A software package for sequential quadratic programming," *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*, 1988.

[35] A. Zhang, S. Sodhani, K. Khetarpal, and J. Pineau, "Learning robust state abstractions for hidden-parameter block mdps," in *International Conference on Learning Representations*, 2020.

[36] A. L. Strehl and M. L. Littman, "A theoretical analysis of model-based interval estimation," in *Proceedings of the 22nd international conference on Machine learning*, 2005.

[37] P. Auer, T. Jaksch, and R. Ortner, "Near-optimal regret bounds for reinforcement learning," *Advances in neural information processing systems*, vol. 21, 2008.

[38] J. Qian, R. Fruit, M. Pirotta, and A. Lazaric, "Concentration inequalities for multinoulli random variables," *arXiv preprint arXiv:2001.11595*, 2020.

[39] I. Osband, D. Russo, and B. Van Roy, "(more) efficient reinforcement learning via posterior sampling," in *Advances in Neural Information Processing Systems*, 2013.

[40] T. Minka, "Bayesian linear regression," Citeseer, Tech. Rep., 2000.

[41] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, 2018.

[42] T. Yu *et al.*, "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Conference on robot learning*, 2020.

[43] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[44] R. Dearden, N. Friedman, and S. Russell, "Bayesian q-learning," *AAAI Conference on Artificial Intelligence (AAAI)*, vol. 1998, pp. 761–768, 1998.

[45] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE transactions on systems, man, and cybernetics*, no. 5, pp. 834–846, 1983.

[46] Y. Tassa *et al.*, "Deepmind control suite," *arXiv preprint:1801.00690*, 2018.

[47] H. Eriksson, D. Basu, M. Alibeigi, and C. Dimitrakakis, "Sentinel: Taming uncertainty with ensemble based distributional reinforcement learning," in *Uncertainty in Artificial Intelligence*, 2022.

[48] Y. Flet-Berliac and D. Basu, "Saac: Safe reinforcement learning as an adversarial game of actor-critics," in *RLDM 2022-The Multi-disciplinary Conference on Reinforcement Learning and Decision Making*, 2022.