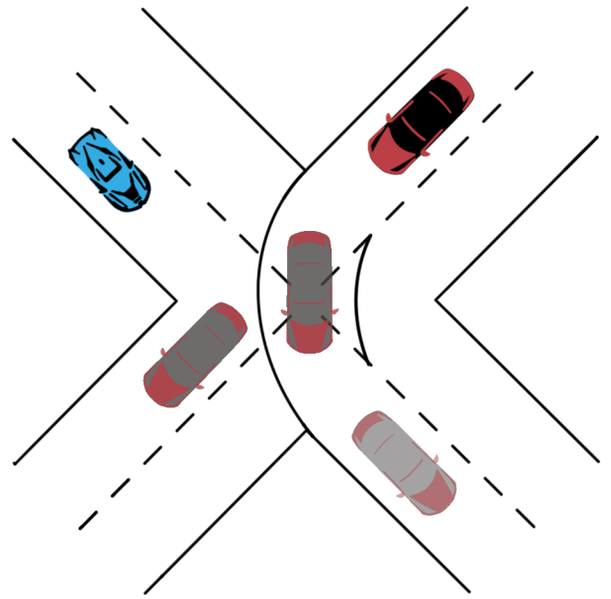
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

PHD THESIS

To Be Written

TOMMY TRAM • Learning When To Drive in Uncertrain Environments • 2022



Learning How To Drive in Uncertrain Environments

A reinforcement learning approach to decision making

TOMMY TRAM

CHALMERS
UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Learning When To Drive in Uncertrain Environments

A reinforcement learning approach to decision making

Tommy Tram

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2022

Learning When To Drive in Uncertrain Environments

A reinforcement learning approach to decision making

Tommy 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, December 2022

To my loving family

Learning When To Drive in Uncertrain Environments

A reinforcement learning approach to decision making

Tommy Tram
Department of Electrical Engineering
Chalmers University of Technology

Abstract

To Be Written

Keywords: Autonomous driving, reinforcement learning, decision making, uncertain environments, 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 Sjöberg, and Mykel Kochenderfer, "Belief State Reinforcement Learning for Autonomous Vehicles in Intersections". *Submitted to IEEE Transactions on Intelligent Vehicles.*

[E] Hannes Eriksson, Tommy Tram, Debabrota Basu, Jonas Sjöberg, and Christos Dimitrakakis, "Reinforcement Learning in the Wild with Maximum Likelihood-based Model Transfer". *Artificial Intelligence and Statistics 2023 (AISTATS).*

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 the oppertunity of pursuing a doctoral degree as your student. It has been an honor and a pleasure.

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 created 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. We started out as supervisor and master thesis students but ended up as friends.

ToDo: Hannes and debba

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. To that end, I am deeply grateful that Professor Mykel Kochenderfer that 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

ToDo: continue

Tommy Tram,
Göteborg, December, 2022.

Acronyms

AD:	Autonomous driving
ADAS:	Advanced driving assistance systems
AV:	Autonomous vehicles
CNN:	Convolutional neural network
DQN:	Deep Q-network
EQN:	Ensabmle quantile networks
IDM:	Intelligent driver
LSTM:	Long short-term memory
MDP:	Markov descision process
MPC:	Model predictive control
POMDP:	Partially observable Markov descision process
RL:	Reinforcement learning

Contents

Part I

Overview

CHAPTER 1

Introduction

The way of transportation is currently evolving, and autonomous driving technology is expected to have a big impact on this transformation. Over one million people are killed in traffic-related accidents each year, where the vast majority of the accidents are caused by human mistakes [1], [2]. By helping humans with perception, prediction and decision making autonomous driving could significantly improve traffic safety and make way for new innovative road infrastructure. Without the requirement of a present human for each transportation vehicle, the efficiency of traffic can be improved by scheduling commercial transports outside of rush hours [3].

ToDo: cite future of mixed-autonomy

ToDo: Level 2: Volvo pilot assist and tesla auto pilot. Level 4: Waymo. Woven automated city

Thanks to the rapid success of deep learning during the last decades, major progress towards deploying autonomous vehicles in the real world. One clear benefactor of the new deep learning technics are the perception systems [4]. However, for scenarios complex scenarios such as urban intersections and round

about with dense traffic remain challenging for autonomous vehicles. This thesis presents methods for generating efficient and scalable decision strategies for autonomous vehicles. In this chapter we introduce the problem of for tactical decision making in autonomous driving (AD), the scenarios that is studied, and a system architecture that separates the decision maker from the safety assurance. Then, the research questions, scope, limitations and contributions of this thesis is presented.

1.1 Autonomous Driving

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 [5], also referred to as L0-L5. L0 is a vehicle with no autonomy, whereas a fully autonomous vehicle that can operate in any environment and without any human supervision is defined as L5. Popular advance driver assistance systems (ADAS) functions today, like Tesla autopilot and the Volvo Pilot Assist, are classified as level 2, with the main criteria being that the driver is in control and is supported by the system. This puts a requirement on the driver to always supervise the vehicle and take over when needed to maintain safety. For L3 and higher the responsibility of driving is on the system. At L3 the driver still has to take control over the vehicle but at the request of the system, and at L4 and L5 the autonomous driving features no longer require the driver to take over. Finally, the main difference between L4 and L5 is the capability of driving anywhere, under all conditions. The methods presented in this thesis are aimed at an autonomy level between L3 and L5. At this level the system is expected to handle all aspects of driving within a specific task such as crossing an unsignalized intersection.

In this thesis, we refer the automated vehicle making the decisions as the ego vehicle. Other traffic participants are assumed to be vehicles driven by other human drivers but could be extended to pedestrians bicyclist and other autonomous vehicles.

1.1.1 Scenarios and unsignalized intersections

An unsignalized intersection is defined as any junction of two or more roads where the right-of-way for a car, bicycle and pedestrian is not controlled by a regulatory (i.e., STOP or YIELD) sign or a traffic signal. Recently nontraditional intersections are also becoming increasingly popular. The goal of these designs is to reduce the number and/or severity of conflict points by altering the customary vehicular paths at the intersection. In light of the increased focus on and occurrence of these intersection types, it is expected that the application of nontraditional designs will continue to spread.

In this thesis we draw a separation between scenario and intersection. An intersection refers the geometrical shape of the intersection, like number of junctions, conflict points, turns and angle of incidence. While a scenario is defined as the combination of intersection and traffic participants and their intentions. More detail on the specific intersections and scenarios considered in this work is presented in Chapter 4.

Tommy: what we want to say here is: define an unsignalized intersection. There are many variations. Roundabouts are defined as unsignalized intersection. To satisfy the requirement of being able to drive anywhere for level 5 it is necessary to find a method that can scale to these different scenarios

Tommy: maybe define intersection as the map, and scenario as the combination of traffic participants and their intentions.

ToDo: Find an appropriate place for this section

In light of the increased focus on and occurrence of these intersection types nationwide, it is expected that the application of nontraditional designs will continue to spread.

1.2 System architecture

The architecture of an autonomous driving system can be divided into perception, planning and control **Schwarting2018**, **koretenkamp2016**.

The perception module is responsible for sensing and mapping the environment with the use of sensors such as LIDARs, cameras, radars etc. The

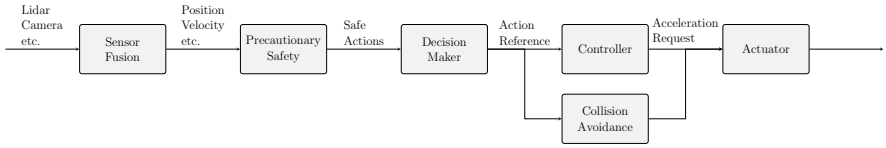


Figure 1.1: Representation of the system architecture.

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. 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. 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. Unlike the decision maker, the collision avoidance module main goal is to identify imminent danger and therefore has access to more aggressive actions like emergency breaking 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" [6]. 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 arguably the most important part for enabling autonomous driving, the work in this paper does not make any safety guaranties. Instead, we proposed that the decision making algorithms presented in this paper are used in system architecture like in Figure **??**. This way, the higher ASIL classifications are on the precautionary safety and collision avoidance modules while the decision makers mainly focus on comfort and the ASIL classification could be on the lower levels and in the best case even be classified as QM.

Tommy: To explore different ways to make the driving policy safe and increase the capability of handling uncertainty in the environment. But even so the state of machine learning and neural networks today is still very limited when its comes to guaranteeing safety. Therefore, it is important to have a system architecture that can separate safety guarantees to another module so that the main benefits of using machine learning can be fully utilized.

1.3 Approach

Human driving can be described as sequential decision making under uncertainty. Decisions such as overtaking a slow driver or when to cross an intersection are often made with limited information and some prediction estimate based on previous experience. This section briefly introduce the intersection problem, why its hard and the research questions that are studied in this paper. A simple unsignalized intersection is shown in Figure 4.1. In some cases like highway driving the uncertainty is very low, but when it comes to more urban environment this uncertainty increases. Compared to highway, urban environments introduce more uncertainty. Investigate how reinforcement learning (RL) methods can be used in practice to create a tactical decision making agent for AD.

Tommy: This paper focus on the DQN algorithm and investigate how far we can push it for decision making

1. We want to drive through intersections.

2. The intersection can be of different shapes.

3. There will be other cars crossing the same intersection.

The work presented in this thesis investigate the following research questions:

Q1. How can RL be used to create a decision-making agent for autonomous driving through an unsignalized intersection?

Q2. Can we find a good driving policy without explicitly predicting other drivers intentions?

Q3. How can AD domain knowledge and models be used to improve the action and state space for a RL agent?

Q4. How can the quality of a RL agent be improved by accounting for uncertainty?

1.4 Scope and limitations

The following aspects of RL to create a tactical decision making agent for autonomous driving are not considered in this thesis.

1. We have access to sensors on-board the ego vehicle. We do not have v2v, or v2x communication.

2. We do not assume any knowledge of traffic signs or traffic lights.

3. Do not guarantee safety, the best we can do its making the decisions not trigger collision avoidance functions.

4. tested in simulation environments and not real world.

5. This work considers the control of one vehicle and not multiple agents.

6. does not optimize the reward function.

   To compensate for not having v2v or v2x communication, we have to, directly or indirectly, predict what other driver will do.

1.5 Contributions

ToDo: Rewrite once thesis is in a better state

The main contributions of this thesis are:

1. General approach to creating a decision making agent for driving in interactions.
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. REWRITE: A belief state representation using a particle filter and a comparison and analysis of different algorithms that utilize the belief state.
4. Two approached to solving a POMDP with hidden intention state. LSTM layer and belief state.
5. General state space representation that is invariat to permutations of the intersection design.
6. Extension of RL methods that provide an estimate of the epistemic uncertainty and use it to create a confidence criteria that can identify situations with high uncertainty.
7. **ToDo:** Transfer reinforcement learning, finding a policy for mdps

1.6 Thesis outline

FILL

CHAPTER 2

Related work

ToDo: Urban challange, winner Carnegie Mellon University John Dolan

2.1 Rule based methods and finite state machines

ToDo: Rule based methods and finite state machines

Limitations: • Requires anticipating every situations • Difficult to scale to complex scenarios • Hard to take into account uncertainty (e.g. perception noise)

2.2 Planning methods

ToDo: Motion planning, Predicting motion of surrounding vehicles, reactive (not intersactive)

MPC cite ivo: A robust scenario MPC approach for uncertain multi-modal obstacles, Real-Time Constrained Trajectory Planning and Vehicle Control for Proactive Autonomous Driving With Road Users.

Limitations: • Requires a model • Computationally expensive

2.3 Learning based methods

ToDo: POMDP model, online and offline solvers

Cite Maxime: Cooperation-aware reinforcement learning for merging in dense traffic, Belief state planning for autonomously navigating urban intersections.

ToDo: online solvers, MCTS

MCTS cite CJ: Combining planning and deep reinforcement learning in tactical decision making for autonomous driving, Tactical decision-making in autonomous driving by reinforcement learning with uncertainty estimation.

ToDo: cite Krook for formal methods and safety, the work from this thesis could be used to create this precationary module that ensure the decision maker in this are only allowed to execute safe actions.

ToDo: cite Bo wahlberg, Morteza Haghir Chehreghani, Fernandez Llorca David, Christian Berge

CHAPTER 3

Technical background

This chapter briefly introduce the partially observable Markov decision process (POMDP) framework and reinforcement learning. A more comprehensive overview of POMDPs and RL is given in the books by Kochenderfer [7] and Sutton and Barto [8], 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 Markov Decision Process (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 defines as the tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, described by the following list [7]:

- The state space $\mathcal{S}$ represents the set of all possible states of the environment. 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 on 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 to action and the goal of the agent is to take a sequence of actions that maximize the accumulated reward. 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 use a POMDP, which is an extension to the MDP that also models state uncertainty. 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 it 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: $O(o, s', a) = \Pr(o|s', a)$.

For 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.

In this paper, the observable states 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 formulates the POMDP studied in this work.

3.2 Reinforcement learning

ToDo: rewrite

In many problems, the state transition probabilities or the reward function are not known. These problems can be solved by reinforcement learning techniques, in which the agent learns how to behave from interacting with the environment [7, Ch. 5]. Compared to supervised learning, reinforcement learning presents some additional challenges. Since the data that are available to an RL-agent depends on its current policy, the agent must balance exploring the environment and exploiting the knowledge it has already gained. Furthermore, a reward that the agent receives may depend on a crucial decision that was

taken earlier in time, which makes it important to assign rewards to the correct decisions.

RL algorithms can be divided into model-based and model-free approaches [7, Ch. 5]. In the model-based versions, the agent first tries to estimate a representation of the state transition function T and then use a planning algorithm to find a policy. On the contrary, as the name suggests, model-free RL algorithms do not explicitly construct a model of the environment to decide which actions to take. The model-free approaches can be further divided into value-based and policy-based techniques. Value-based algorithms, such as Q-learning, aim to learn the value of each state and thereby implicitly define a policy. Policy-based techniques instead search for the optimal policy directly in the policy space, either by policy gradient methods or gradient-free methods, such as evolutionary optimization. There are also hybrid techniques that are both policy and value-based, such as actor critic methods.

RL algorithms generally assume that the environment is modeled as an MDP, i.e., that the state of the environment is known by the agent. However, in many cases of interest, only partial information about the state of the environment is available, which is modeled in the POMDP framework.

CHAPTER 4

Modeling Intersection Driving Scenarios

RQ 1: How can RL be used to create a decision-making agent for autonomous driving through an unsignalized intersection?

As previously mentioned in Section 1.3, driving through an intersection is a sequential decision making problem and can be mathematically formulated using a POMDP, introduced in Section 3.1.1. This Chapter defines the POMDP for the intersection studied in this thesis, by To be able to derive algorithms for decision making we must first describe the mathematical formulation of the problem.

4.1 Intersection scenarios

Figure **??** shows a simple intersection with one crossing point.

Tommy: show examples of intersections.

Tommy: zone 0 - after intersection, zone 1 - conflict zone, zone 2 - right before the intersection, zone 3 - first obsereved interstion to zone 2

4.2 State space

From Section 3.1, the state space contains all the information necessary about the agent and environment to be able to transition to any given state. In the scenario from Figure **??**, the red car on the horizontal lane represents the ego vehicle and the blue cars on the vertical lane are the other vehicels. Let's start by defining the information needed. Starting with the terminal states: goal, to know if this state is reached we need to know the distance from ego to goal p_{goal}. Collision, we need to know the position of all surrounding traffic participants and a description of the intersection itself. Instead of using a Cartesian coordinate system, we propose using relative distance measures instead. This way, states describing the intersection and its participants are generalizable to different intersection designs, e.g., the angle of incidence and number of crossing points. The we have velocity and acceleration of all the traffic participants, to be able to predict what position they will be in the next state. Finally, the intention of all the other participants. This state is necessary to efficently navigate through an interdection. Paper Dshows a comparision 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. Paper Aproposed the first set of states.

Tommy: Coordinate system, distances to intersection, position, velocity and accelration of other vehicles. abstract away the information about traffic lights, traffic signs, as intention

4.3 Action space

Tommy: options, take way, give way, follow car

One of the limitations of deep Q-learning is that the action space has to be discreet. In other work it is common to set the action space to diffecnt acceleration request. However, we propose using short term goals as actions, this is also refered to as options **options**. The short term goals are high level objectives like stop at the start of the intersection, follow car with id 1 or drive through the intersection at the reference speed. This high level action is then sent to a sliding mode controller that generates an acceleration. In Paper Bthe actions is sent to a model predictive control (MPC) to generate a

velocity profile that considers consecutive intersection points, this appoach is decribed in more detail in Chapter 6.

4.4 Transistion model

In this work the transision model is not known to the agent and RL is used to learn this model through experience, by taking actions at different states in an environment and recording the reward and what state the agent transition into. the environment in this work is a simulator and the main thing the agent is traying to learn is the transision of the other vehicles which depends on their intentions. The intentions are models as predetermined actions while folloing a intelligent driver model (IDM) ontop of that. This makes the interaction between cars more complicated.

Tommy: IDM, and other agents behaviors/intentions.

Paper Arandom parameters, speeds and spawn rates. Paper Bcautious, give way, take way

4.4.1 Simulation scenarios or simulator

parameters, randomized cars. Behaviors. spawn rates and more. up to four cars. singla crossing, double crossing,

4.5 Observation model

The observation model can be interpeted as the noise from the sensors, Paper Aassumes perfect sensing while Paper Dhas some added noise to the observed states. The observation space is usually the same as the state space without the intention state. Because there are no sensors that can detect other drivers intentions.

Tommy: Everything in the state space except intention. Everything that is observable through the sensors in the car.

4.6 Reward function

designing the reward model from the objective we are trying to achieve. this is not the optimal values. We are starting of a relative reward difference around 0 and 1. Then hand tune to get a performance close to the desired outcome. All papers formulated the reward based on the terminal states: goal, collision and timeout. Paper Aand Paper Bhas a continious negative reward for change in acceleration to punish jerk that would come from changing between actions that would make it uncomfortable for the passengers. Paper Aalso gives a relative large negative reward for choosing to follow a car that does not exist.

4.7 Discussion

Why is it hard.

Tommy: zone 0 - after intersection, zone 1 - conflict zone, zone 2 - right before the intersection, zone 3 - first obsereved interstion to zone 2

With the MDP, defined a typical intersection can be described as in Figure 4.1. From the figure we can divide the path of the ego vehicle into four zones. Starting from the end, zone 0 is the "safe zone" where the ego is our of danger and can return to nominal driving. Zone 1 is the conflict zone, this is where there is a possibility to collide with another vehicle. Zone 2 is critical decision zone, where this is the last chance the vehicle has to stop or cross. We define the size of this zone as the minimal distance the car needs to come to a complete stop to the start of the conflict zone. The final zone is zone 3, the information gathering zone, and is the furthest from the intersection and where the agent can observe the scenario and the other vehicles behavior over time.

the goal is to reach zone 0, to do this the agent would want to minimize the time in zone 1, if there is chance of intersection with another car. Because our actions are formulated as options and designed to be conformtable with lower acceleration rates. The size of zone 2 is dependent on the vehicels current speed, which is dependent on how the vehicle behaved in zone 3. Now there are two conflicting strategies, to minimize time in zone 1 the agent wants a high speed coming into the intersection. while it would want a low speed to shorten the zone 2 and the critical decision.

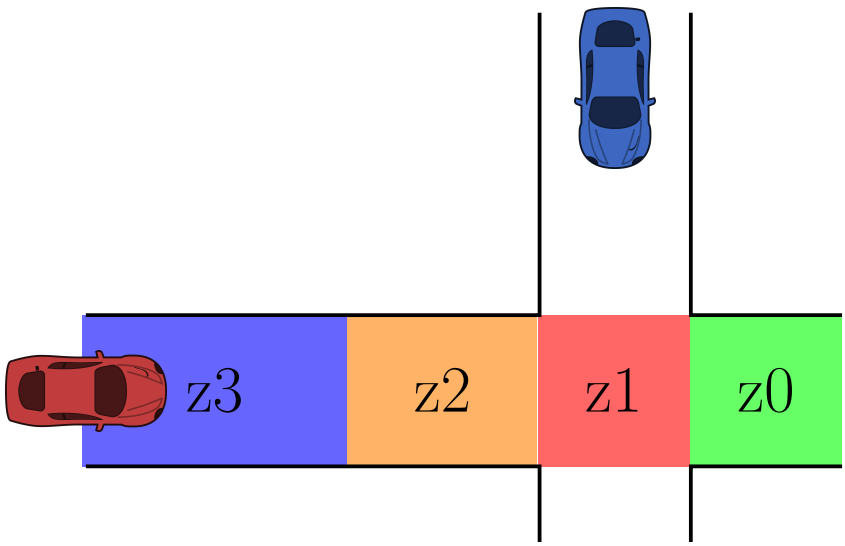


Figure 4.1: Intersection scenario divided into zones describing what is required of the decision maker in different zones

If we know the intention of the other vehicles the stochasticity in zone 1 would be gone and the problem becomes a scheduling problem of creating a velocity profile that minimizes the time to cross.

CHAPTER 5

Learning without a model

RQ 1: Can we find a good driving policy without explicitly predicting other drivers intentions?

Tommy: Deep Q-learning approach

We want to formulate the problem in such a way that abstracts the information of traffic lights, traffic signs and intention. This way the car is closer to L5 by not relying on the different traffic lights.

One motivation example is in how traffic lights works. In Sweden, we have sensors that can sense if there are cars in an intersection and create a traffic light schedule accordingly, compared to the US where the traffic signals set up using timers. As a consequence, Drivers approaching a yellow light

The deep Q-network (DQN) algorithm uses a neural network (NN) and has two disadvantages. One is that the size of the network is fixed, this forces the input space to be fixed. Our state space is defined by the physical state of the surrounding cars, and the number of cars we observe at each situation variate.

5.1 Approach (State representation, observable and unobservable)

This paper explored the possibility of solving the problem with RL by trying a verity of different methods from the rainbow paper with the addition to the LSTM layer and presented the results that had the highest impact on the conversion.

ToDo: State representation

This section describes the general state representation used in this research that enables these methods to be generalizable for different type of intersection and crossings. By describing the state space as a set of distances to intersection points, we can abstract the map layout of different intersections and the same algorithms would work for intersections variations that we haven't specifically trained on.

ToDo: add image of intersection scenario with 90 degree entry point and 45 degree entry point.

ToDo: Observable states

Position, velocity and acceleration.

ToDo: Unobservable states

Intentions. Traffic lights and traffic signs.

ToDo: explain rewards

large negative reward for invalid actions.

5.2 Simulated experiments

5.3 Results and discussion

- STG as actions
- shared weights

- effect of replay, dropout
- comparing a dqn and Drqn

however sensitive to noise.

LSTM take into account the history, but when applied in the real world with noise the model did not perform as well. The immidiate reward for jerk is

Tommy: finding time to intersection and position itself in a way that does not conflict with other cars.

Tommy: FIND A SECTION. Collisions in this thesis, may sound critical and extreme. But collisions in this content is for the purpose of the simulator and for the terminal state of the agent. Translated to a system perspective, it would mean that a backup collision avoidance algorithm had to interfere and in the worst case take over.

CHAPTER 6

Combining reinforcement learning and model based optimization

RQ 3: How can AD domain knowledge and models be used to improve the action and state space for a RL agent?

ToDo: as we see in previous chapter

6.1 MPC

ToDo: rewrite

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. In particular, for every time instance, a mathematical model of the controlled system is used to simulate the future states over a finite horizon, while a sequence of control inputs are selected and optimized given an objective cost function. The first element in the sequence

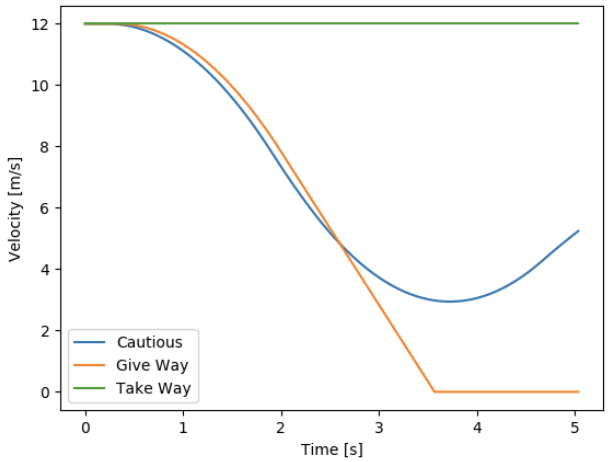


Figure 6.1: Example of how velocity profile of the different intention agents can look like. All agents have the same starting velocity of 12m/s and are approaching the same intersection

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.

6.2 Approach, (Action space, options. MPC)

MPC has a mixed integer problem, calculating the optimal path for all possible action is very computationally heavy. RL DQN. Only has discrete actions. Can not guarantee safety but is good at choosing actions with the best utility (value). The reward function takes in the predicted outcome of the model in the MPC and can penalize the choice of action. but if experience show that the outcome is better than the model, it can choose to take a bad action that would lead to a better total reward compared to only following a conservative model.

Tommy: in this work we simulate three different intention agents, take way, give way and cautious agent.

ToDo: reward function

added Q masking. Q-masking reduce the search space, in this work we showed it for unavailable actions but could be extended to actions limited by the precautionary safety module. By combining the having the mpc cost as a negative rewward the DQN can balance the control cost with the high level goal of reaching the goal and even choose an action that is on average good on a high level but may not seem that way to the MPC.

Given the state representation, the dynamics of the vehicle is then modeled using a triple integrator with jerk as control input.

Tommy: mpc cost function

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

$$J = \min_{\bar{\mathbf{x}},\bar{\mathbf{u}}} \sum_{k=0}^{N-1} \begin{bmatrix} \bar{\mathbf{x}}_k - \mathbf{r}_k^{\mathbf{x}} \\ \bar{\mathbf{u}}_k - \mathbf{r}_k^{\mathbf{u}} \end{bmatrix}^\top \begin{bmatrix} Q & S^\top \\ S & R \end{bmatrix} \begin{bmatrix} \bar{\mathbf{x}}_k - \mathbf{r}_k^{\mathbf{x}} \\ \bar{\mathbf{u}}_k - \mathbf{r}_k^{\mathbf{u}} \end{bmatrix} \tag{6.1a}$$

$$+ \begin{bmatrix} \bar{\mathbf{x}}_N - \mathbf{r}_N^{\mathbf{x}} \end{bmatrix}^\top P \begin{bmatrix} \bar{\mathbf{x}}_N - \mathbf{r}_N^{\mathbf{x}} \end{bmatrix}$$

$$\text{s.t.} \quad \bar{\mathbf{x}}_0 = \hat{\mathbf{x}}_0, \tag{6.1b}$$

$$\bar{\mathbf{x}}_{k+1} = A\bar{\mathbf{x}}_k + B\bar{\mathbf{u}}_k, \tag{6.1c}$$

$$h(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k, \bar{\mathbf{o}}_k, a_k) \leq 0, \tag{6.1d}$$

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 (**??**) enforces that the prediction starts at the current state estimate $\hat{\mathbf{x}}_0$, (**??**) enforces the system dynamics, and (**??**) 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.

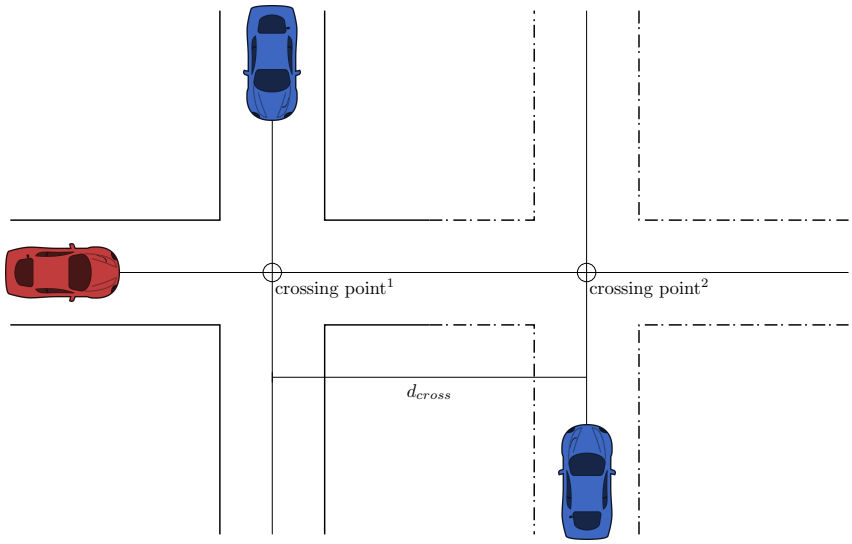


Figure 6.2: Illustration of a intersection scenario, where the solid line is a single crossing and together with the dashed line creates a double crossing.

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

Table 6.1: Average success rates and collision to timeout rates.

6.3 Simulated experiments

We show the difference in a multi crossing scenario where the MPC can plan a path for both intersections while our previous DQN only handles one at a time.

6.4 Results and discussion

synergies between mpc and dqn

CHAPTER 7

Estimating the uncertainty

ToDo: Rewrite chapter name

RQ 3: How can the quality of a RL agent be improved by accounting for uncertainty?

The motivation of handling uncertainty. In this chapter we present two approches to handling the uncertainty, one is the uncertainty in output of the DQN and the other in the uncertainty of the intention estimation that is feed as an input to the DQN.

7.1 Uncertainty of the decision

Tommy: NN is a black box. The utility value q that come from the DQN works great if it was trained on the

7.1.1 Approach

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 **Garcia2015**. One approach to obtain a variance estimation is through statistical bootstrapping **Efron1982**, which has been applied to the DQN algorithm **Osband2016**. 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 **Osband2018**. 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.,

$$Q_k(s,a) = f(s,a;\theta_k) + \beta p(s,a;\hat{\theta}_k). \tag{7.1}$$

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. **??** becomes

$$\begin{aligned} L(\theta_k) = \mathbb{E}_M \Big[& (r + \gamma \max_{a'} (f_{\theta_k^-} + \beta p_{\hat{\theta}_k})(s',a') \\ & - (f_{\theta_k} + \beta p_{\hat{\theta}_k})(s,a))^2 \Big]. \end{aligned} \tag{7.2}$$

Algorithm 1 outlines the complete ensemble RPF method, which was used in this work. An ensemble of K trainable and prior neural networks are 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. 7.2.

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 $L(\theta_k)$
        $i \leftarrow i + 1$

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

Tommy: 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 previous work, we introduced a confidence criterion that disqualifies actions with $c_\text{v}(s,a) > c_\text{v}^\text{safe}$, where c_safe is a hard threshold **Hoel2020**. 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. **??** for further details.

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

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.,

$$\begin{aligned} &\underset{a}{\operatorname{argmax}} \frac{1}{K} \sum_{k=1}^{K} Q_k(s,a), \\ &\text{s.t.} \quad c_{\mathrm{v}}(s,a) < c_{\mathrm{v}}^{\mathrm{safe}}. \end{aligned} \tag{7.3}$$

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

7.1.2 Simulated experiments

ToDo: show the difference of having the uncertainty estimate not having the estimate

7.1.3 Results and discussion

We show two approaches of handing uncertainty. with an estimate of the uncertainty in actions we showed that it can be used to reduce collisions and risk by choosing another policy than the one trained on data it is not confident in.

7.2 Uncertainty of the intention

In paper A the policy put itself in a position that would not be in conflict with another cars time to intersection and could avoid a lot of the collisions. But the cases the cars collided was when in somehow ended up in a collision course and thats when it had trouble making its way out.

7.2.1 Approach

Because the state is no longer observable, the agent must reason about the history of taken actions and observations. Often, this history can be summarized in a statistic refered to as a belief, or belief state. A belief is a probability distribution over states so that $b : \mathcal{S} \to [0,1]$ and $\sum_s b(s) = 1$, or $\int_s b(s) = 1$ for continuous states.

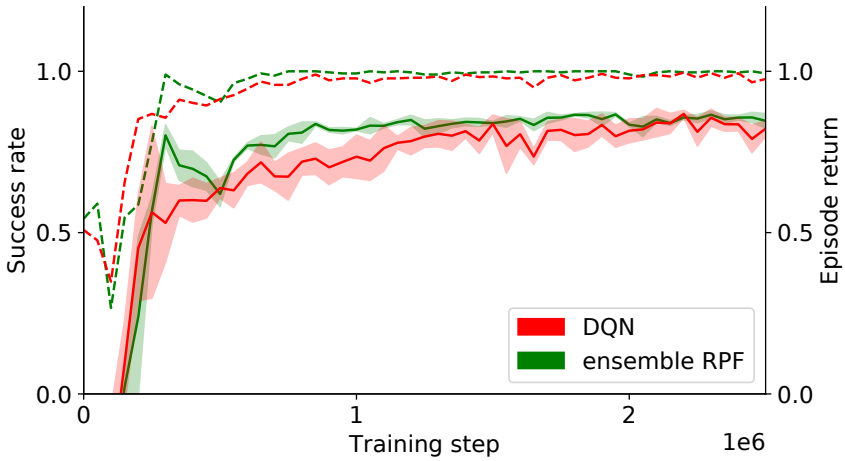


Figure 7.1: Proportion of test episodes where the ego vehicle reached its goal (dashed), and episode return (solid), over training steps for the ensemble RPF and DQN methods. The shaded areas show the standard deviation for 5 random seeds.

7.2.2 Simulated experiments

7.2.3 Results and discussion

We show two approaches of handing uncertainty. with an estimate of the uncertainty in actions we showed that it can be used to reduce collisions and risk by choosing another policy than the one trained on data it is not confident in. The other work show how bad DQN is at handing uncertainty in the input space. The results from the experiments show that the algorithms trained with an estimate from the probability distribution outperformed the algorithm trained with the probability distribution as inputs.

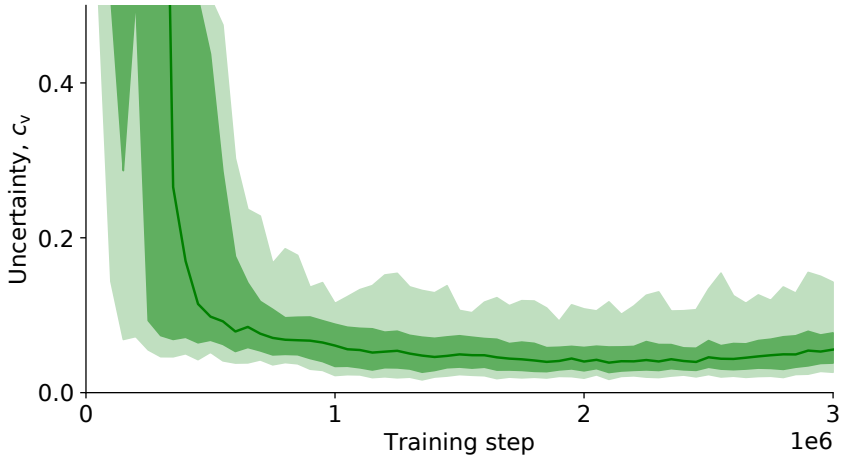


Figure 7.2: Mean coefficient of variation c_v for the chosen action during the test episodes. The dark shaded area shows percentiles 10 to 90, and the bright shaded area shows percentiles 1 to 99.

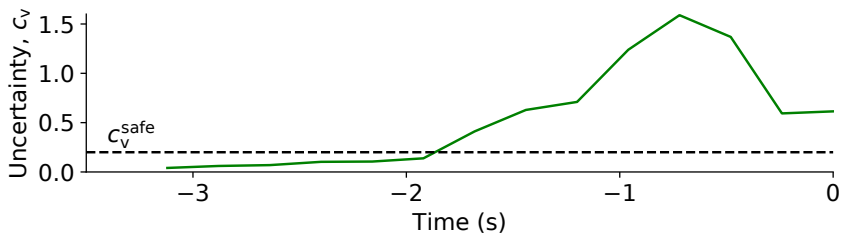
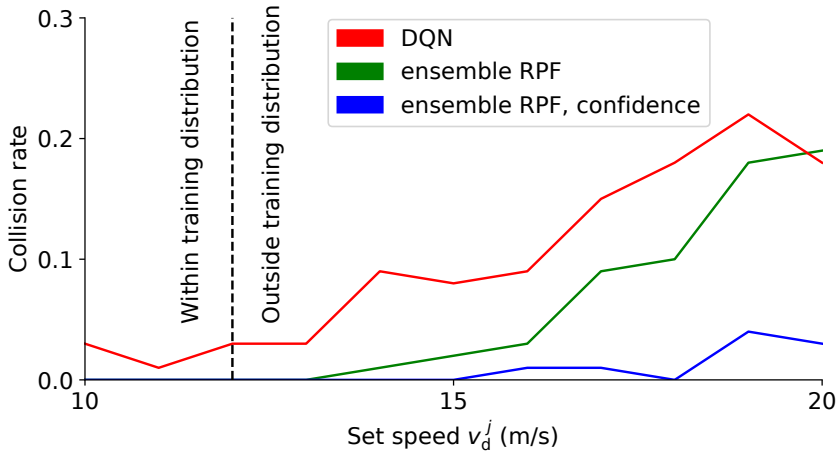
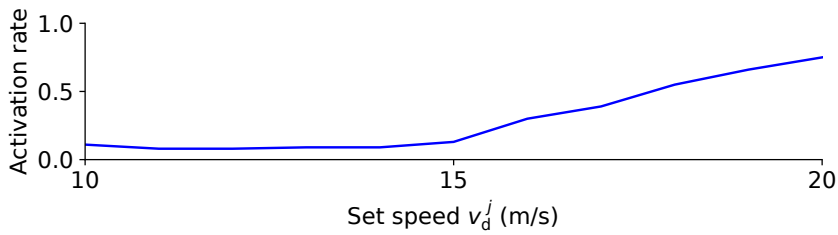


Figure 7.3: Uncertainty c_v during the time steps before one of the collisions in the test episodes, within the training distribution. The collision occurs at $t = 0$ s.

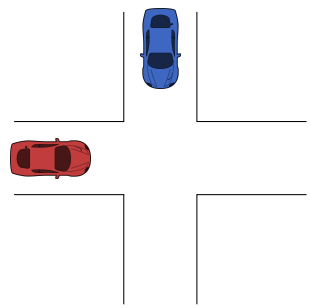


(a) Proportion of collisions.

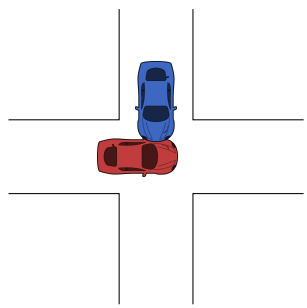


(b) Proportion of episodes where a_{safe} was used at least once.

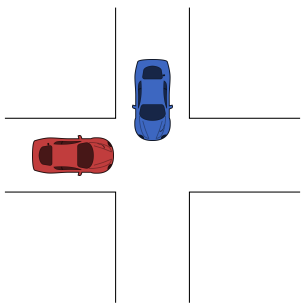
Figure 7.4: Performance of the ensemble RPF agent, with and without the confidence criterion, and the DQN agent, in test episodes with different set speeds v_d^j for the surrounding vehicles.



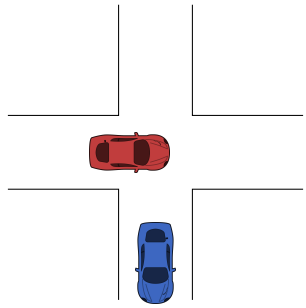
(a) $t = 0$, initial situation.



(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 7.5: 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.

CHAPTER 8

Generalizing over different scenarios

Tommy: Uncertainty in MDP which one are we in?

As mentioned in the Seciton 3.1. The MDP is defined by the tuple (S,A,R,T,γ). The work until now has solved the MDP or POMDP without defining the transition function T by using RL. So what happens when you take a policy trained on one MDP defined by one transition function? Well the short answer is that because DQN uses a NN to approximate the utility Q for taking an action in each state.

Application areas. A transition function is defined as the probability of transitioning from one state to another. This could be different for example when the ego vehicle properties are different, if the DQN is trained on a sports car and then applied on a truck, the difference in acceleration capability would result in a different transition function. Another example would be on the other traffic participants, if the DQN is train in an environment where the driving culture is on the passive side and that is later put in an environment that has a more aggressive driving culture. The DQN/policy would probably not perform so well. This is especially important when you have an intention state directly correlated with the transition. For example in beliefwere the intentions are described on a high level such as take way or give way. An

example is lane changes in Sweden, it is normal to signal first, wait for the other vehicle to slow down before initiating a lane change. While in a high density traffic jam in Paris, it is more normal to show intention by starting a lane change and observe if the other vehicle yields.

8.1 Approach

Some easy solutions would to be have some geo identifier that can choose which policy to use given the country. That may work for an L4 system but could show to be difficult for an L5 system. Our approach is to use transfer RL to identify where we are in the convex hull of MDPs and then choose the policy for MDP we are closest too. Given at set of MDPs, a convex hull is created using the MDPs as the boudries. The goal is then to identity where in the convex hull of MPDs the agent is existing or which MDP is the closest. This does require some number of MDPs to span out the convex hull of models. This can find a policy between MDPs. Identify which MDP is closest. For example Downtown driving in one country may be similar to the driving stile to another.

ToDo: cons: computationally heavy. Have to create the complex hull of MDPs from e set of MDPs.

ToDo: Pros: able to identify which policy to use and scale better by generalizing MDPs instead of countries.

8.2 Simulated experiments

8.3 Results and discussion

FILL

CHAPTER 9

Discussion

FILL

CHAPTER 10

Concluding remarks and future work

10.1 Conclusions

FILL 1. RL by itself still has a long way to guarantee safety, but the methods presented in this paper. the unncertainty can be reduced. safety is better suited for contrl or formal methods. RL is a great tool for creating policy that can adapt to different driver interntions. 2. Even with todays advancements in NN DQN still has a hard time handling

10.2 Future work

FILL

CHAPTER 11

Summary of included papers

This chapter provides a summary of the included papers.

11.1 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%.

11.2 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.

11.3 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.

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.

11.4 Paper D

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

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.

11.5 Paper E

Hannes Eriksson, Tommy Tram, Debabrota Basu, Jonas Sjöberg, and Christos Dimitrakakis
Reinforcement Learning in the Wild with Maximum Likelihood-based Model Transfer
Submitted to Artificial Intelligence and Statistics 2023 (AISTATS), pp. 3169–3174, Nov. 2023.
 DOI: 10.1109/ITSC.2018.8569316.

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.

References

[1] N. Tran *et al.*, "Global status report on road safety 2018," World Health Organization, Tech. Rep., 2018.

[2] S. Singh, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," National Highway Traffic Safety Administration, Tech. Rep. DOT HS 812 506, 2018.

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

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

[7] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015, ISBN: 0262029251, 9780262029254.

[8] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 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 **Bojarski2016EndCars**. 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, **Liebner2012DriverModel**, **Lefevre2012EvaluatingIntentions**. 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 **Shalev-ShwartzSafeDriving** 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 **BrechtelProbabilisticPOMDPs**. 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

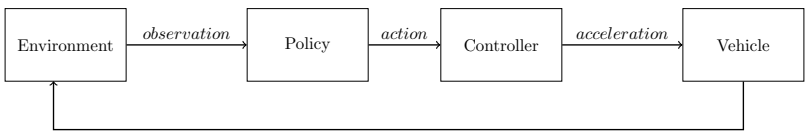


Figure 1: Representation of the architecture

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 **MemonAnalysisManoeuvers**
  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.

3.3 Observations that make up the state

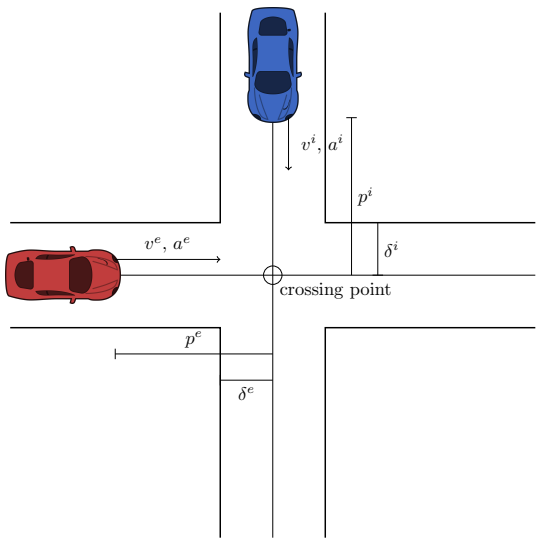


Figure 2: Observations that makes the state

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) **BellmanMDP** 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 **MnihPlayingLearning**, 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 **MnihPlayingLearning** 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 **Tsitsiklis1997AnApproximation**.

Proposed by **MnihPlayingLearning** 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 **Lin1992Self-ImprovingTeaching**.

5.3 Dropout

Overfitted neural networks have bad generalization performance **Hinton2012ImprovingDe** 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 **Srivastava2014Dropout:Overfitting**.

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 **Hochreiter1997LONGMEMORY** 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 **LamplePlayingLearning**.

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 **VanHasseltLearningMagnitude**. 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 action 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

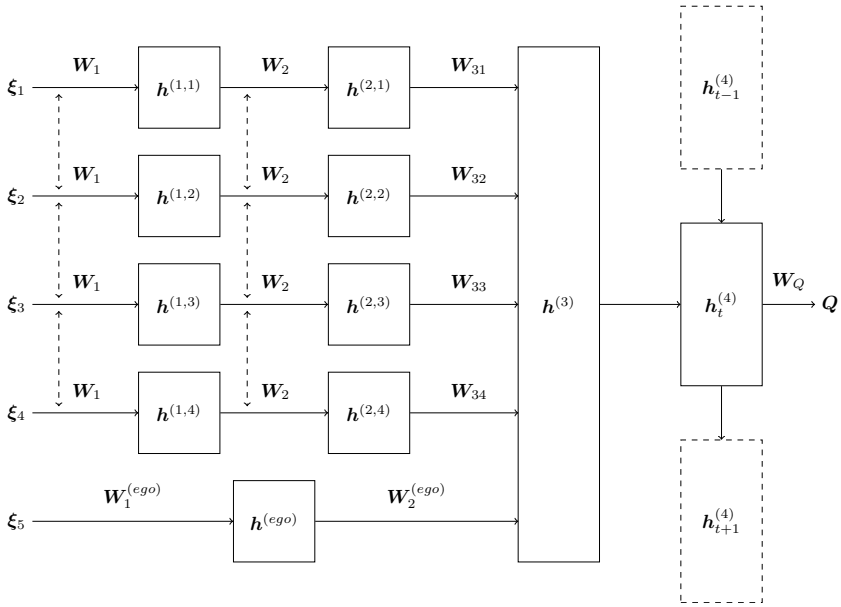


Figure 3: Deep Recurrent Q Network layout with shared weights and a LSTM

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:

- $\xi_1 = [\ p_t^e \quad v_t^e \quad a_t^e \quad \delta^e \quad p_t^1 \quad v_t^1 \quad a_t^1 \quad \delta^1\ ]^T$
- $\xi_2 = [\ p_t^e \quad v_t^e \quad a_t^e \quad \delta^e \quad p_t^2 \quad v_t^2 \quad a_t^2 \quad \delta^2\ ]^T$
- $\xi_3 = [\ p_t^e \quad v_t^e \quad a_t^e \quad \delta^e \quad p_t^3 \quad v_t^3 \quad a_t^3 \quad \delta^3\ ]^T$
- $\xi_4 = [\ p_t^e \quad v_t^e \quad a_t^e \quad \delta^e \quad p_t^4 \quad v_t^4 \quad a_t^4 \quad \delta^4\ ]^T$
- $\xi_5 = [\ a_{t+1}^{e,1} \quad a_{t+1}^{e,2} \quad a_{t+1}^{e,3} \quad a_{t+1}^{e,4} \quad a_{t+1}^{e,5} \quad a_{t+1}^{e,6}\ ]^T$

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

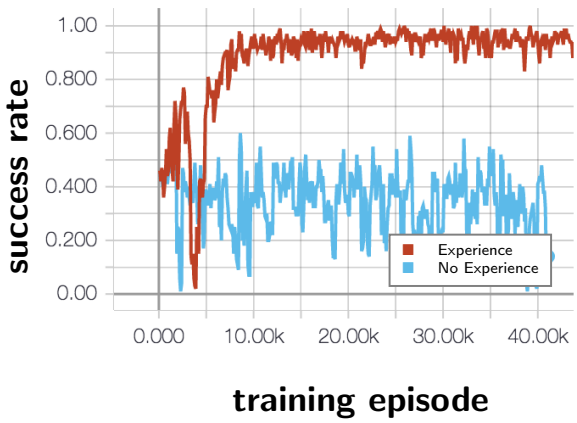


Figure 4: Success rate trend comparing using experience replay (red) and not using experience replay (blue)

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.

7.2 Effect of using Dropout

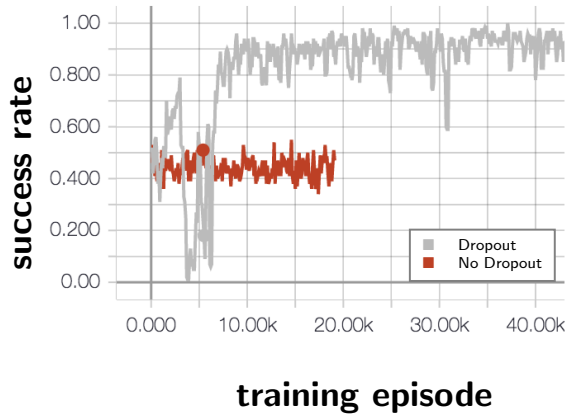


Figure 5: Success rate trend comparing using dropout (grey) and not using dropout (red)

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

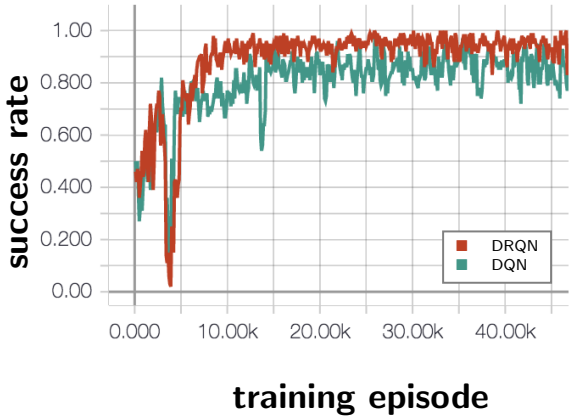


Figure 6: Graphs presenting the performance of a DRQN (red) compared to a DQN with a single observation (green), running on scenarios with cars that have different behaviors. When compared a DRQN succeeds in 3 out of 4 attempts, where a DQN fails.

weights between cars, significantly improved how fast the network converged compared to a fully connected DRQN.

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.

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).

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 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.

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 **Liebner2012DriverModel**. Many advancements aim to bring self driving Level 4 to the market by trying to imitate human drivers **Bansal2018ChauffeurNet:Worst** or predicting what other drivers in traffic are planning to do **Zyner2017LongPrediction**.

Previous research **Tram2018LearningQ-Learning** showed that reinforcement learning (RL) can be used to learn a negotiation behaviour between cars without vehicle to vehicle communication when driving in an intersection. The method found a policy that learned to drive though 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 **BrechtelProbabilisticPOMDPs**

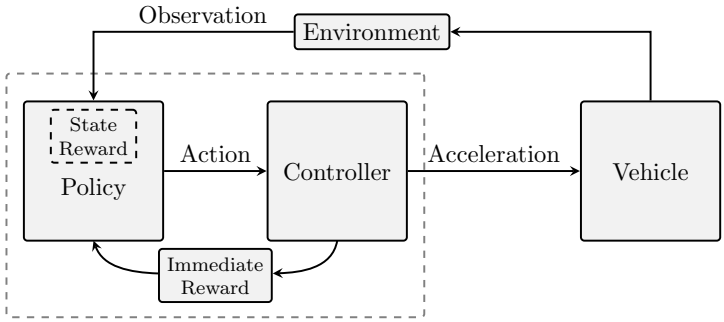


Figure 1: Representation of the decision making architecture

and still be safe **BoutonReinforcementDriving**.

Because the decision policy is separated from the control, high level decision 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 **hult**, 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 **decentralizedMPC** 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 7-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) **BellmanMDP** 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 $o_t \in \Omega$.

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

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. 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 $p_{\text{ego}}^{\text{cross},j}$ 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.

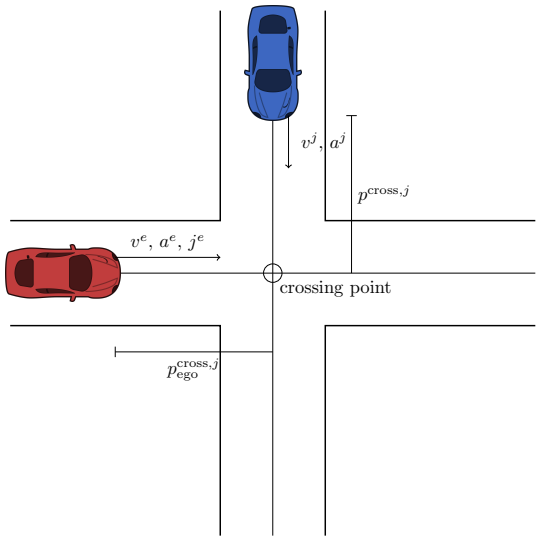


Figure 2: Observations of a scenario

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 **batkovic2019**, we assume that a sensor system 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. **lefevre2014survey**, **batkovic2018**, 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 $p_k^{\mathrm{e}} \leq p_{\mathrm{ego}}^{\mathrm{cross,j}}$. We construct constraints for obstacles $i \neq j$ according to

- if $p^{\mathrm{cross,i}} < p^{\mathrm{cross,j}}$ then $p^{\mathrm{cross,i}} + \Delta \leq p_k^{\mathrm{e}}$, which implies that the ego-vehicle should drive ahead of all obstacles i that are approaching the intersection;
- if $p^{\mathrm{cross,i}} > p^{\mathrm{cross,j}}$ then $p_k^{\mathrm{e}} \leq p^{\mathrm{cross,i}} - \Delta$, which implies that the ego-vehicle should wait to pass obstacle j and other obstacles i;
- if $p^{\mathrm{cross,i}} = p^{\mathrm{cross},j}$ 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 **Tram2018LearningQ-Learning** is introduced.

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

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

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

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

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

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 **Tram2018LearningQ-Learning**.

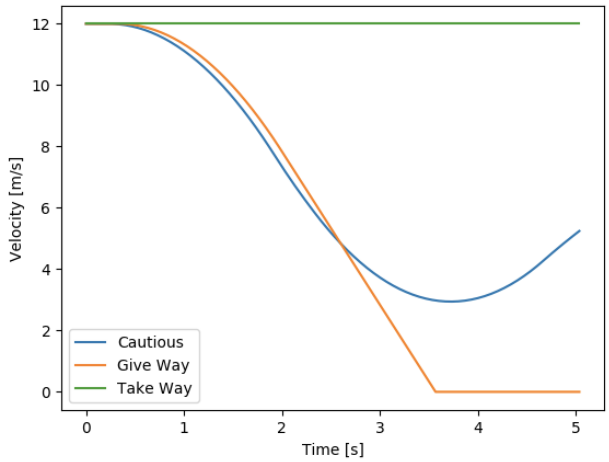


Figure 3: Example of how velocity profile of the different intention agents can look like. All agents has the same starting velocity of 12m/s and are approaching the same intersection

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 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

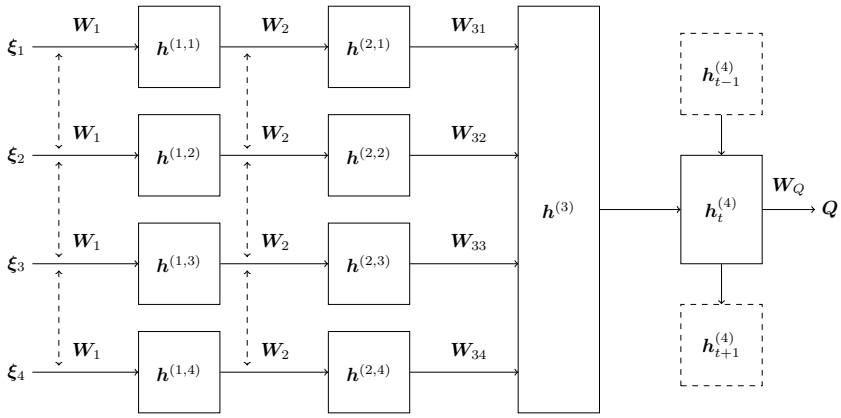


Figure 4: Representation of the network structure

Tram2018LearningQ-Learning 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:

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

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 **Hoel**. 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 an Long Short-Term Memory (LSTM) **Hochreiter1997LONGM** 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 **Mukadam2017** 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 **Tram2018LearningQ-Learning**, 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 **Tram2018LearningQ-Learning**.

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

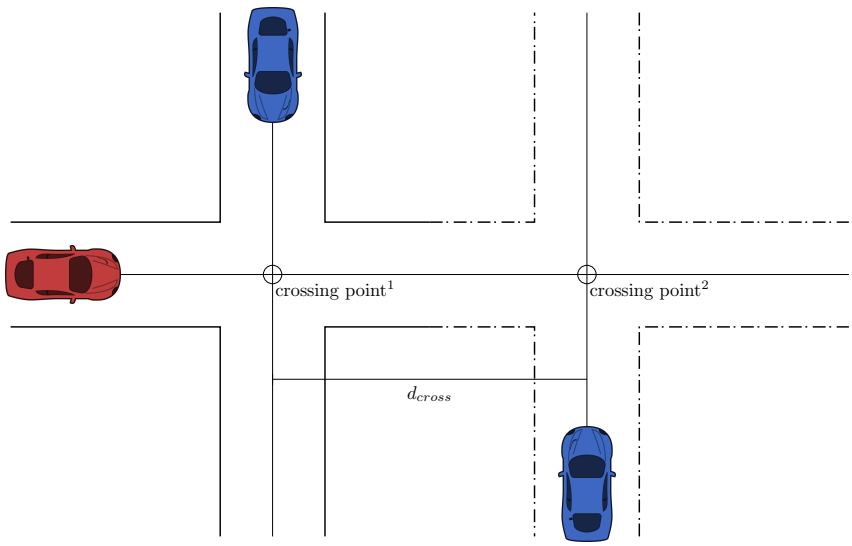


Figure 5: Illustration of a intersection scenario, where the solid line is a single crossing and together with the dashed line creates a double crossing.

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.

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 **VanHasseltLearningMagnitude**, 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

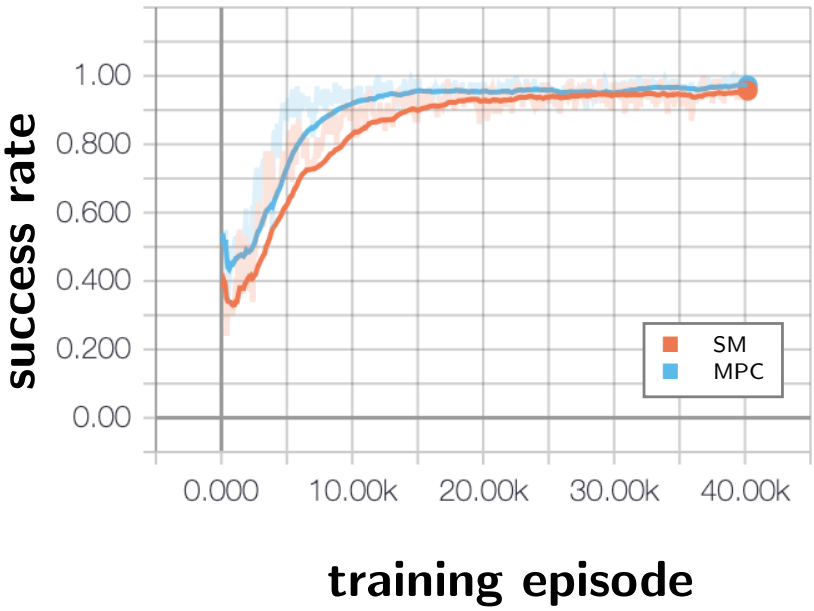


Figure 6: Average MPC and SM success rate for a single corssing after evaluating the policy 300 episodes.

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.

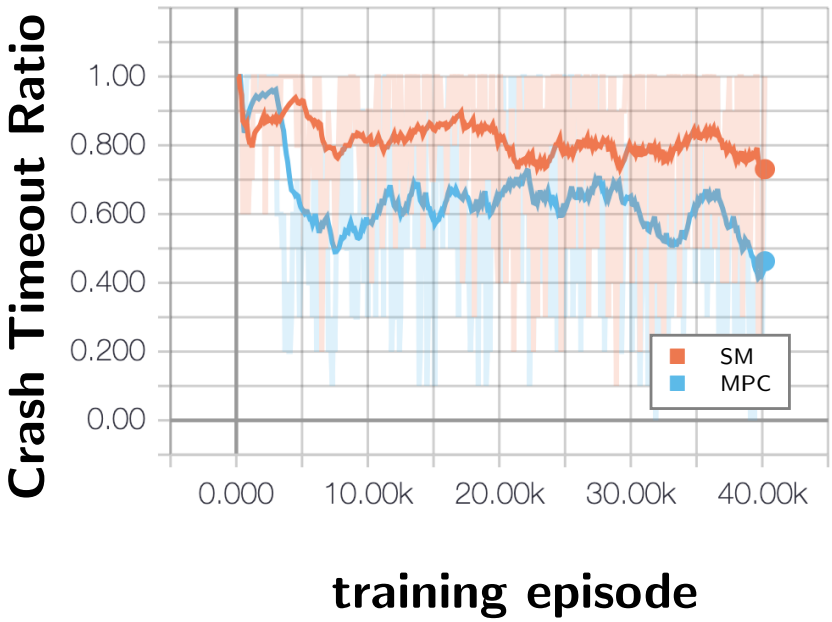


Figure 7: Average MPC and SM crash to timeout ratio for a single crossing after evaluating the policy in 300 episodes. A CTR of 0 means that all failures are timeouts, while a CTR of 1 means that all failures are collisions.

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.

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 **NHTSA**. 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 [1], 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 **Isele2018**, [2], and highway driving **Wang2018**, [3], or a policy gradient method to a lane merging situation **Shalev2016**. Some studies have trained an RL agent in a simulated environment and then deployed the agent in a real vehicle **Pan2017**, **Bansal2018**, and for a limited case, trained the agent directly in a real vehicle **Kendall2017**.

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 **McAllister2017**.

A common way of estimating uncertainty is through Bayesian probability theory [4]. Bayesian deep learning has previously been used to estimate uncertainty in autonomous driving for image segmentation **Kendall2017** and end-to-end learning **Michelmore2018**. Dearden et al. introduced Bayesian approaches to RL that balances the trade off between exploration and exploitation **Dearden1998**. In recent work, this approach has been extended to deep RL, by using an ensemble of neural networks **Osband2018**. 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 **Hoel2020**.

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 [1]. 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 **Watkins1992**, 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. [5].

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 **Garcia2015**. One approach to obtain a variance estimation is through statistical bootstrapping **Efron1982**, which has been applied to the DQN algorithm **Osband2016**. 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 **Osband2018**. 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 2 outlines the complete ensemble RPF method, which was used in this work. An ensemble of K trainable and prior neural networks are

Algorithm 2 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 **Hoel2020**. 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 **tram2019**. 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 $p_{\mathrm{e}}^{\mathrm{c},j}$ 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) [6], with a set

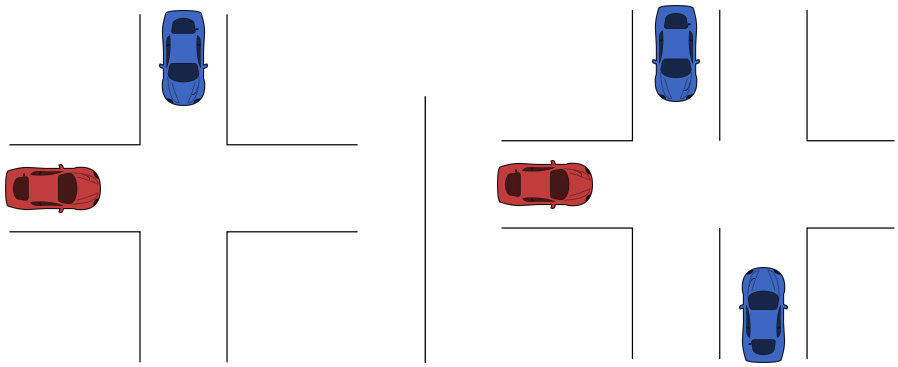


Figure 1: The two intersection scenarios considered in this work; single directional to the left and bidirectional to the right. The agent controls the red car.

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

time gap of $t_{\mathrm{d}}^{j} = 1$ 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) **Kaelbling1998**.

However, by using a k-Markov approximation, where the state consists of the k last observations, the POMDP can be approximated as an MDP [5]. 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 [2]. 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 **Mukadam2017**.

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

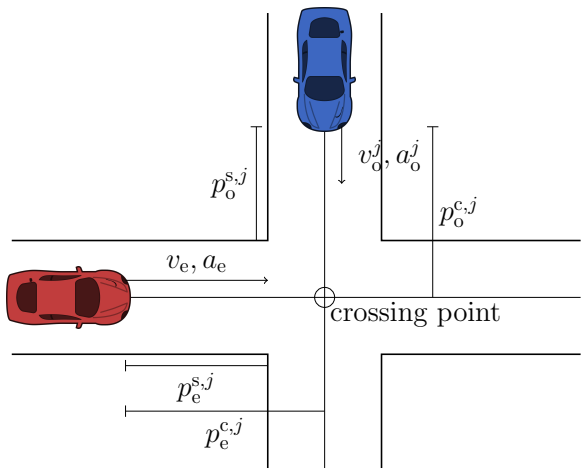


Figure 2: The state space definitions for a single crossing scenario, where subscript e and o denotes ego and other vehicle, respectively.

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 **tram2019**.

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 [3], [2]. 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 $A(s, a)$ **Wang2016**. 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 2 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 $\gamma Q(s', \text{argmax}_{a'} Q(s', a'; \theta_i); \theta_i^-)$ **Hasselt2016**. The Adam optimizer is used to update the weights **Kingma2014**, 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

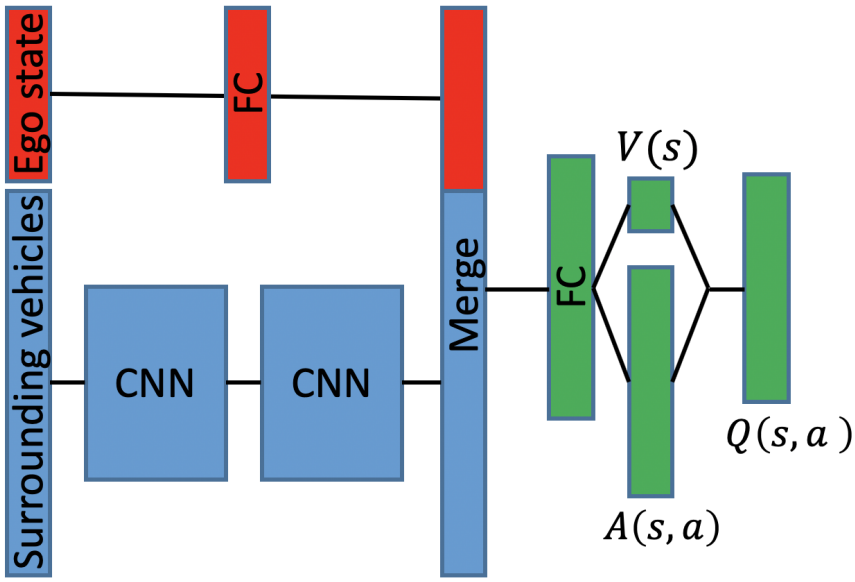


Figure 3: The neural network architecture that was used in this work.

continue forever, which is important, since the terminating state due to the time limit is not part of the MDP [3].

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 2 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

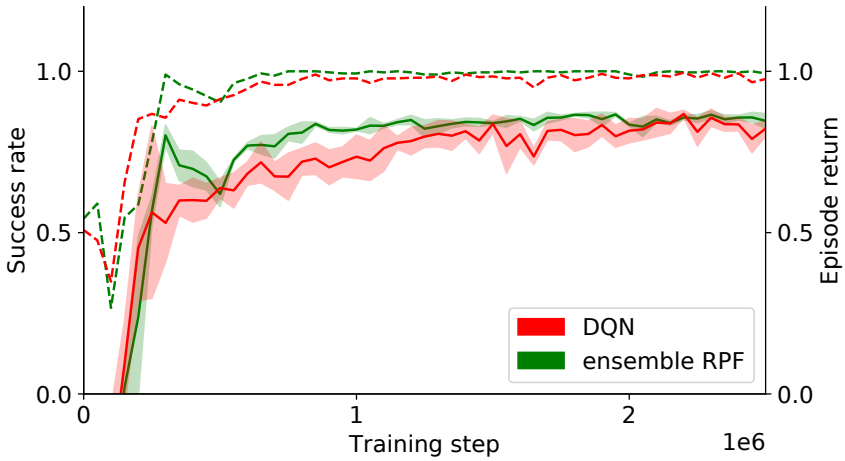


Figure 4: Proportion of test episodes where the ego vehicle reached its goal (dashed), and episode return (solid), over training steps for the ensemble RPF and DQN methods. The shaded areas show the standard deviation for 5 random seeds.

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

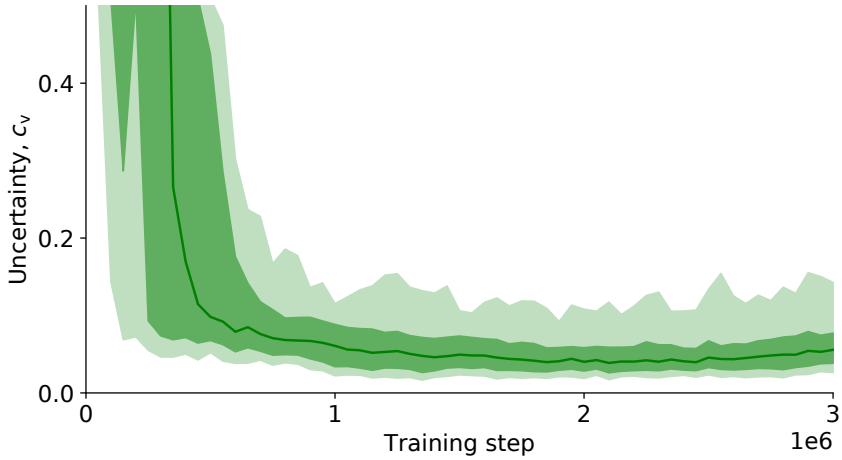


Figure 5: Mean coefficient of variation c_v for the chosen action during the test episodes. The dark shaded area shows percentiles 10 to 90, and the bright shaded area shows percentiles 1 to 99.

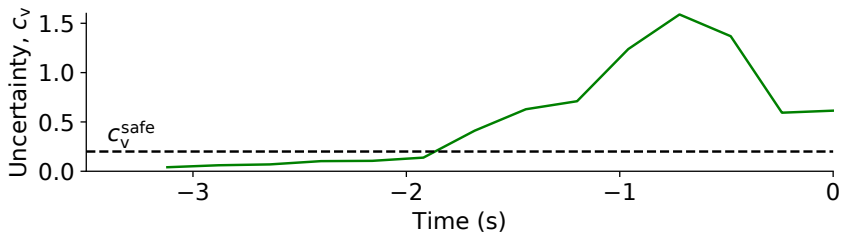


Figure 6: Uncertainty c_v during the time steps before one of the collisions in the test episodes, within the training distribution. The collision occurs at $t = 0$ s.

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.

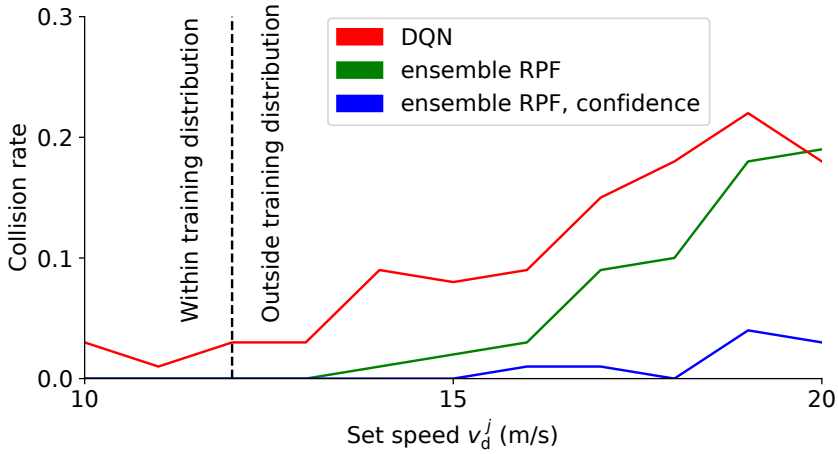
4.2 Outside training distribution

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.

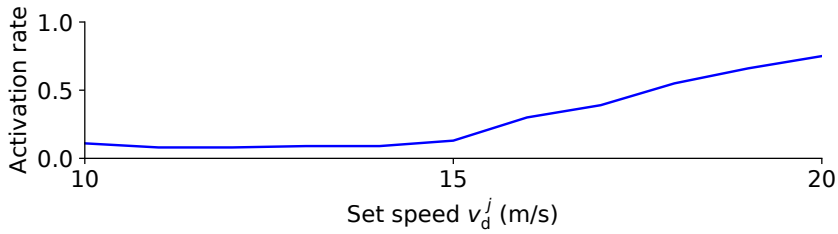
An example of a situation that causes a collision is shown in Fig. 8, where an approaching vehicle drives with a speed of 20 m/s. The Q-values of both the trained ensemble RPF and DQN agents indicate that the agents expect to make it over the crossing before the other vehicle. However, since the approaching 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 **Hoel2020**. 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



(a) Proportion of collisions.

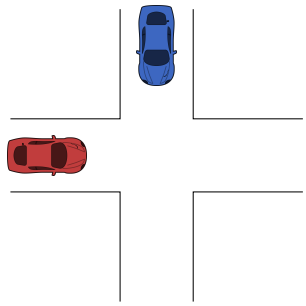


(b) Proportion of episodes where a_{safe} was used at least once.

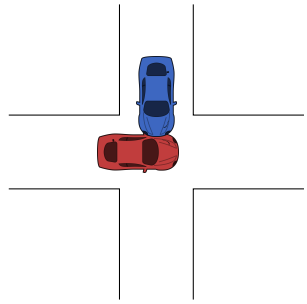
Figure 7: Performance of the ensemble RPF agent, with and without the confidence criterion, and the DQN agent, in test episodes with different set speeds v_d^j for the surrounding vehicles.

networks of the 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.

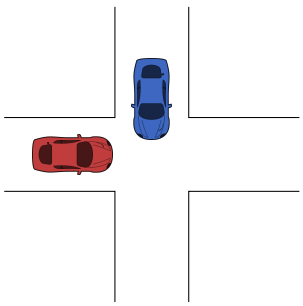
In this work, the estimated uncertainty is used to choose a safe fallback action if the uncertainty exceeds a threshold value. For the cases that are



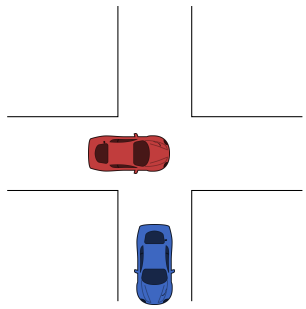
(a) $t = 0$, initial situation.



(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.

considered here, this confidence criterion removes all collisions within the 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 **Underwood2016**. 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] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.

[2] 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.

[3] 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.

[4] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015, ISBN: 0262029251, 9780262029254.

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

[6] 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.

PAPER D

Belief State Reinforcement Learning for Autonomous Vehicles in Intersections

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

Submitted to IEEE Transactions on Intelligent Vehicles,

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

During the last decade, major progress has been made towards deploying autonomous vehicles (AV) and improving traffic safety. Structured traffic scenarios can often be solved by rule-based or planning-based methods, where the surrounding vehicles are assumed to follow a simple driving model. However, according to the Insurance Institute for Highway Safety **IIHS2019**, in 2019, an estimated 115 741 people were injured by drivers running a red light and 928 of them were killed. Hence, to give AV the capability to avoid accidents when other traffic participants do not follow the rules, they need the capability to consider the intentions and predict future actions of other participants. For example, in an intersection or a roundabout scenario, a human driver would observe other vehicles and form a belief of the driver's intention to yield or not and would then act according to this belief. Although AVs have the potential to simultaneously observe more objects than a human, there is a vast number of possible traffic situations, making it difficult for an engineer to anticipate every situation and design a suitable rule-based strategy. Furthermore, both traffic rules and driving styles vary between different countries. Therefore, we focus on approaches that instead learn from experience how to behave in

different situations and adapt to different driving styles. A desirable property of such a learning-based agent is that it accounts for the intentions of the surrounding traffic participants.

Using reinforcement learning (RL) to create a general decision-making agent for autonomous driving has been suggested in many studies during the last few years. For example, **Isele2018empty citation** demonstrated how the deep Q-network (DQN) algorithm could be used to navigate through occluded intersections. **chen2019empty citation** compared the performance of DQN with policy gradient and an actor critic method for different roundabout scenarios. An overview of RL-based studies for autonomous driving is given by **Ye2021empty citation** and **kiran2021empty citation**. Most of these studies both train and evaluate the agents in simulated environments, whereas **Bansal2018empty citation** and **Pan2017empty citation** focus on transferring an agent that has been trained in simulation to drive in the real world. **Kendall2019empty citation** trained in the real world. Game-theoretic approaches to unsignalized intersection scenarios have been proposed by **chen2020empty citation** and **li2021empty citation**. A drawback is that these methods are limited by their models of other driver intentions and the prediction of their future actions. Therefore, we propose using RL to learn a policy based on experience.

The decision-making task of an autonomous vehicle can be formulated as a partially observable Markov decision process (POMDP), e.g., where the intentions of other traffic participants are included in the state space but are not directly observable. **Sunberg2017empty citation** models freeway lane changes as a POMDP with latent internal states and found a policy using Monte Carlo Tree Search. Our previous work [1] showed that it was possible for an agent to solve such a POMDP by finding gaps between cars in an intersection by using a long short-term memory (LSTM), but the intentions were not explicitly estimated. In this paper, we hypothesize that explicitly estimating the intention probability distribution of the other drivers can improve the performance of the policy by reducing the number of collisions and reach the other side of the intersection faster. **Hubmann2017empty citation** proposed a similar POMDP framework for solving the same intersection problem using a particle filter, but they did not use RL. By using RL, the method is more scalable to different environments and has the potential to adapt to real world behaviors through experience.

This hypothesis, that explicitly estimating the intention can reduce the number of collisions, is verified by training and comparing two DQN agents: one with full observability (**DQN FO**) including the true intention as an input feature to the network; and one without intention, that is referred to as **DQN without intention**. While we show in our simulations that observing the true intention leads to performance improvements, it is not a realizable approach in practice since the intention is not observable. We will therefore investigate and answer the following questions: *How can the intention of the other vehicles approaching an intersection be estimated? With a belief state, is it better to train on the full distribution of the belief or use an estimate of the intention? Can the approximation of the neural network compensate for the inaccuracy of the particle filter?*

To answer the first question, a particle filter is proposed in 3.2, which uses a sequence of past observations to generate a belief state. This belief state can then be used, instead of the true intent, to create a policy. In addition to the two baseline DQN agents, four approaches that combine a particle filter with the DQN algorithm are presented in 3.3. The **Q Particle Filter (QPF)** approach uses the belief state, i.e., the full set of particles to estimate the Q-values both during the training and evaluation processes. If the intention is accessible during training the second approach, **QMDP**, is trained under full observability and then tested with the belief state as input. The third approach, **Q Intention Distribution (QID)**, only uses a probability distribution of the non-observable states as an input both during the training and testing processes. Finally, **QMDP-Intention Estimate (QMDP-IE)** is trained under full observability and tested with an estimate of the intention from the particle filter. The detail of each approach are explained in more detail in 3.3 and their performance is evaluated in an unsignalized intersection scenario, explained in 4.

The results show that the intention state improves the policy when comparing the performance between a DQN with access to the true intention and a DQN without the intention state. The QMDP-IE algorithm found the safest policy with zero collisions in the evaluation set, while the QID found the fastest and most aggressive policy.

The main contributions of this paper are:

- Multiple DQN models that capture the belief state at training time either in the form of particle, point estimate of the intention, or the likelihood

of the intention.

- A particle filter approach for estimating the intentions of surrounding traffic participants in an intersection.
- Empirical analysis of the performance of four approaches to explicitly consider the intentions of other traffic participants in an RL framework.

The structure of the paper is as follows. In 2, we review POMDPs and the intelligent driver model (IDM). In 3, we formulate the problem as a POMDP and introduce the particle filter used to generate the belief state along with the algorithms used to solve the POMDP. In 4, we describe the simulation setup, the neural network architecture, and training procedure. 5 compares the different algorithms and our conclusions are presented in 6.

2 Background

This section briefly describes the POMDP framework, the approximation method QMDP, deep Q-learning, and the IDM.

2.1 Partially observable Markov decision process

A POMDP is a mathematical framework used for modeling sequential decision making problems under uncertainty. It consists of a set of states $\mathcal{S}$, actions $\mathcal{A}$, observations Ω, a transition model T, observation model O, reward function R, and a discount factor γ. Together they create the tuple $(\mathcal{S}, \mathcal{A}, \Omega, T, O, R, \gamma)$ that formally defines a POMDP [2]. While in a state $s \in \mathcal{S}$ and executing an action $a \in \mathcal{A}$ the probability of transitioning to a future state s' is described by the transition model $T(s' \mid s, a)$ and the reward r is given by the reward function $R(s, a)$. 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 $b(s) = \Pr(s \mid o_{1:t})$ is the probability of being in state s given observations $o_{1:t}$. 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:

$$b'(s') \propto O(o \mid s', a) \sum_{s \in S} T(s' \mid s, a) b(s). \tag{D.1}$$

In a POMDP, a policy is a mapping from a belief state to an action. A value function $Q^{\pi}(b, a)$ represents the expected discounted accumulated reward received by following policy π after taking action a from belief state b. The goal is to find a policy, that maximizes the expected reward. Finding such a policy is generally intractable [2]. We can use an approximation referred to as QMDP:

$$Q(b, a) \approx \sum_{s} b(s) Q_{\mathrm{MDP}}(s, a) \tag{D.2}$$

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

While finding the optimal $Q(b, a)$ is intractable, finding Q_{MDP} is usually easier. When the transition function can be written explicitly and the state space is finite, we can use dynamic programming to compute Q_{MDP}. Often, the transition function is 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, we can use deep Q-learning to approximate Q_{MDP} **LITTMAN1995**.

In deep Q-learning, the value function is approximated by a neural network. The function approximating the optimal value function is given by minimizing the following loss function:

$$J(\theta) = E_{s'}[(r + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta))^2] \tag{D.3}$$

where θ represents the weights of the neural networks and (s, a, r, s') is obtained from one simulation step in the environment. The weights are updating through gradient updates. In addition, innovations such as the use of a replay buffer and a target network can greatly help the convergence [3].

2.2 Driver model

The general behaviour of cars are modeled using IDM [4] and the acceleration is described by

$$\alpha = \alpha_{\max}\Big(1 - \Big(\frac{v_\alpha}{v_0}\Big)^{\delta} - \Big(\frac{s^*(v_\alpha, \Delta v_\alpha)}{s_\alpha}\Big)^2\Big)$$
$$\text{with } s^*(v_\alpha, \Delta v_\alpha) = s_0 + v_\alpha T_{\mathrm{gap}} + \frac{v_\alpha \Delta v_\alpha}{2\sqrt{a_{\max}\alpha_b}}$$

where v_0 is the desired velocity, s_0 is the minimum distance between cars, T_{gap} is the desired time gap with the vehicle in front, $a_{\max}$ is the maximum vehicle acceleration, and α_b and δ are comfort braking deceleration and a model parameter. The velocity difference Δv_α between the two vehicles and the distance to the vehicle in front s_α determine the acceleration for the next time step. The acceleration can be simplified into two terms, one for free road with no lead vehicle

$$a^{\text{free}} = \alpha_{\max}\Big(1 - \Big(\frac{v_\alpha}{v_0}\Big)^{\delta}\Big) \tag{D.4}$$

and an interaction term for when there is a vehicle in front

$$a^{\text{int}} = -a\Big(\frac{(s_\alpha, \Delta v_\alpha)}{s_\alpha}\Big)^2 \tag{D.5}$$

$$= -a\Big(\frac{s_0 + v_\alpha T_{\text{gap}}}{s_\alpha} + \frac{v_\alpha \Delta v_\alpha}{2\sqrt{a_{\max}\alpha_b}s_\alpha}\Big)^2. \tag{D.6}$$

These equations are used to both model behaviors of other drivers and generate the acceleration for the ego vehicle.

3 Proposed approach

In this section, we define the main problem that this paper addresses, as well as its components. Further, the particle filter used to estimate the belief state is introduced. Finally, we present the RL algorithms that are used to train the agents.

3.1 Problem formulation

The problem setting is a four-way intersection shown in 1. The ego vehicle approaches an intersection with at least one other car on the perpendicular lane with about the same time to intersection assuming a constant velocity model. The goal for the ego vehicle is to reach the other side of the intersection as fast as possible without colliding with the other car. With only onboard sensors on the ego vehicle, it can observe the physical state of other vehicles but not their intention. Such an intersection can be described as a POMDP.

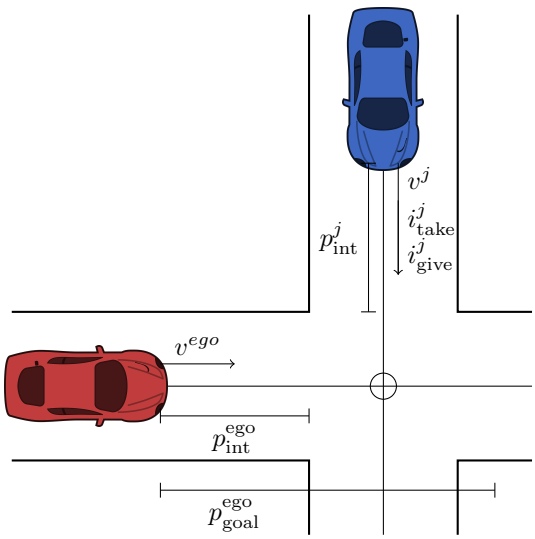


Figure 1: State definitions for the intersection. The red vehicle is the ego vehicle.

State space, $\mathcal{S}$

The state of the system,

$$s = (p^{\text{ego}}_{\text{goal}}, p^{\text{ego}}_{\text{int}}, v^{\text{ego}}, t_{\text{stop}}, \{p^j_{\text{int}}, v^j, i^j\}_{j=1}^N), \tag{D.7}$$

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 stand still $v^{\text{ego}} = 0$. This state tracks the amount of time the ego vehicle has been standing still at the intersection and indicates the time to the terminal states *safe stop* or *deadlock*, which are explained later when we define the reward function. The states of the surrounding vehicles, indexed by $j \in \{1, \dots, N\}$, are described by the distance to the intersection p^j_{int}, the velocity v^j, and the intention i^j. The intentions are represented by a one-hot vector and can be either *give way*, which means that the vehicle will stop before the intersection, or *take way*, which means that the vehicle will drive through the intersection. These intentions control the behavior of the IDM model, described in 2.2.

Observation space, Ω

An observation o consists of the ego vehicle state and the physical state of the surrounding vehicles, while the intentions of the surrounding vehicles i^j are not observed. An observation is described by

$$o = (p_{\text{goal}}^{\text{ego}}, p_{\text{int}}^{\text{ego}}, v^{\text{ego}}, t_{\text{stop}}, \{\hat{p}_{\text{int}}^{j}, \hat{v}^{j}\}_{j=1}^{N}). \tag{D.8}$$

Observation model

The agent observes the ego vehicle states precisely and takes noisy measurements of the positions and speeds of the surrounding vehicles, given by

$$\hat{p}_{\text{int}}^{j} = p_{\text{int}}^{j} + \epsilon_{\text{p}}, \tag{D.9}$$

$$\hat{v}^{j} = v^{j} + \epsilon_{\text{v}}. \tag{D.10}$$

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

Action space, $\mathcal{A}$

The action space $\mathcal{A}$ consists of two high level actions take way and give way, also known as options **SUTTON1999**. The motion of the ego vehicle is controlled by changing the acceleration using the IDM, introduced in 2.2. Depending on the action, the IDM parameters for target vehicle would be different. The take way action uses the free term from D.4, not following any car and just drives though the intersection. The give way action on the other hand uses the interaction term from D.6, with the variable distance s set to the distance to intersection and the velocity difference Δv_α to 0. The action then controls the acceleration of the ego vehicle.

Reward function, R

The reward function is designed to encourage safety and efficiency. The main goal of the agent is to reach the goal on the other side of the intersection without colliding with other traffic participants. There are four terminal states: goal, safe stop, collision, and deadlock. While goal and collision is 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 other vehicles are at

stand still and waiting for the ego vehicle at the terminal state, the deadlock reward is received while the safe stop is received if all other vehicles are in motion while the ego vehicle stopped. The reward function

$$r_t = \begin{cases} 8 & \text{reaching the goal,} \\ 0.4 & \text{safe stop,} \\ -10 & \text{collision,} \\ -0.6 & \text{deadlock,} \\ -0.01 & \text{otherwise} \end{cases}$$

is designed to have a large relative difference between reaching the goal and colliding, with a small incentive for reaching a terminal state faster. The reward for safe stop is positive to account for scenarios where there are many cars and none of them has the intention to yield. Combined with the negative reward for deadlock, the agent is incentivised to not stop unless another vehicle is yielding.

Transition model, T

The state transition probabilities are implicitly defined by the generative simulation model and are not known to the agent. The simulation model is defined in 4.1.

3.2 Belief state estimation using a particle filter

To estimate the state of the environment D.7 from noisy observations, we use a particle filter algorithm that takes advantage of the IDM and assumptions of observation independence. The role of the particle filter in our method is to estimate the position and velocity of the observed vehicles along with their intention (give way or take way) from noisy measurements of position and velocity. This work does not focus on optimize the design of a particle filter but rather to demonstrate how it can be used jointly with a RL agent. In particular, we investigate whether it provides any benefits to perform state estimation at training time and whether the decision making agent can be made more efficient by reasoning about a distribution over intentions instead of just a state.

Previous work has shown that Bayesian filters can be used to infer driver intentions in merging scenarios [5]. One challenge in estimating driver intentions

is that one might need to consider the joint distribution over the intentions of all drivers interacting in the scenario. In this intersection scenario, a joint estimate of intentions for the four closest drivers is created. Each driver is modeled by a position p^j_{int}, a velocity v^j, and a binary intention i^j (give way or take way). Let s^j be the three dimensional state of a car. Considering four vehicles at the intersection, the state estimation algorithm must estimate a 12 dimensional distribution. A particle filter is used to estimate the state because the motion of a vehicle can easily be simulated given its intention.

We make independence assumptions about the other cars that allow us to factor the distribution joint transition model as:

$$\Pr(s^{1:4}_{t+1} \mid s^{1:4}_t) = \Pr(s^1_{t+1} \mid s^1_t) \prod_{i=2}^{4} \Pr(s^i_{t+1} \mid s^i_t, s^{i-1}_t). \tag{D.11}$$

Without loss of generality, it is assumed that the order of vehicles is $1, 2, 3$, and 4. Hence, the position and speed of vehicle 2 only depends on the front vehicle 1 according to the IDM. In addition, the observations of each vehicle are assumed to be independent from each other. Let o^i_t be the observation of vehicle i at time t. The joint observation distribution is:

$$\Pr(o^{1:4}_t \mid s^{1:4}_t) = \prod_{i=1}^{4} \Pr(o^i_t \mid s^i_t). \tag{D.12}$$

Given these two assumptions about the problem structure, we can define the full particle filter procedure. A set of M ordered particles for each observed vehicle are maintained, where a particle representing the joint state can be expressed as follows: $s^{[m]} = (s^{1[m]}, \ldots, s^{4[m]})$. The set of joint particles can be represented by maintaining ordered sets of individual particles for each vehicle. The first particle associated with vehicle 1 will always represent the leading car to the first particle associated with vehicle 2. At each time step, the standard particle filter operations of prediction and measurement updates are performed with a few improvements.

- For all particle indices m, $s'^{[m]} \sim \text{simulate}(s^{[m]})$. Predict the particle one step forward in time using the IDM. The prediction is performed sequentially by updating the leading vehicle first, then its following car, and so on, according to the process described by D.11.
- Compute observation weights for each particle. A Gaussian sensor model was used for the position and velocity. The weight $w^{[m]}$ of a particle

is given by multiplying each of the four individual vehicle weights as described in D.12. The individual weights are assumed to follow a Gaussian observation model:

$$w^{j,[m]} \propto \mathcal{N}([p^j_{\text{int}}, v^j]^T; [\hat{p}^j_{\text{int}}, \hat{v}^j]^T, \text{diag}[\sigma_p^2, \sigma_v^2]). \tag{D.13}$$

- A new set of particles is resampled from the set $\{s'^{[m]}\}_{m=1}^{M}$ weighted by $w^{[m]}$, if the effective number of particles fall bellow a threshold.
- Noise is added to the resampled particles. This is done in two ways. By adding some noise to the acceleration in the prediction model and by letting the intention of each particle have a slight probability p_i to change intention. This noise models the fact that the transition model is not fully known.

At the end of these steps, the set of particles are updated to estimate the current state based on the current observation. The belief of our POMDP agent is represented by the set of resampled particles.

In order to make the particle filter efficient and avoid known issues such as particle depletion, two improvements were incorporated. When a vehicle is observed for the first time, a set of M particles are sampled, where each individual particle is associated with a joint particle. To respect the IDM, the position of a new vehicle is sampled uniformly around the first observed position and the position of its leading vehicle, while the velocity is sampled uniformly in the range described in 1. Intentions are initialized as 50% give way and 50% take way. Finally, the state of the individual particles for this new vehicle is appended to the joint particles of the other vehicles that have just been resampled.

The particle filter implementation is tailored to the problem of intersection navigation. We take advantage of observation independence and the structure of the car following model to efficiently update the joint particles. The particles are used to represent the belief state b, and the algorithms used train the RL agents are described in the next section.

3.3 Belief state reinforcement learning

This section describes the proposed approach to train an RL agent in the belief space. In classical deep RL methods for POMDPs, the learned value

	State s	Particles b	State Approximation D(b)
Training with intention	DQN FO	QMDP	QMDP-IE
Training without intention	DQN without intent	QPF	QID

Figure 2: Table showing the difference between proposed algorithms. The first row are algorithms that have access to the true intention during training, while the algorithms on the second row do not. The columns show the input to the neural network, where the first column contains the baseline algorithms that use the noisy observation from the sensors either with or without the intention i^j. The second column contains algorithms using all M particles as input and the final column uses an estimate of the intention.

function processes a sequence of observations using recurrent neural networks **HausknechtS15drqn** or by concatenating a finite history of observations and feeding it to a simple feed forward neural network [3]. In this approach, a state estimation algorithm is used instead to compute a belief state based on the sequence of past observation and action. The resulting distribution is then fed to a deep RL agent. Instead of processing observations, our agent processes belief states. By learning in the belief space, our agent is more robust to state uncertainty and, contrary to the QMDP approach, it can compensate for some limitations of the state estimation module. This ability is key to learning intention aware policies for autonomously navigating intersections.

All six algorithms in this paper follow the same deep Q-learning structure, shown in 3, with the added subtlety that the architecture of the value function, in step 10 from 3, varies in order to process belief states. The loss function in D.3 takes the form of a mean-squared error with a target corresponding to $r + \gamma \max_{a'} Q(s', a'; \theta)$ and a prediction corresponding to $Q(s, a; \theta)$. Depending on the algorithm presented, the prediction will be performed in a different way.

To confirm the hypothesis that the intention state is necessary, two baseline DQN algorithms are used.

DQN FO: This algorithm is trained with full observability and has access to the true intention at training. As mentioned earlier, intention can not be measured directly, and therefore this algorithm is just used to show the

best policy if the intention approximation is accurate. It also assumes perfect observability of the other state variables.

DQN without intention: Compared to the previous algorithm, this has access to the same states except the intention. By omitting the intention from the input state that is fed to the neural network, this baseline algorithm can be used to show the limit of a policy that does not have access to the intention.

QMDP: The first approach to use the belief state is the QMDP approach described in D.2. The belief state is used at test time but the agent is trained in an environment with full observability in order to estimate Q_{MDP}. The prediction used in the loss function is $Q(s, a)$ and is learned from (s, a, r, s') tuples. The resulting policy can be sensitive to the performance of the state estimation algorithm used to generate the belief state b. To address this issue, we use the true state at training time so that the training procedure matches what the agent will experience at test time.

QPF (Particle Filter): With this algorithm, the agent is trained in an environment with partial observability and uses our proposed particle filter algorithm. The input to the agent is a set of M particles. The loss function is minimized from (b, a, r, b') experience samples and the Q-learning prediction is given by:

$$Q(b, a; \theta) = \frac{1}{M} \sum_{m=1}^{M} Q(s^{[m]}, a; \theta). \tag{D.14}$$

This operation is differentiable and allows training $Q(b, a)$ using deep Q-learning. One challenge with this approach is that the number of particles used to represent the belief can be very large in practice and cause the training to be unstable and very slow. It is challenging to update the individual Q-values by back propagating the loss function computed based on an aggregation of the values of many particles.

QID (Intention Distribution): The agent is also trained in an environment with partial observability and the loss is minimized using (b, a, r, b') samples. To simplify the training procedure compared to QPF, the probability distribution of the intention is used to compute the prediction:

$$Q(b, a; \theta) = Q(\mathrm{D}(b), a; \theta) \tag{D.15}$$

where

$$\mathrm{D}(b) = (p_{\text{goal}}^{\text{ego}}, p_{\text{int}}^{\text{ego}}, v^{\text{ego}}, t_{\text{stop}}, \{\hat{p}_{\text{int}}^{j}, \hat{v}^{j}, \tilde{i}^{j}\}_{j=1}^{N}), \tag{D.16}$$

and the probability distribution of the intention

$$\tilde{i}^j = \sum_{m=1}^{M} w^{j,[m]} [\tilde{i}_{\text{take}}, \tilde{i}_{\text{give}}]^{j,[m]} \tag{D.17}$$

is given by the particle filter, where $w^{j,[m]}$ is the weight of the particle m and $[\tilde{i}_{\text{take}}, \tilde{i}_{\text{give}}]$ is the one-hot vector specifying intention. This operation does not affect the differentiability of the Q function that is trained using the same procedure as QPF. The advantage of this approach is that, at training time, the agent can experience changes in the estimated intention that would match what the state estimator would produce at test time. As a consequence, the agent becomes more robust to the error in intention estimation.

QMDP-IE (Intention Estimate): The last algorithm consists of training an agent on a fully observable MDP just like the QMDP approach. At test time, instead of averaging the Q-values of all the particle, a threshold $i_{\text{threshold}}$ is set on the intention distribution and then use it as the true intention state. Similar to D.16, the particle filter is only used to generate the intention and is represented as a one-hot vector

$$\hat{i}^j = \begin{cases} [0\ 1] & \text{if } \tilde{i}^j_{\text{give}} > i_{\text{threshold}} \\ [1\ 0] & \text{otherwise,} \end{cases} \tag{D.18}$$

where a high value on the threshold $i_{\text{threshold}}$ makes the agent more conservative.

4 Experiments

We present in this section the experiment setup in three steps. First, we describe how the simulator generates the different traffic scenarios. Second, the network architecture is defined. Third, we describe the details of the training procedure.

4.1 Simulator setup

At the start of each episode, up to N vehicles are spawned with initial positions p_0^j distributed along the intersecting lane, a starting velocity v^0, and a desired velocity v^j_{desired}. Each vehicle is spawned with a deterministic policy that represents their intention, which can be either take way or give way as described

Algorithm 3 Training process

Initialize θ randomly
$\mathcal{D} \leftarrow \emptyset$
for nr episodes **do**
    $o \leftarrow$ initiate environment
    $b \leftarrow$ InitializeBelief(o)
    while episode not finished **do**
        if $e \sim \mathcal{U}(0,1) < \epsilon$ **then**
            $a \leftarrow$ random action
        else
            $Q(b,a) \leftarrow$GetActionValues(b, θ)
            $a \leftarrow \text{argmax}_a\, Q(b,a)$
        $o', r \leftarrow$ StepEnvironment(a)
        $b' \leftarrow$ UpdateBelief(b, o', a)
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(b,a,r,b')\}$
        $M \leftarrow$ sample from $\mathcal{D}$
        update θ with SGD and loss $J(\theta)$

in 3.1. The ego vehicle is spawned last with an initial speed v^{ego} and desired speed $v^{\text{ego}}_{\text{desired}}$. The starting position of ego is then set based on the time to intersection τ_{int} of one of the other vehicles. This other car is referred to as conflict car. The initial position of the ego becomes:

$$p_0^{\text{ego}} = v^{\text{ego}} \frac{p_0^c}{v_0^c}, \tag{D.19}$$

where c is the index of the conflict car. This initialization increases the probability that the ego will conflict with at least one other car as shown in 3.

The decision time between decisions steps are $\text{d}t_{\text{decision}}$. For every decision step, the simulator updates the objects states four times with a simulation time $\text{d}t_{\text{sampling}}$, checking for terminal states at each update. The simulator keeps track of the true state of all objects. An observation model is used to add noise to each observation, following **??**. The update function updates the true state s_t. Every time a vehicle in the perpendicular lane crosses the intersection, they are re-spawned at the start of the lane at a random time with new initial states and intention.

The possible terminal states are the following: (i) Goal, ego vehicle reaching

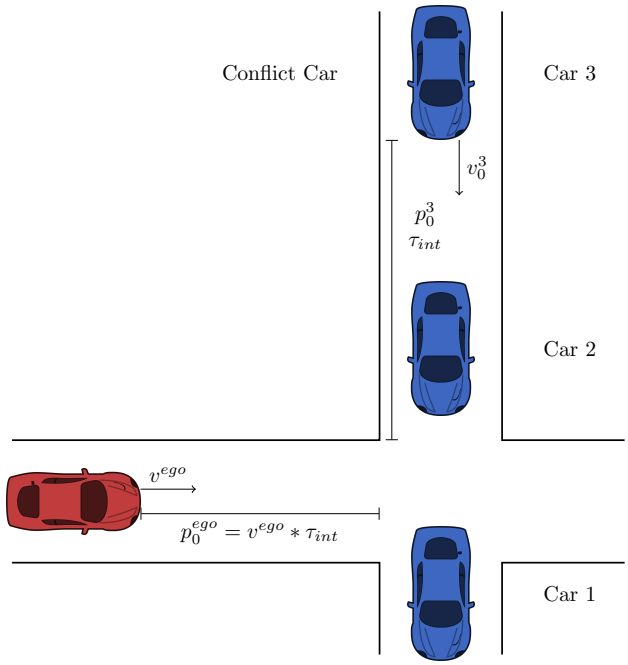


Figure 3: Initial state with three other cars and ego car initial position is set to have the same time to intersection as the conflict car, in this case car 3.

the goal. (ii) Collision, ego collides with one of the other vehicles. (iii) Safe stop, if the ego vehicle has been standing still for more than T_{stop} seconds and no other car is standing still. (iv) Deadlock, if another vehicle standing still at the intersection yielding for the ego vehicle and while the ego vehicle has also been standing still for more than T_{stop} seconds. (v) Timeout, if the total simulation time exceeds T_{lim}. Each of these terminal states, except for the simulation timeout, has a corresponding reward, described in 3.1. The terminal state referred to as collision in this paper may sound drastic, but could also be interpreted as intervention by a collision avoidance system.

4.2 Neural network architecture

The neural network architecture that is used in this study is shown in 4. A previous study [6] introduced applying a convolutional operator to the states

Table 1: Hyperparameters of Simulator

sampling time [s],	$\mathrm{d}t_{\mathrm{sampling}}$	0.5
decision time [s],	$\mathrm{d}t_{\mathrm{decision}}$	2
initial speed [$\mathrm{m\,s^{-1}}$],	v_0^o	$2-7$
initial acceleration [$\mathrm{m\,s^{-2}}$],	a_0^o	0
desired speed [$\mathrm{m\,s^{-1}}$],	v_{desired}^o	$2-7$
time gap [s],	T_{gap}	1.5
Stop time limit [s],	T_{stop}	10
Timeout time [s],	T_{lim}	120
ego spawn speed [$\mathrm{m\,s^{-1}}$],	v^{ego}	5
ego desired speed [$\mathrm{m\,s^{-1}}$],	$v_{\mathrm{desired}}^{\mathrm{ego}}$	5
noise position [m],	σ_{p}	2
noise velocity [$\mathrm{m\,s^{-1}}$],	σ_{v}	1
max observed vehicles,	N	4
IDM max acceleration [$\mathrm{m\,s^{-2}}$],	$\alpha^{\max}$	0.73
IDM deceleration [$\mathrm{m\,s^{-2}}$],	α_b	$0.5-4.0$
IDM acceleration exponent,	δ	4
IDM minimum distance [m],	s_0	2
number of particles	n_b	100
acceleration noise [$\mathrm{m\,s^{-2}}$]	σ_a	0.1
intention switch probability	p_i	5
intention probability threshold	$i_{\mathrm{threshold}}$	0.8
Batch size	B	128
Learning rate	lr	0.0001
Discount factor	γ	0.95
Replay memory size	M_{replay}	20,000
Target network update frequency	N_{update}	1,000

that describe the surrounding vehicles. With this structure, the states that describe each surrounding vehicle are passed through the same weights. The size and stride of the convolutional layer is set to $(4, 1)$, where four equals the number of states that describe each vehicle and one means that the different particles are handled individually. The convolutional layer uses 32 filters with a tanh activation function. To speed up learning, the index j are ordered by distance p_{int}^{j} starting from the vehicle closest to the intersection and a default state used for cars that do not exist.

The ego vehicle states are passed through a fully connected layer of size 32 before being concatenated with the output of the convolutional layer. The output of the concatenated layer is then passed through two fully connected layers of size 32 and finally a fully connected layer of size two gives the Q-values. These values are then merged according to one of the algorithms in 3.3 to form a combined estimate of the Q-values. The belief dimension of the neural network represents the number of particles n_b that are passed through the network. This number varies depending on which algorithm that is used as discussed in 3.3.

4.3 Training procedure

Each network was trained for 200 000 episodes. The loss function of Double DQN is applied, which modifies the maximization operation of D.3 to $\gamma Q(s', \text{argmax}_{a'} Q(s', a'; \theta_i); \theta_i^-)$ **Hasselt2016**. Stochastic gradient decent is used to update the weights **Kingma2014**. The hyper parameters of the training process are shown in 1, and the values were selected by an informal search due to the computational complexity.

If the current policy of the agent do not reach a terminal state before the simulation timeout time T_{lim}, an episode in the real world could continue forever. Therefore, when the simulation reaches the timeout state, the last experience of such an episode is not added to the replay memory. This trick prevents the agent from learning that an episode can end due to a timeout, because the terminating state is not part of the MDP [6].

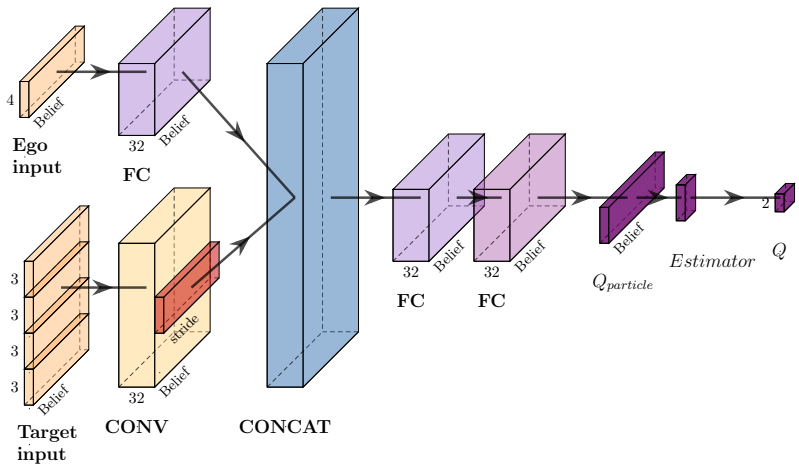


Figure 4: Network architecture, where ego input represents the input features for the ego vehicle, $p^{\text{ego}}_{\text{goal}}, p^{\text{ego}}_{\text{int}}, v^{\text{ego}}, t_{\text{stop}}$, and Target input $\{p^j_{\text{int}}, v^j, i^j\}_{j=1}^N$ for the the target vehicle j. The layers consists of one convolution (Conv) and a total of three fully connected (FC), where the depth of the layers represents the belief size.

5 Results / Results and discussion

This section presents the evaluation metrics and results from the experiments, presented in 2. Although the agents are train and evaluated on scenarios with different number of other cars. Because the scenario with four other cars is the hardest to cross without colliding, it is used to compare the performance of the different agents.

Each agent is evaluated on 1000 episodes. The random seeds are based on the episode number, which generates the same values for the random parameters, defined in 1, for each test scenarios when evaluating different agents. The metrics of interests are the average time it takes for the agent to either 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, from 3.1, where both are considered good outcomes. For the purpose of this paper, it is more important to first find an agent with no collisions and no deadlocks, and then optimize for the time it takes to reach the goal.

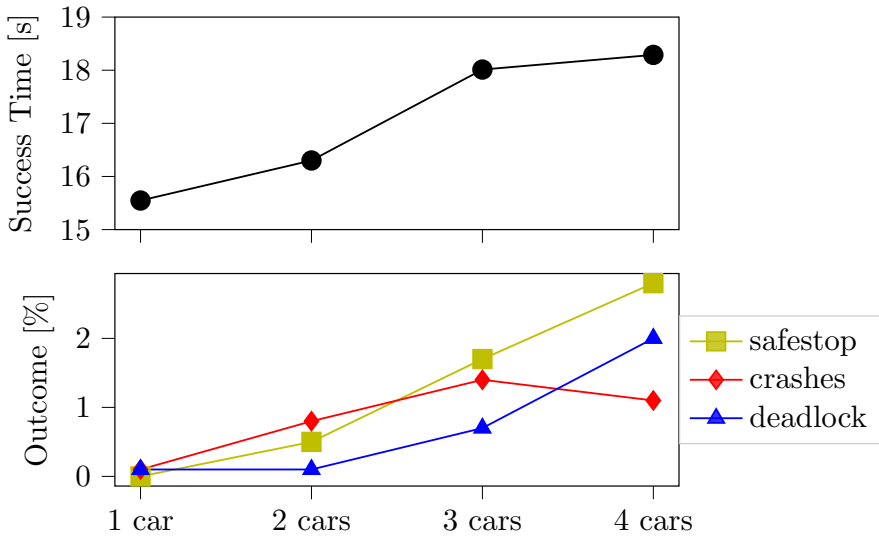


Figure 5: Performance results for DQN without intention algorithm, showing the increasing difficulty for scenarios with different number of cars in the environment.

5.1 Baseline DQN results

The agent trained with full observability, DQN FO, shows how well an agent could perform if it had access to the true intention of other drivers. From 2, the DQN FO agent has 94.2 % goals reached, 5.6 % safe stops, 0 % collisions, 0.2 % deadlocks, and an average success time of 16.93 s. While DQN without intention has 94.1 % goal reached, 2.8 % safe stops, 1.1 % collisions, 2.0 % deadlocks, and reached the goal slower with an average time of 18.29 s. The difference in collision rates confirms our hypothesis that a agent trained with an intention state is better than one without. DQN with intention was both faster to reach the goal and could do it with zero collisions in the evaluation space. All agents are trained with a variable number of other vehicles. 5 shows the performance difference with increasing number of vehicles. In scenarios with only one other vehicle, the agent trained using DQN without intention found a policy that could avoid collisions. However, increasing the number of other vehicles makes the problem harder to solve, which results in an increasing

Table 2: Result summary for scenario $N = 4$ cars

Experiments	Goal reached	Safe stop	Collision	Deadlock	Success time	Training ti
DQN FO	94.20	5.60	**0.00**	0.20	16.93	1d7h
DQN w/o intent	94.10	2.80	1.10	2.00	18.29	1d7h
QMDP	93.70	5.60	**0.10**	0.60	19.95	1d7h
QMDP-IE	94.70	4.80	**0.00**	0.50	17.60	1d7h
QPF	84.70	8.90	3.60	2.80	20.16	2d17h
QID	**97.00**	0.90	2.00	0.10	**16.90**	2d10h

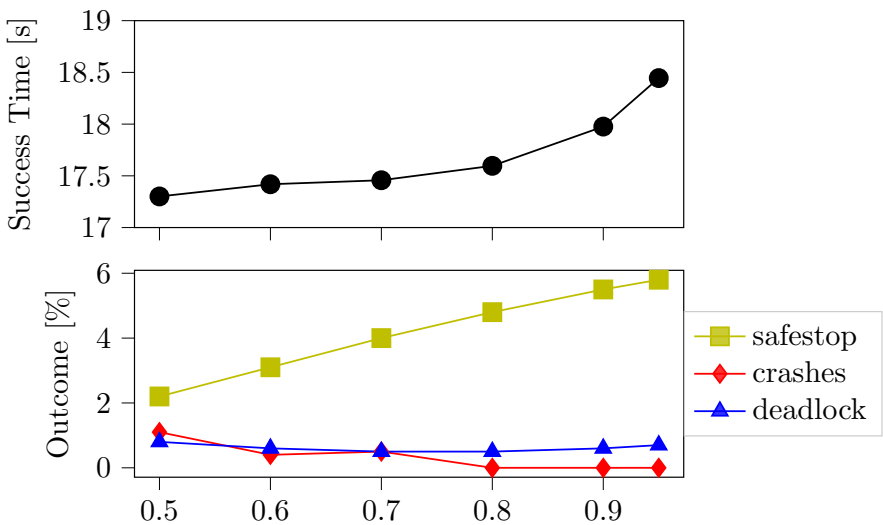


Figure 6: QMDP-IE performance (aggressiveness) for different $i_{\text{threshold}}$ values. Increasing $i_{\text{threshold}}$ lowers the collision rate while at the same time increases the safe Stop rate and the time it takes to reach a success state.

number of collisions and a longer success time.

5.2 QMDP and QMDP-IE results

QMDP and QMDP-IE are both approximation algorithms that use the network trained on DQN FO. Both have similar outcome performance, QMDP reached

the goal 93.7 %, safe stopped 5.6 %, collided only 0.1 %, and got stuck in deadlock 0.6 % with an average success time of 19.95 s. for QMDP-IE, the performance for different threshold values of $i_{\text{threshold}}$ are shown in 6. The lowest threshold $i_{\text{threshold}} = 0.5$ is equivalent to just taking the most likely estimation of the intention and it has the most aggressive behaviour with the lowest percentage of safe stops, highest collisions and the fastest time to reach the goal. When the threshold increases, so does the safe stop rate and the time it takes to reach the goal, while the collision rate decreases and deadlock rates stay about the same. An agent with $i_{\text{threshold}} = 0.8$ was the lowest threshold that achieve zero collisions during testing and is therefore chosen as the threshold value when compared to the other agents in 2. With $i_{\text{threshold}} = 0.8$, QMDP-IE has 43.7 % goal reached, 4.8 % safe stops, 0.0 % collisions, 0.5 % deadlocks, and with an average success time of 17.60 s Compared to DQN without intention, that has a success time of 18.29 s, QMDP is more conservative with a slower success time of 19.95 s, while the QMDP-IE is faster with 16.90 s.

The proposed particle filter performs well at estimating the probability distribution of the intention. However, it relies on an accurate prediction model. It is possible to improve the estimate by increasing the number of particles at the expense of additional computation.

5.3 QPF and QID results

The algorithms that do not have access to the intention during training, QPF and QID, reach the goal much less frequently. QPF has the lowest goal reached rate at 84.7 %, 8.9 % safe stop, the highest collision rate at 3.6 %, 2.8 % deadlocks, and the highest average success time of 20.16 s. QID learned the most aggressive policy, with 97.0 % goal reached, 0.7 % safe stop, 2.0 % collision, 0.1 % deadlock and the fastest average success time of 16.90 s, which is equal to the DQN baseline algorithm. As mentioned in the beginning of the section, the a slightly higher collision rate of 2.0 % classifies this performance as poor compared to QMDP and QMDP-IE, but it is much better than QPF on all evaluation metrics. The higher collision rate could be because of the larger state space, which could create some instability during training. The particle filter was also not fully optimized, because we wanted to investigate if the approximation of the DQN could compensate for the noisy states of the particles. Finally, it should be noted that it required almost two and a half

days to train QPF and QID, which is almost twice as much time required to train the fully observable network.

5.4 Discussion

Is it better to train on the full distribution of the belief or use an estimation of the intention? When comparing QPF and QID to their baselines, it is observed that both algorithms that were trained on the ground truth outperform the algorithms that were trained using the belief state or even just an estimate of the belief distribution. Both QMDP and QMDP-IE had zero collisions in the evaluation space just like DQN FO, while QMDP-IE had a slightly faster policy.

We used a particle filter 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 of the belief in the same way it can approximate the noise. The higher collision rate for the QPF compared to QMDP shows that training on the full set of beliefs can make finding a good policy difficult.

6 shows that the aggressiveness of the policy from QMDP-IE is correlated with $i_{\text{threshold}}$ and by choosing a relatively high value at 0.8 the policy could achieve zero collisions in the evaluation space and in this case could to some extent compensate for the loosely optimized particle filter. The ability to adjust the aggressiveness of the policy by changing $i_{\text{threshold}}$ is also a strength of QMDP-IE compared to our previous black box methods that use an LSTM to estimate the latent state [1].

6 Conclusion

In this paper, we use a particle filter implementation to maintain a belief state to train an RL agent using a DQN. A set of four different algorithms are evaluated with the aim to find the safest and fastest policy that can drive through an intersection crossing traffic participants that has a latent intentions state. Two algorithms, QMDP and QMDP-IE, use a network trained with access to the true intention of the other vehicles but was evaluated with an estimate of their intention. Both algorithms could find a policy with zero collisions in the evaluation space, while QMDP-IE lead to the most efficient

policy. The other two algorithms, QPF and QID, were trained without access to the true intention state and could not achieve a policy without collisions. The QID agent had the fastest average time to reach the goal, but resulted in collision more often than the baseline DQN agent. As for the QPF agent, training on the entire belief state as input yielded the worst performance of all evaluated algorithms. It seems reasonable that the results can be improved by using a network structure that can better learn the latent state. One such candidate is Deep Variational Reinforcement Learning proposed by **Igl2018empty citation**.

References

[1] 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.

[2] M. J. Kochenderfer, *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015, ISBN: 0262029251, 9780262029254.

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

[4] 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.

[5] 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.

[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.

PAPER E

Reinforcement Learning in the Wild with Maximum Likelihood-based Model Transfer

Hannes Eriksson, Tommy Tram, Debabrota Basu, Jonas Sjöberg, and Christos Dimitrakakis

Submitted to Artificial Intelligence and Statistics 2023 (AISTATS), pp. 3169–3174, Nov. 2023.
 DOI: 10.1109/ITSC.2018.8569316.

The layout has been revised.

Abstract

For deploying model-based reinforcement learning in the wild, we aim to compute an efficient policy for an unseen Markov Decision Process (MDP) setting, where we do not have access to an explicit model. But due to availability of either simulators or data for similar tasks, we often can build accurate MDP models for similar problem settings (or tasks), which might differ slightly from the target MDP. In this paper, we study the problem of transferring the available 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 both the 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 also achieves near-optimal performance depending on the similarity of the available MDP models and the target MDP.

1 Introduction

Deploying autonomous agents in the real world poses a wide variety of challenges. As in [**dulac2021challenges**], we are often required to learn the real-world model with limited data, and use it to plan so as 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 [**skirzynski2021automatic**]. 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 knowledge external to the task. As the novel task is unknown, we adopt the Reinforcement Learning (RL) [**sutton2018reinforcement**] 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 uses it to improve performance [**moerland2020model**]. In many applications, the depreciation in performance due to sub-optimal model learning can be paramount. For example, if the agent is interacting with living things or expensive equipment, decision-making with an imprecise model might incur significant cost [**polydoros2017survey**]. In such instances, boosting the model learning by leveraging external knowledge from the existing models, such as simulators, physics-driven engines, etc., can be of great value [**taylor2008transferring**]. 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 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 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 behaviours, traffic rules, signs etc., can be different for each of them. 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 also time-consuming, as both data collection and learning take time. Thus, the company might like to use the models learnt 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 in order to plan efficiently.* We observe that this problem falls at the juncture of *transfer learning* and *reinforcement learning* [**taylor2009transfer**, **lazaric2012transfer**, **laroche2017transfer**]. **lazaric2012transferempty citation** 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 [**taylor2008transferring**]. (ii) *Representation transfer:* a representation of the task, such as learned neural network features, are transferred to perform the new task [**zhang2018decoupling**]. (iii) *Parameter transfer:* the parameters of the RL algorithm or *policy* are transferred [**rusu2015policy**]. 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** approach (MTRL), which encompasses both (i) and (ii) (Section 4).

langley2006transferempty citation 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 7). 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 can allow us to plan almost optimally in the target MDP (Section 6).

Contributions. In brief, we aim to answer the two 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 allows us to perform 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 concretely formulate the problem with Markov Decision Processes (MDPs) setting of RL. 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).

3. *Performance Analysis:* In Section 7, we empirically verify whether MLEMTRL improves the performance for unknown tabular MDPs and LQRs than learning from scratch. MLEMTRL exhibits learning speed improvement and asymptotic improvement for tabular MDPs. In case the of LQRs, it incurs learning speed improvement and jumpstart improvement. We also observe that improvements obtained by using MLEMTRL depend on the similarity between the target and source models. The more similar the target and the source models better is the performance of MLEMTRL, as indicated by the theoretical analysis.

Before elaborating on the contributions, we posit this work in the existing literature (Section 2) and discuss the necessary background knowledge of MDPs and MLEs (Section 3).

2 Related Works

Our work on Model Transfer Reinforcement Learning (MTRL) 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 (DRL). [**zhu2020transfer**] 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. [**rusu2015policy**] uses this approach, where a student learner is combined with a set of teacher networks to guide learning in multi-task RL. [**parisotto2015actor**] develops an actor-critic structure in order to learn how to transfer its knowledge to new

domains. [**arnekvist2019vpe**] invokes generalisation across Q-functions by learning a master policy. Here, *we focus on model transfer instead of policy.*

Another seminal work in TRL, [**taylor2009transfer**] 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** approach to TRL, where the state-action spaces and also dynamics can be different [**atkeson1997comparison**]. [**atkeson1997comparison**] performs model-transfer for a target task with an identical transition model. Thus, the main consideration is to transfer knowledge to tasks with same dynamics but varying rewards. [**laroche2017transfer**] assumes a context similar to that of [**atkeson1997comparison**], 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 questions 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. [**rommel2017aircraft**] describes a multi-task Maximum Likelihood Estimation (MLE) 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 in order to optimise performance for the target MDP (or a target task) than restricting to a mixture of Gaussians.*

The Bayesian approach to multi-task RL [**wilson2007multi**, **lazaric2010bayesian**] tackles the problem of learning jointly how to act in multiple environments. [**lazaric2010bayesian**] 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, [**tamar2022regularization**] 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 a prior or posterior distributions to sample from.

There is another related line of work, namely multi-agent transfer RL [**da2019survey**]. For example, [**liang2023federated**] develops a TRL framework for autonomous driving using federated learning. They accomplish this by aggregating knowledge for independent agents. This setting is significantly different from the general transfer learning but could be incorporated if each of the source tasks were being learned simultaneously as the target task. This requires cooperation among agents and is out of the scope of this paper.

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) [**puterman2014markov**]. 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) [**kalman1960new**]. In tabular MDPs, we employ ValueIteration [**puterman2014markov**] for model-based planning whereas in the LQR setting we use RiccatiIteration [**willems1971least**].

The standard metric used to measure the performance of a policy π [**bell1982regret**] 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) [**casella2021statistical**] 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. [**kiefer1956consistency**] 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 [**newey1987asymmetric**]. 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* [**ciuperca2003penalized**, **ouhamma2022bilinear**]. As we in our work are mixing over known param-

eters, 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 (MTRL) problem and derive a taxonomy of settings that one can encounter 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 completion of 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 put any restrictions on 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 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 error 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 only a finite set of parameters, i.e. the parameters of the source MDPs $\mathcal{M}_s$. We compute the ML estimator $\hat{\mu}$ 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

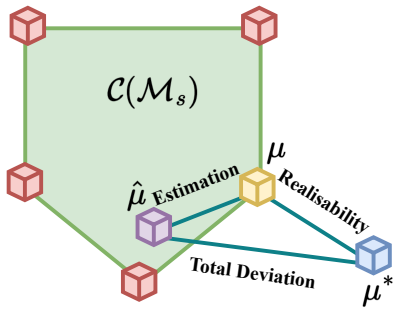


Figure 1: An illustration of the MTRL setting. The source models $\mathcal{M}_s$ are the red boxes. The green area is the convex hull $\mathcal{C}(\mathcal{M}_s)$ spanned by the source models. The target MDP μ^* is displayed in blue and the best proxy model contained in the convex hull μ in yellow. Finally, the estimator of the best proxy model $\hat{\mu}$ is shown in purple.

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, with respect to 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 given by:

$$\hat{\mu} \in \underset{\mu' \in \mathcal{C}(\mathcal{M}_s)}{\arg\max} \log \mathbb{P}(D_t \,|\, \mu'), D_t \sim \mu^*. \tag{E.3}$$

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* [**aitchison1958maximum**]. It implies that, if the parameter corresponding to the target MDP is in $\mathcal{M}'$, it can be correctly identified. In 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 these cases 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 $||\mu^* - \mu||_2$. This allows us to describe the limitation of expressivity of the source models by defining the *realisability gap*: $\Delta_{\text{Realise}} \triangleq \min_{\mu \in \mathcal{C}(\mathcal{M}_s)} \|\mu^* - \mu\|_2$. The realisability gap becomes important while dealing with continuous state-action MDPs with parameterised dynamics, such as LQRs.

5 MLEMTRL: MTRL with Maximum Likelihood Model Transfer

In this section, we will 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 4.

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 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,

Algorithm 4 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 $t = 0, \dots, T$ **do**
  // Stage 1: Model Estimation //
  $\boldsymbol{w}^{t+1} \leftarrow \textsc{Optimiser}(\log \mathbb{P}(D_t \,|\, \Sigma_{\mu_i \in \mathcal{M}_s} w_i \mu_i), \boldsymbol{w}^t), D_t \sim \mu^*$
  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

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,

$$\begin{aligned}\log \mathbb{P}(D_t \,|\, \boldsymbol{p}) &= \log \left(\prod_{(s,a) \in \mathcal{S} \times \mathcal{A}} n^{s,a}! \prod_{i=1}^{|\mathcal{S}|} \frac{p_i^{x_i^{s,a}}}{x_i^{s,a}!} \right) \\ &= \sum_{(s,a) \in \mathcal{S} \times \mathcal{A}} \left(\log n^{s,a}! + \Big(\sum_{i=1}^{|\mathcal{S}|} x_i^{s,a} \log p_i - \log x_i^{s,a}! \Big) \right).\end{aligned}$$

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

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

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.

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & \log \mathbb{P}(D_t \mid \Sigma_{i=1}^m w_i \mu_i), D_t \sim \mu^*, \mu_i \in \mathcal{M}_s, \\ \text{s.t.} \quad & \sum_{i=1}^{m} w_i = 1, w_i \geq 0. \end{aligned} \tag{E.4}$$

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) [**kraft1988software**] 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 4, we compute the next weight vector $\boldsymbol{w}^{t+1}$ by solving the optimisation problem in Eq. E.4. 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 $\mathcal{L}$,

$$\mathcal{L}(\boldsymbol{w}, \lambda, \kappa) = f(\boldsymbol{w}) - \lambda^\top \boldsymbol{w} - \kappa(1 - \mathbf{1}^\top \boldsymbol{w}). \tag{E.5}$$

Next, we replace the full objective in Eq. E.4 by its local quadratic approximation.

$$\begin{aligned} f(\boldsymbol{w}) \approx & f(\boldsymbol{w}^k) + \nabla f(\boldsymbol{w}^k)(\boldsymbol{w} - \boldsymbol{w}^k) \\ & + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^k)\nabla^2 f(\boldsymbol{w}^k)(\boldsymbol{w} - \boldsymbol{w}^k). \end{aligned} \tag{E.6}$$

Here, $\boldsymbol{w}^k$ is the k-th iterate. Finally, taking the local approximation, we define the optimisation problem as:

$$\begin{aligned} &\min_{\boldsymbol{d}} \frac{1}{2}\boldsymbol{d}^\top \nabla^2 \mathcal{L}(\boldsymbol{w}, \lambda, \kappa)\boldsymbol{d} + \nabla f(\boldsymbol{w}^k)\boldsymbol{d} + f(\boldsymbol{w}^k) \\ &\text{s.t. } \boldsymbol{d} + \boldsymbol{w}^k \geq 0, \mathbf{1}^\top \boldsymbol{w}^k = 1. \end{aligned} \tag{E.7}$$

The minimisation gives the search direction $\boldsymbol{d}_k$ for the k-th iteration. Applying this repeatedly and using the construction above ensures that the constraints posed in Eq. E.4 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 in Algorithm 4.

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 using the following equation [**sutton2018reinforcement**].

$$V^*_{\mu^t}(s) = \max_a \sum_{s'} \mathcal{T}^a_{s,s'} \Big(\mathcal{R}(s,a) + \gamma V^*_{\mu^t}(s') \Big). \tag{E.8}$$

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. An LQR-based control system is defined by its system matrices [**kalman1960new**]. Let d_s be the state dimensionality and d_a be the action dimensionality. Then, $\mathbf{A} \in \mathbb{R}^{s_d} \times \mathbb{R}^{s_d}$ is a matrix describing state associated state transitions. $\mathbf{B} \in \mathbb{R}^{s_d} \times \mathbb{R}^{s_a}$ is a matrix describing control associated state transitions. The final two system matrices are cost related with $\mathbf{Q} \in \mathbb{R}^{s_d} \times \mathbb{R}^{s_d}$ being a positive definite cost matrix of states and $\mathbf{R} \in \mathbb{R}^{s_a} \times \mathbb{R}^{s_a}$ a positive definite cost matrix of control inputs. The transition model described under this model is given by,

$$\boldsymbol{s}_{t+1} - \boldsymbol{s}_t = \mathbf{A}\boldsymbol{s}_t + \mathbf{B}\boldsymbol{a}_t. \tag{E.9}$$

When an MDP is mentioned in the context of an LQR system in this work, the MDP is the set of system matrices. Further, the cost (or reward) of a policy π under an MDP μ is given by,

$$V_{\mu}^{\pi} = \sum_{t=0}^{T} \boldsymbol{s}_t^{\top} \mathbf{Q} \boldsymbol{s}_t + \boldsymbol{a}_t^{\top} \mathbf{R} \boldsymbol{a}_t. \tag{E.10}$$

Optimal policy identification can be accomplished using [**willems1971least**], which begins by solving for the cost-to-go matrix $\mathbf{P}$ by,

$$\underset{\mathbf{P}}{\text{solve}} \quad \mathbf{A}^{\top}\mathbf{P}\mathbf{A} - \mathbf{P} + \mathbf{Q} - (\mathbf{A}^{\top}\mathbf{P}\mathbf{B})(\mathbf{R} + \mathbf{B}^{\top}\mathbf{P}\mathbf{B})^{-1}(\mathbf{B}^{\top}\mathbf{P}\mathbf{A}) = 0.$$

Then, using $\mathbf{P}$ the control input $\boldsymbol{a}$ for a particular state $\boldsymbol{s}$ is

$$\boldsymbol{a} = -(\mathbf{R} + \mathbf{B}^{\top}\mathbf{P}\mathbf{B})^{-1}(\mathbf{B}^{\top}\mathbf{P}\mathbf{A})\boldsymbol{s}. \tag{E.11}$$

With some abuse of notation and for compactness, we allow ourselves to write $\boldsymbol{a}_t = -(\mathbf{R} + \mathbf{B}^{\top}\mathbf{P}\mathbf{B})^{-1}(\mathbf{B}^{\top}\mathbf{P}\mathbf{A})\boldsymbol{s}_t$ for $\boldsymbol{a}_t \sim \pi(\boldsymbol{s}_t)$.

6 Theoretical Analysis

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 size of the source MDP set $\mathcal{M}_s$ and the realisability of μ^* under $\mathcal{C}(\mathcal{M}_s)$. A visualisation of the model dissimilarities is available in Figure 1, where $||\mu - \hat{\mu}||$ is the model estimation error, $||\mu^* - \mu||$ is the model realisability error and $||\mu^* - \hat{\mu}||$ the total model deviation error.

Theorem 1 (Performance Gap of Non-Realisable Models)**:** *Let $d = |\mathcal{S}|^2|\mathcal{A}|$ be the dimensionality of the MDP and $\forall_{(s,a)\in\mathcal{S}\times\mathcal{A}}\, 0 \leq r(s,a) \leq 1$. Further, let μ^* be the true underlying MDP, μ, the maximum likelihood $\mu \in \underset{\mu'\in\mathcal{C}(\mathcal{M}_s)}{\arg\min}\, \mathbb{P}(D_{\infty} \,|\, \mu'), D_{\infty} \sim \mu^*$ and $\hat{\mu}$ a maximum likelihood estimator of μ. In addition, let $\pi^*, \pi, \hat{\pi}$ be the optimal policies for the respective MDPs. Let $||\mu^* - \mu||_2 \leq \Delta_{\text{Realise}}$ be the **non-realisability bias** of μ and ϵ be a cover of the convex set $\mathcal{C}(\mathcal{M}_s)$. Then, the performance gap is given by,*

$$|V_{\mu^*}^{*} - V_{\mu^*}^{\hat{\pi}}| \leq \frac{2\gamma\sqrt{d}}{(1-\gamma)^2}(\Delta_{\text{Realise}} + \epsilon), \tag{E.12}$$

this is of order $\mathcal{O}(\sqrt{d}(\Delta_{\text{Realise}} + \epsilon))$.

Proof of Theorem 1. Given the assumptions in Lemma 1 hold true. Then, using triangle inequalities the performance gap can be bounded,

$$\begin{aligned} |V^*_{\mu^*} - V^{\hat{\pi}}_{\mu^*}| &= |V^*_{\mu^*} - V^{\hat{\pi}}_{\hat{\mu}} + V^{\hat{\pi}}_{\hat{\mu}} - V^{\hat{\pi}}_{\mu^*}| \\ &\leq |V^*_{\mu^*} - V^{\hat{\pi}}_{\hat{\mu}}| + |V^{\hat{\pi}}_{\hat{\mu}} - V^{\hat{\pi}}_{\mu^*}| \end{aligned} \tag{E.13}$$

The first term on the right side of the inequality in Eq. E.13 can be bounded using (Theorem 7 **zhang2020multiempty citation**),

$$|V^*_{\mu^*} - V^{\hat{\pi}}_{\hat{\mu}}| \leq \frac{\gamma}{(1-\gamma)^2}||\mu^* - \hat{\mu}||_1. \tag{E.14}$$

Likewise, the second term in the inequality can be bounded using the the general form of the Simulation Lemma (Lemma 4 **kearns2002nearempty citation**),

$$|V^{\hat{\pi}}_{\hat{\mu}} - V^{\hat{\pi}}_{\mu^*}| \leq \frac{\gamma}{(1-\gamma)^2}||\mu^* - \hat{\mu}||_1. \tag{E.15}$$

Using Cauchy-Schwartz and the triangle inequality the distance between the MDPs $\mu^*, \hat{\mu}$ can be bounded, $||\mu^* - \hat{\mu}||_1 \leq \sqrt{d}||\mu^* - \hat{\mu}||_2 = \sqrt{d}\|\mu^* - \mu + \mu - \hat{\mu}\|_2 \leq \sqrt{d}(||\mu^* - \mu||_2 + ||\mu - \hat{\mu}||_2)$. From our assumptions, we know $||\mu^* - \mu||_2 \leq \Delta_{\text{Realise}}$ and $||\mu - \hat{\mu}||_2 \leq \epsilon$. Thus, the final bound of Eq. E.13 is obtained,

$$|V^*_{\mu^*} - V^{\hat{\pi}}_{\mu^*}| \leq \frac{2\gamma\sqrt{d}}{(1-\gamma)^2}(\Delta_{\text{Realise}} + \epsilon), \tag{E.16}$$

which yields us the result. □

Remark (Performance Gap in the Realisable Setting)**:** A trivial worst-case bound for the realisable case (Section 4.2) can be obtained by setting $\Delta_{\text{Realise}} = 0$ as by definition in that case $\mu^* \in \mathcal{C}(\mathcal{M}_s)$.

7 Experiments

To benchmark the performance of MLEMTRL, we compare ourselves to a posterior sampling method (PSRL) [**osband2013more**], equipped with a combination of product-Dirichlet and product-NormalInverseGamma priors for the tabular setting, and Bayesian Multivariate Regression prior [**minka2000bayesian**] 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 [**willems1971least**] to obtain the optimal linear controller for the sampled model. The experiments are deployed in Python 3.7, with support of SciPy [**virtanen2020scipy**], and ran on a i5-4690k CPU and a GTX-960 GPU.

The objectives of our empirical study are two-fold:

1. How does MLEMTRL using impact performance in terms of **learning speed**, **jumpstart improvement** and **asymptotic convergence** compared to our baseline?

2. What is the performance loss of MLEMTRL with increasing **model dissimilarity** between the estimated model and the target model?

We conduct two kinds of experiments to verify our hypotheses. Firstly, in Figure 3, we compare the performance of MLEMTRL to PSRL [**osband2013more**] in terms of learning speed and jumpstart improvement. We also identify a zero or near-zero regret for MDPs satisfying the realisability conditions in Section 4.2. Lastly, we see a trend of increase in regret with an increase in *Kullback-Leibler divergence* of the true MDP μ^* from our estimate, $\hat{\mu}$. In another experiment, in Figure 4, we further verify the relation between increase in divergence and increase in regret. We begin by recalling the goals of the transfer learning problem [**langley2006transfer**].

Learning Speed Improvement: A learning speed improvement would be indicated by the algorithm reaching its asymptotic convergence with less data.

Asymptotic Improvement: An asymptotic improvement would mean the algorithm converges asymptotically to a superior solution to that one of the baseline.

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.

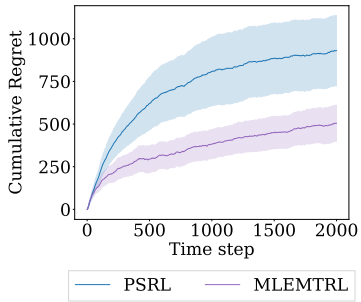


Figure 2: This visualisation is of multiple variants of the Chain environment. Here, the slippage parameter is being varied. The cumulative regret is shown over the first 2000 time steps and the shaded regions represent the 95% confidence interval of the cumulative regret for the two algorithms. The agents update their policies every 20 steps. For regret, lower values are better.

7.1 RL Environments

Chain. A common testbed for RL algorithms in tabular settings is the Chain [**dearden1998bayesian**] environment. In it, there is a chain of states where the agent can either walk forward or backward. At the end of the chain, there is a state yielding the highest rewards. At every step, there is a chance of the effect of the opposite action occurring. This is denoted as the *slipping probability*. The slippage parameter is also what is used to create the source models, in this case, those parameters are $\{0.01, 0.20, 0.50\}$. For both algorithms, we use a product-NormalGamma prior over the reward functions. For PSRL, we use product-Dirichlet priors over the transition matrix.

CartPole. The environment used to testbed our algorithm is the *Cartpole* environment [**barto1983neuronlike**]. The task is to stabilise a pole attached to a cart and keep it upright for the total episode length of 200 time steps. The state-space $s \in \mathbb{R}^4$ and the action-space $a \in [-1, 1]$. A continuous action applies a bounded force in the opposite direction on the cart. The original environment parameters are the *gravity* $g = 9.8$, *mass of cart* $m = 1.0$, *mass of pole* $p = 0.1$ and *length of pole* $l = 0.5$. In order to construct our source tasks, we create a variation of these parameters in the following way,

ToDo: Fix kbordermatrix

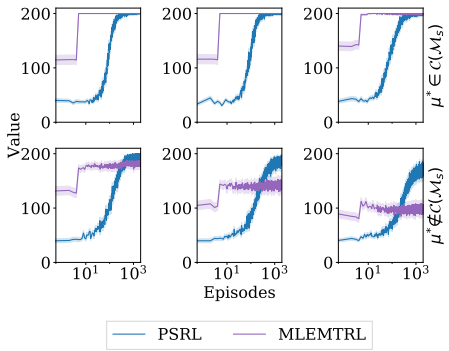


Figure 3: We compare MLEMTRL against PSRL in terms of convergence speed and early performance. The value functions of the algorithms are plotted against the log of the number of episodes. The shaded area is the 68% confidence of the mean over multiple MDPs with the same model dissimilarity. In the topmost row, all of the true MDPs are within the convex hull $\mathcal{C}(\mathcal{M}_s)$. In the bottom row, the MDPs are outside. As you go from top-left to bottom-right, the divergence from the true model to the average model in the convex hull increases. For utility, higher values are better.

These source tasks are then used to transfer knowledge from to the novel task. The maximum attainable return in this environment is 200 which happens if the agent is able to balance the pole for the full episode length. In this setting, the MLEMTRL algorithm effectively creates a mixture of linear-Gaussians as the full likelihood model and obtains the mixture weights by using experience from the novel task.

7.2 Impacts of Model Transfer with MLEMTRL

Asymptotic and Learning-speed Improvements in Tabular MDPs. We begin by evaluating the proposed algorithm in the Chain environment. The results of said experiment is available in Figure 2. In it, we evaluate the performance of MLEMTRL against PSRL for the first 2000 time steps. The experiments are done by varying the slippage parameter $p \in [0.01, 0.50]$ and the results are computed for each different setup of Chain from scratch. The cumulative regret is taken w.r.t the optimal policy and the shaded regions represent the

95% confidence of the cumulative regret at the time step. The confidence interval is taken over the various variations of the Chain environment. The image clearly shows MLEMTRL has a jumpstart improvement coming as a result of the transfer learning scheme but also an indication of a learning speed improvement.

Learning-speed and Jumpstart Improvements in LQRs. In the experiment depicted in Figure 3, we investigate the convergence rate and the jumpstart improvement of the MLEMTRL algorithm on 100 independent target MDP realisations at six different levels of divergence. The divergence is measured from the centroid of the convex hull to the target MDP. Further, in the topmost row, all of the target MDPs belong to the convex hull of source models. As we can see, in this setting, identification of the true model occurs rapidly. One reason for this is because of the near-determinism of the environment. Compared to the agent learning from scratch, we observe zero-regret with faster convergence. As we go from top-left to bottom-right, the divergence increases. For the bottom-most row, we can again observe a faster learning rate. In this case, the degradation in performance increases with the divergence, resulting in poor performance in the final case. The experiment demonstrates that under the TRL framework, we require that the source models are not too dissimilar from the target model.

7.3 Impact of Realisability Gap on Regret

In the final experiment, we further illustrate the observed relation between model dissimilarity and degradation in performance. Figure 4 depicts the regret against the KL-divergence of the target model to the best proxy model in the convex set. As we can see, there is a clear connection between model dissimilarity and the performance gap in the proposed algorithm. This is also justified in the Section 6 where the bounds have an explicit dependency on the model difference. Note that in the 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.

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 **langley2006transferempty citation**. In our first experiment, we see a clear superiority in terms of learning speed and jumpstart improvement.

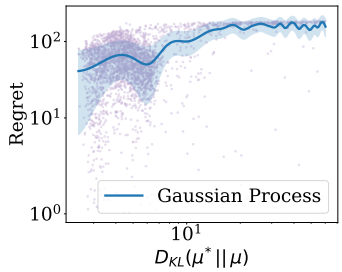


Figure 4: A log-log plot depicting the regret against the KL-divergence between the true MDP and the best proxy model. Only the non-zero regret results are displayed to indicate how sub-optimal performance relates to model dissimilarity. The thick blue line is a Gaussian Process regression model fitted on the observed data (in purple). The shaded area is the 68% confidence interval of the mean.

These results are consistent for the second experiment as well, given that the true MDP is similar enough to the source tasks. In the second and third experiment we verify the claim that a loss in performance (in terms of regret) has a strong dependence on the dissimilarity of the estimated MDP and the true MDP.

8 Discussions and Future Work

In this work, we aim to answer the 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 allows 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 the way of a maximum likelihood procedure, which resulted in 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 7.2 and the theoretical results in Section 6. We can clearly see the model allows for generalisation to novel tasks, given that the tasks are similar enough to the existing task.

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 relation of the *regret* w.r.t the model dissimilarity in terms of KL-divergence and the quick, near-optimal performance of the algorithm in the case where the new domain is similar. In addition to this, we prove worst-case performance bounds of the algorithm in both the realisable and non-realisable settings.

Future work. An interesting future work is to derive tighter bounds for the *realisable* case in Section 4.2. Note $\delta = 0$ in the realisable setting but the large constant ϵ remains. A concentration bound in the number of samples from the true MDP could be constructed using the proof techniques in **anastasiou2019normalempty citation** and **ouhamma2022bilinearempty citatio**
We would also like to investigate larger and more difficult problems. One challenge is devising tractable models in such cases, and to be able to do optimal model-based planning for large scale problems. We would start by considering large problems that can be written as LQR problems, such as [**tunyasuvunakool2020dm_control**] and Mujoco [**todorov2012mujoco**]