Falsification of Autonomous Driving using Reinforcement Learning

Master's thesis in Complex Adaptive Systems

ANDREAS SPETZ & AMANDA ABEDAHAD

DEPARTMENT OF ELECTRICAL ENGINEERING

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

Master's thesis 2022

Falsification of Autonomous Driving using Reinforcement Learning

ANDREAS SPETZ & AMANDA ABEDAHAD

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2022

Falsification of Autonomous Driving using Reinforcement Learning
ANDREAS SPETZ & AMANDA ABEDAHAD

Supervisor: Yuvaraj Selvaraj, Zenseact
Dapeng Liu, Zenseact
Constantin Cronrath, Department of Electrical Engineering
Examiner: Martin Fabian, Department of Electrical Engineering

Master's Thesis 2022
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Falsification of Autonomous Driving Using Reinforcement Learning
ANDREAS SPETZ
AMANDA ABEDAHAD
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Every year, approximately 1.35 million people die as a result of road traffic accidents. Autonomous vehicles can potentially improve road safety by reducing traffic accidents. However, the safety validation of autonomous vehicles is one of the key challenges in their successful commercial deployment. One method of testing vehicle safety is falsification, which attempts to provide evidence of failure by finding counterexamples to a specification.

In this thesis, we investigate the possibilities and limitations of using reinforcement learning in falsification of an autonomous vehicle performing lane changes. The goal is to find critical scenarios where the decision-making of the autonomous vehicle may cause a dangerous lane change. To this end, we propose a framework for falsification of autonomous driving using Deep Double Q-Network. Firstly, we show that the framework is successful in finding critical lane change scenarios by altering the behavior of one adversarial vehicle. Secondly, we show that the framework can be used to train two agents in a multi-agent system in order to expand the number of unique scenarios to be found. Lastly, by reusing the policy obtained in the first experiment, we show that lane change decision-making can be improved to reduce crashes.

Keywords: Falsification, Deep Reinforcement Learning, Autonomous driving.

Acknowledgements

We want to thank our supervisors Yuvaraj Selvaraj, Dapeng Liu, and Constantin Cronrath for all the great feedback and guidance throughout our whole project. We also thank Zenseact for supporting our work. Finally, we thank our examiner Martin Fabian.

Andreas Spetz & Amanda Abedahad, Gothenburg, June 2022

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

A3C	Asynchronous Advantage Actor-Critic
AD	Autonomous Driving
AST	Adaptive Stress Testing
BTN	Brake Threat Number
CPS	Cyber Physical System
DDQN	Double Deep Q-Network
DRL	Deep Reinforcement Learning
MDP	Markov Decision Process
PID (-controller)	Proportional–Integral–Derivative controller
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RSS	Responsible-Sensitive Safety
STL	Signal Temporal Logic

Contents

List of Figures

List of Tables

1

Introduction

Every year, approximately 1.35 million people die as a result of road traffic accidents [1]. To reduce this number, autonomous vehicles could be a potential solution due to the possibility of eliminating human error. In addition, traffic congestion and environmental issues such as fuel consumption and pollution are estimated to be substantially reduced as autonomous vehicles enter the commercial market [2].

However, there exists several challenges that need to be solved before this technology is ready for commercial release [3]. One key challenge is to ensure that the vehicles are safe - an autonomous vehicle is required to behave in a way resulting in a magnitude of fewer crashes than a human driver [4]. With these safety requirements, there is an emerging necessity for effective verification methods. Two of the main verification approaches used in industry today are real car testing and simulation-based testing. Usually, to obtain reliable validation results using real car testing, the car needs to drive a substantial amount of kilometers, close to one billion. This approach is therefore classified as extremely time-consuming and expensive [5]. Consequently, this stresses the importance of reliable simulations and software validation approaches.

Previous research has shown promising results adopting scenario-based testing. One method using scenario-based testing is *falsification* [6]. Given some safety requirements for the vehicle, falsification aims to find evidence for the presence of counterexamples violating these requirements by a systematic search for failure scenarios. The search is formulated as an optimization problem, where the objective function measures how far a formal specification is from being falsified [7]. Various optimization algorithms have been proved to yield successful results, where one is reinforcement learning (RL) [8].

Recent research conducted at Zenseact [9] introduces a new method for a safe lane change in a highly dense traffic situation, using RL. The contribution consists of shielding the *host* agent from taking unsafe actions. Consequently, the safety shielding resulted in more efficient training. The surrounding adversarial vehicles, the *ado* vehicles, are controlled by Proportional-Integral-Derivative controllers (PID controllers), programmed to mimic human driving. This is mainly done by adjusting their acceleration to maintain a safe distance from the surrounding vehicles.

1.1 Aim

This thesis aims to research the possibilities and limitations of using deep reinforcement learning (Deep RL) in the falsification of autonomous driving. To achieve this, a Double Deep Q-Network (DDQN) algorithm will be implemented to control adversarial agents to perform falsification. This project will use the simulation environment presented in [9], further described in Section 2.5. In [9], the adversarial agents were PID-controlled vehicles designed to mimic human driving in a dense traffic situation. With every adversarial vehicle driving according to this behavior, the safety of the lane changing was only tested in scenarios where the traffic was flowing normally. This project aims to perform more rigorous testing of the safety shielding algorithm by trying to find edge case scenarios where the host vehicle's decision-making may lead to dangerous situations. This thesis will also investigate how the falsification scenarios change depending on if one or several vehicles are controlled by a Deep RL policy. Lastly, this thesis aims to investigate how the ado agent controlling policy can be utilized to increase performance with regard to the safety of the host vehicle.

Formally, this project aims to answer the following questions:

- What are the strengths and limitations of using Deep RL to falsify an autonomous vehicle performing lane changes in a dense traffic situation?
- How does the falsifying scenario change depending on whether one vehicle or several vehicles are controlled by a Deep RL policy?
- How can the decision-making ability of the autonomous vehicle improve in regards to safety by utilizing a Deep RL-based adversarial agent?

1.2 Limitations

The research questions in this thesis were discussed and answered with several limitations present. First of all, the simulation environment used is an extension of an already existing environment. Modifying this environment to suit our needs would be time-consuming, and not reasonable in the short time frame of this project. In this environment, the agents are trained to be specialists in the specific situation presented in this thesis. Therefore, caution must be exercised when the agents are applied to other situations.

When working with RL, there exists a multitude of algorithms suitable for different tasks. In addition to this, the algorithms are generally extremely sensitive to hyper parameters. Investigating all possible combinations of RL algorithms and hyper parameters is not in the scope of this project. The provided framework is intended to work as a proof of concept, which invites the possibility of further work in the future.

1.3 Contribution

The contribution of this thesis is threefold. The first contribution is the proposal of a framework that uses Deep RL to conduct falsification on autonomous lane changing scenarios. The results show that the framework is successful in finding falsifying lane change scenarios by altering the behavior of one adversarial agent. This suggests that the safety of the autonomous lane change process can be improved. In addition, this work highlights how differences in vehicle positions change the outcome of the falsification process.

The second contribution is the introduction of using multiple agents to perform falsification, by an extension to the first contribution. This is shown to expand the diversity in the falsification scenarios, in comparison with a single agent system.

Finally, a method of improving the original autonomous lane changing policy is suggested. This method makes use of the policy achieved in the previously proposed framework, to improve the decision-making used to perform the autonomous lane change.

2

Theory

This chapter provides relevant background theory for the thesis. First, falsification is introduced, followed by key concepts in reinforcement learning and artificial intelligence. Thereafter, related work in falsification and reinforcement learning is presented. Finally, the paper that introduces the simulation environment is presented thoroughly as well as relevant evaluation methods.

2.1 Falsification

Relevant theory of falsification and related work, combining falsification and reinforcement learning, are presented in this section.

Autonomous driving systems consist of physical components, such as sensors, and software components. Such systems are referred to as Cyber-Physical Systems (CPS). Verification and evaluation methods used with entire software or physical systems are however not applicable to the majority of CPS due to complexity [10]. One verification method that has been applied to CPS is *falsification* [6]. Falsification of CPS is a method based on parameter optimization, used to search for the existence of *scenarios* that violate some specification. In autonomous driving settings, the specifications can for example be related to safety.

While the definition of the term *scenario* is somewhat difficult to generalize, Ulbrich et al. [11], provide a definition in the context of autonomous vehicles. However, to define a scenario properly, one must define the term *scene*. These two definitions are used frequently trough-out the thesis.

> **Definition 1 (Scene)** *A scene describes a snapshot of the environment including the scenery and dynamic elements, as well as all actors' and observers' self-representations, and the relationships among those entities. Only a scene representation in a simulated world can be all-encompassing (objective scene, ground truth). In the real world, it is incomplete, incorrect, uncertain, and from one or several observers' points of view (subjective scene)[11, p. 983].*

Using this definition, a scenario is defined as

Definition 2 (Scenario) *A scenario describes the temporal development between several scenes in a sequence of scenes. Every scenario starts with an initial scene. Actions & events as well as goals & values may be specified to characterize this temporal development in a scenario. Other than a scene, a scenario spans a certain amount of time [11, p. 986].*

2.2 Reinforcement Learning

This section includes basic reinforcement learning theory and key ideas, mainly MDP, reward, value-function and Q-learning.

Reinforcement Learning (RL) is a subfield of machine learning, where intelligent agents take actions within an environment, learning by their own decisions. Each time step, an agent observes the state of the environment and uses that information to decide on an action to execute and interact with the environment. This causes the environment to change, which creates a new state to observe. During each step of the process, the agent will be given a reward by the environment based on how good the state became as a consequence of the previously taken action. The reward is defined so that the agent is guided to reach a goal in the environment. The iterative process is continued until a terminal state is reached, which marks the end of an *episode*. The iterative process is illustrated in Figure 2.1.

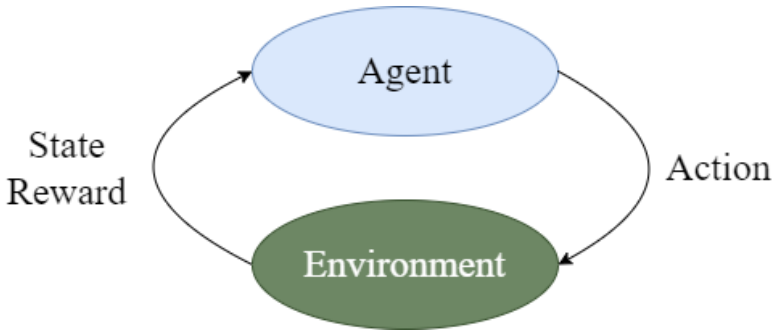


Figure 2.1: Reinforcement learning loop.

The goal in RL is to maximize the future cumulative reward over an episode, called the *return*. As the RL loop is repeated many times over, the agent learns a policy through trial and error. The agent can then follow the policy to take actions that will maximize the return.

RL algorithms can be categorized into model-based approaches, a famous example being the chess program AlphaZero [12], and model-free methods such as Q-learning and policy optimization.

2.2.1 Markov Decision Process

Formally, RL problems are formulated in the form of a Markov decision process (MDP) [13]. The MDP is a discrete-time stochastic control process often represented by a 5-tuple

$$\mathcal{M} = (S, A, P_a, R_a, \gamma), \tag{2.1}$$

where S is the set of states, A is the set of available actions. By selecting action $a \in A$ in a state $s \in S$, the system transitions from state s to $s' \in S$ with a probability determined by the state probability function $P(s, a, s')$. This transition is evaluated by the instantaneous reward function $R(s, a, s')$. Finally, γ is the discount factor.

The MDP advances from state to state in discrete steps $k = 0, 1, ...$, where each state has the Markov property, meaning that all information needed to proceed to the next state s' is provided by the current state s.

2.2.2 Reward and return

Given an MDP, the goal is to obtain a control policy π for the agent to follow which will guide the agent to maximize some cumulative function of rewards. As previously mentioned, the function is called the *return*, and is defined in different ways depending on its usage. In this project, the infinite-horizon discounted return [13] will be used, defined as

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t \tag{2.2}$$

where r_t is the reward at step t and $\gamma \in (0, 1]$ is the discount factor.

Regardless of which return function is selected, the objective is to select a policy π that maximizes the expected return

$$J(\pi) = \mathrm{E}_{\pi}[R(\tau)], \tag{2.3}$$

when the agent acts according to it.

In total, the optimization problem is to find the optimal policy

$$\pi^* = \arg\max_{\pi} J(\pi). \tag{2.4}$$

2.2.3 Value functions

There are two core expressions in RL for the expected return. The first expression is the On-Policy Value function

$$V^{\pi}(s) = \mathrm{E}_{\pi}\left[R(\tau) \mid s_0 = s\right], \tag{2.5}$$

which gives the expected return if the agent starts in the state s and always acts according to policy π. The second expression is the On-Policy Action-Value Function

$$Q^{\pi}(s,a) = \mathrm{E}_{\pi}\left[R(\tau) \mid s_0 = s, a_0 = a\right], \tag{2.6}$$

which gives the expected return if the agent starts in the state s, takes an action a arbitrarily, and then follows the policy π from the next state until the end of the episode.

An important property of the value functions is that they satisfy recursive relations similar to (2.2), that is for any policy π and state s the value function can be expressed recursively as

$$\begin{aligned} V^{\pi}(s) &= \mathrm{E}_{\pi}\left[R(\tau) \mid s_0 = s\right] \\ &= \mathrm{E}_{\pi}\left[r + \gamma R_{t+1} \mid s_0 = s\right] \\ &= \sum_{a} \pi(a|s) \sum_{s',r} p(s,a,s')[r + \gamma \mathrm{E}_{\pi}\left[R(\tau) \mid s_0 = s\right]] \\ &= \sum_{a} \pi(a|s) \sum_{s',r} p(s,a,s')[r + \gamma V^{\pi}(s')]. \end{aligned} \tag{2.7}$$

Expression (2.7) is the Bellman equation for V^{π}. It sums over three variables, a, s', and r. For each triple the probability, $\pi(a|s)p(s,a,s')$, is calculated which weights the quantity in the bracket to compute the expected value [14].

Solving an RL problem means finding a policy that yields a high return. The definition of what makes a policy π better than a policy π' is that for each state s the policy π yields higher (or equal) expected return than π'. Formally, π is better than π' if $V^{\pi}(s) \geq V^{\pi'}(s) \, \forall \, s \in \mathrm{S}$.

An optimal policy π^* that performs better than, or equal to every other policy always exists [14]. The optimal policy has a corresponding optimal value function V^* defined as $V^*(s) := \max_{\pi} V^{\pi}(s)$. Similarly, the optimal action value function Q^* is $Q^*(s,a) := \max_{\pi} Q^{\pi}(s,a)$.

2.2.4 Q-learning

One of the earliest breakthrough methods in RL was the Q-learning algorithm. Q-learning is an off-policy temporal difference control algorithm. An algorithm that learns off-policy can use previously collected data for optimization, regardless of what action the agent is taking each update. The Q-learning method is defined as

$$Q(s,a) \leftarrow Q(s,a) + \eta\left[R + \gamma \, \mathrm{argmax}_{a'} Q\left(s',a'\right) - Q(s,a)\right], \tag{2.8}$$

where η is the learning rate. The learning rate determines the magnitude of the update in (2.8).

The method learns an action-value function $Q(s,a)$ that approximates the optimal action-value function Q^*. The full Q-learning algorithm is shown in Algorithm 1.

Q-learning at its simplest saves the Q values for each state-action pair in a Q-table. This works well with a small, finite set of possible states and actions. As the number of states

Algorithm 1 Q-learning

Initialize $Q(s, a)$ for all $s \in$ S, $a \in$ A arbitrarily
for all episodes **do**
  Initialize s
  for all steps in episode **do**
    Choose a using Q
    Take action a, observe R, s'
    $Q(s, a) \leftarrow Q(s, a) + \eta \left[R + \gamma \text{ argmax}_{a'} Q\left(s', a'\right) - Q(s, a)\right]$
    s $\leftarrow$ s'
  end for when s is terminal
end for

and actions becomes larger, the Q-learning approach performs worse as the probability of the agent visiting a state and performing a single action becomes much lower. Moreover, as the state-action pairs grow the Q-table becomes larger and larger, which is inefficient for memory consumption.

2.3 Deep reinforcement learning

This section introduces the theory behind artificial neural networks and how it is applied in Double Deep Q-Networks.

Deep reinforcement learning (Deep RL) combines the semantics of reinforcement and deep learning, which is presented in this section. As presented in Section 2.2, Q-learning is one common RL-algorithm where the agent learns from trial and error by exploring the state space. However, in situations approaching real-world complexity, the feasibility of Q-learning is limited. This is mainly due to memory problems, induced by the inefficiency of Q-table at such high-dimensional data [14]. This problem is approached by substituting the Q-table with a function approximator - an artificial neural network.

2.3.1 Artificial neural network

The use of artificial intelligence and cognitive science started when McCulloch and Pitts [15] introduced a formal design of artificial neurons, mainly with inspiration from one of the most complex systems of all time - our brain and nervous system. The human perception of memory and learning was studied, then imitated by engineers in a computer setting - leading to artificial neural networks [16]. As in human intelligence, one major aim of artificial intelligence is the ability to process information. Given some input vector $\boldsymbol{x} = \{x_i\}_{i=1}^n$ that is to be evaluated, the purpose is to map the input to some output $\boldsymbol{y}$ by using a set of parameters $\boldsymbol{\theta}$, generally weights $\boldsymbol{w} = \{w_i\}_{i=1}^n$ and a bias b. The network learns the values of the parameters $\boldsymbol{\theta}$ that results in the most accurate mapping and function approximation.

The calculation process performed by a single neuron is shown in Figure 2.2, where the scalar product between the input vector x and the weights w are calculated. Then, the bias b is added. This expression is formally defined as $\boldsymbol{w}^T\boldsymbol{x} + b$. Lastly, the output is obtained by applying the non-linear activation function $g(\cdot)$ to the linear transformation. A multitude of different activation functions exists, of which one is ReLU [17]. It has

become common practice to use the ReLU activation function to solve gradient related problems, such as the infamous vanishing gradient problem.

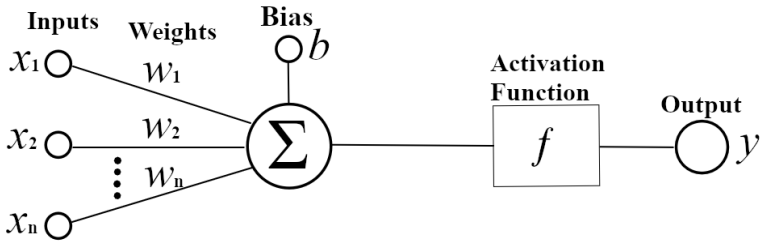


Figure 2.2: Schematic view of a single neuron. Inputs $\boldsymbol{x} = \{x_i\}_{i=1}^n$ are scalar multiplied with weights $\boldsymbol{w} = \{w_i\}_{i=1}^n$ and summed with bias b. Finally, the activation function is applied to the sum, which results in the output y.

Furthermore, a deep neural network is defined by multiple succeeding layers stacked, as shown in Figure 2.3. The layers consist of arbitrary many neurons. The layers are then connected either partially, or as in the case of fully connected layers, all the neurons in layer l are connected to all the neurons in the previous layer $l-1$ and the succeeding layer $l+1$. The information flows through the network, where the output from neurons in the previous layer acts as an input to neurons in the succeeding layer. This, as well as the non-linear transformation, enables the network to learn complex structures and behavior in the input data [18].

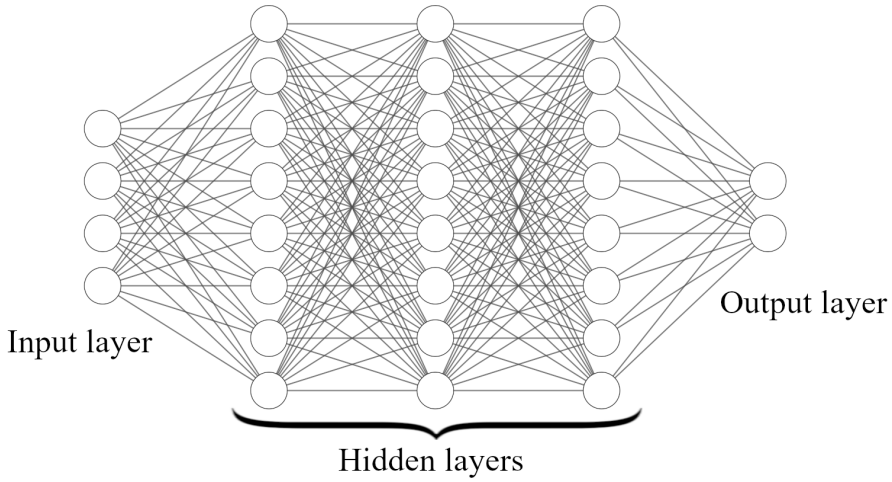


Figure 2.3: Deep neural network consisting of an input layer, hidden layers and an output layer. Every neuron follows the structure in 2.2.

The parameters $\boldsymbol{\theta}$ in the layers of the neural network are trained, i.e. optimized, to minimize an error, frequently called the *loss function*. The loss function evaluates the network and provides a metric of the performance of the model. There exist several

different loss functions, which are chosen on the task that is to be solved. Supervised learning tasks, such as classification, usually use some kind of classification error rate as a loss function, while loss functions such as mean squared error (MSE) or mean absolute error (MAE), are usually applied for regression problems [19]. Furthermore, Huber loss [20] is a widespread loss function that combines MSE and MAE. The Huber loss is given by

$$L_{Huber} = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{if } |y - f(x)| \leq \delta \\ \delta \cdot (|y - f(x)| - \frac{1}{2}\delta) & \text{otherwise,} \end{cases} \tag{2.9}$$

where y is the target and $f(x)$ is the predicted value of some input x. The hyper parameter δ defines the range for MSE and MAE. The combination of these two loss functions has been suggested to be less sensitive to outliers in data, compared to for instance MSE.

As mentioned, the trainable parameters $\boldsymbol{\theta}$ in the network are *optimized* to minimize the loss. One of the most simple and well-known optimization algorithms derives the gradients of the loss L for each trainable parameter, called gradient descent. The parameters at time step $t + 1$ are calculated by subtracting the gradients, calculated using backpropagation [18], from the parameters at time step t, mathematically defined as

$$w_{t+1} = w_t - \eta \frac{\partial L}{\partial w_t}. \tag{2.10}$$

The magnitude of the adjustment is defined by the hyper parameter η, which is the *learning rate*, by multiplying η with the gradients. However, there exist drawbacks with this simple optimization method, especially for highly non-convex functions, such as getting trapped in sub-optimal local minima [21]. To deal with the drawbacks, several modifications of the gradient descent algorithm have been invented - such as stochastic gradient descent [22] and Adaptive Moment Estimation (Adam) [23]. Adam combines the advantages of two optimization algorithms: RMSprop and AdaGrad [21]. Consequently, the Adam algorithm has shown efficiency and success when optimizing problems including a lot of data and parameters.

2.3.2 Double Deep Q-Network

One of the first implementations using deep neural networks in combination with RL, with satisfactory results was presented by Mnih et al. [24], termed Double Deep Q-Network (DDQN). The authors demonstrate that their DDQN-algorithm successfully obtains performance higher than all previous algorithms on the Atari 2600 games. In addition, the agent performs equivalent to a professional human player in the majority of games. Previous research adopting neural networks as a Q-value approximators has shown several instabilities, such as correlation between observations and that small updates to the neural network can affect the policy drastically [25]. These instabilities are in [24] approached by two key ideas: *experience replay* and *target network*.

To increase the stability of the neural network, separate networks for generating the actions and for generating the targets, respectively, is implemented. The target network outputs the targets, used to calculate the loss. The target network $\hat{Q}$ is held fixed, except for every C episode, where the parameters $\hat{\boldsymbol{\theta}}$ for the target network are updated from the action

network parameters $\boldsymbol{\theta}$. This modification reduces the correlation between the target and the action values, which otherwise is strong since the states s and s' are often similar [24].

Experience replay is implemented to reduce the correlation between observations. The agent's experiences e_t (see Section 2.2.1) are at each time-step t stored in a buffer. The buffer is typically implemented such as when the buffer is full, the oldest sample is removed and the newest experience is added, resulting in a continuous update of samples in the buffer. Typically, uniformly distributed observations (mini-batches) from the buffer are used to update the action network with parameters $\boldsymbol{\theta}$. Thus, one sample can be used several times to update the network, in different batches.

The DDQN-algorithm, as presented in [24] with experience replay, is presented in Algorithm 2.

Algorithm 2 DDQN with experience replay.

```
Initialize experience replay buffer D
Initialize Q-network Q, target network Q̂ with random weights θ = θ̂
Define probability ϵ used to select action
for all episodes do
  Initialize state s
  for all time steps do
    p ← uniform random number between 0 and 1
    if p ≥ ϵ then
      a = argmax_a' Q(s, a'; θ)
    else
      Choose random action a
    end if
    Execute action a, observe reward r and new state s'
    Save experience e = (s, a, r, s') in D
    if D is full then
      Sample mini-batch of experiences from D
      Calculate target y = r + γ argmax_a' Q̂(s, a'; θ̂)
      Perform gradient descent step using temporal difference error l_td = (y −
      Q(s, a; θ))^2 on action network Q(s, a; θ)
      Set Q = Q̂ every C steps
      Remove oldest sample in D
    end if
  end for
end for
```

In addition to the crude uniformly distributed sampling technique adopted in [24], the *prioritized replay* scheme [26] has shown to result in faster learning, using the same environment as in [24]. The main idea is that some experiences are more important than others, and sampling these experiences more frequently could lead to more efficient learning. Adopting prioritized replay, the probability of selecting each sample is calculated for all the samples in the experience replay buffer. When sampling from the buffer, samples with a large probability have a larger possibility of being replayed. The probability of

sampling experience i is calculated as

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \tag{2.11}$$

where $p_i > 0$ is the priority of transition i. The constant α denotes how much prioritization that is used. Note that $\alpha = 0$ represent the uniformly distributed case. The priority of a transition is updated by the absolute value of the temporal difference error, $p_i \leftarrow |l_{td,i}| + \varepsilon$, where ε is a small positive constant.

However, bias is introduced when implementing prioritized replay, since the sample distribution changes using this sampling scheme. The bias can be corrected by calculating the importance of sampling weight, defined as

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta \cdot \frac{1}{\max_j w_j}, \tag{2.12}$$

where N is the number of samples in the buffer and β defines how much prioritization to use. The value of β increases towards 1 during training, with the motivation that Deep RL is unstable at the beginning and prioritization matters more near the end of training. During the optimization step, the importance weights are multiplied with the calculated loss, and then used by the optimization algorithm.

2.4 Related work

This section brings up relevant works on regular and reinforcement learning augmented falsification.

To find faults in a CPS, [7] uses robustness-guided falsification. There exist several approaches on how to apply robustness-guided falsification in autonomous driving systems. One way is to define the specification ψ as a signal temporal logic (STL) function, which is a formalism to describe characteristics of a real-valued, dense trajectory dependent on time. In this context, an example of a specification could be the safety distance between two agents in the environment within a specific time frame. This approach enables the possibility to obtain a value of how far a signal is from violating the specifications, which is determined by the robustness function ρ. The function is real-valued and dependent on the specifications ψ and the current signal from the CPS. A negative value represents that the specification is violated, while a positive value represents non-violation of the specification. As previously mentioned, the robustness function value also provides a measure for how far a signal is from violating or satisfying the specifications. A signal that leads to a small positive robustness value is closer to falsifying the system than a signal that results in a large positive value [7]. Consequently, the robustness function is to be minimized to go below zero, at which point the specification is successfully falsified. This is usually guided by an optimization algorithm, such as simulated annealing [6].

Furthermore, recent research has shown the feasibility of formulating falsification as an RL problem. Yamagata et al. [8] adopt two state-of-the-art Deep RL methods, in particular Asynchronous Advantage Actor-Critic (A3C) and Double Deep Q Network (DDQN),

and theoretically present how robustness guided falsification can be formulated as an RL-problem. This was experimented on with three different kinds of CPS. First, an array of five cars where one car was controlled by the Deep RL system whereas the other cars were driven autonomously following a set of instructions. Second, an automatic transmission controller for a train that controls RPM, gear, and vehicle speed. The third experiment was on a power train controller, which controls the air/fuel ratio in a combustion engine. The Deep RL algorithms are evaluated and compared to baseline methods traditionally employed in falsification experiments. The authors conclude that when certain conditions are met, the RL methods outperform the three baseline methods: uniform random sampling, cross-entropy, and simulated annealing. These conditions are the availability of the states for the Deep RL algorithm, as well as there being something for the deep RL algorithm to exploit in the reward function to speed up the falsifying process.

In the work of Corso et al. [27], Adaptive Stress Testing (AST) was used to find the failure scenarios with the highest possibility of occurring, which is solved using Trust Region Policy Optimization (TRPO) and Monte Carlo Tree Search (MTCS). The authors however state that ordinary AST tends to find scenarios where failure is avoidable as the vehicle specification are often falsified during the final time step, and the same scenarios are often discovered. Hence, two modifications of the reward function are presented. The first modification is based on Responsible-Sensitive Safety (RSS). The RSS reward aims to find failure scenarios where the vehicle to be falsified behaves improperly multiple time steps. The other modification encourages dissimilarity between the failure scenarios by adding a dissimilarity reward to the reward function. The dissimilarity reward compares the trajectory of a collision to all previously found collisions and rewards the agent depending on how it differs from the other scenarios. The results suggest that the augmented reward functions can address the problems with ordinary AST.

In the two papers [8, 27] briefly reviewed above, as well as in [28] and [29], the authors suggest that reinforcement-based falsification can be advantageous in comparison with traditional methods, such as Breach [7, 30] and S-TaLiRo [31]. These methods perform falsification as a parameter search over predefined parameter spaces. During initialization of the methods, a parameter range is provided as well as some specification ψ that are to be violated. This parameter range is used to generate simulation trajectories, which are evaluated against ψ - thus determined to either violate ψ or not. The problem is solved when the parameter sequence yields a trajectory that violates ψ. With this approach, given an input sequence, the found falsifying trajectory is deterministic, that is, acting according to a fixed instruction and not dependent on the current state. Consequently, a parameter sequence found by these tools is not generalizable. If one were to change the model frequently, for example by changing initial conditions, the falsification process must be performed again to ensure successful falsification. This is in contrast to RL, which uses temporal structures to evaluate how to succeed with the falsification. In addition to this, Deep RL has proven to be successful in learning structures in data from complex environments such as video games [24, 32] and CPS's [8].

In the previously mentioned works, the RL algorithms chosen to produce the results were not the same. Corso et al. [27] used TRPO, while Yamagata et al. [8] used DDQN and A3C. Generally, there are two main approaches for RL algorithms: value function-based methods, such as DDQN, and methods based on policy search, such as A3C [33] and TRPO [34]. There exist hybrid actor-critic approaches as well (SAC [35], DDPG [36]), which include both value functions and policy search. There exist several disadvantages

and advantages to each algorithm. TRPO applies to high dimensional problems, and is at the same time shown to be robust. However, the algorithm is computationally expensive. The algorithm A3C has been shown to result in more efficient training [37]. In the literature, there exists small to no consensus on whether one algorithm is more suitable than another in all cases. It heavily depends on the simulation setup and the task that is to be solved [8, 27].

2.5 Lane changes with shielded semantic actions

This section provides information on how the simulation environment was created. The behavior of the host policy, and the key concepts of the safety shielding are described.

The simulation environment that is being experimented on in this project is an extension of [9]. The authors introduce a new approach to tactical decision-making for a lane change in a densely packed traffic situation. Using low-level controllers, such as acceleration and speed, semantic actions are formulated and defined as the action space for a Deep RL algorithm, in particular DDQN. The DDQN-algorithm was used to train a host vehicle V_{host} to succeed with a lane change, from start lane to target lane. The surrounding adversarial cars operate using PID-controllers, designed to mimic human driving. The described situation is presented in Figure 2.4.

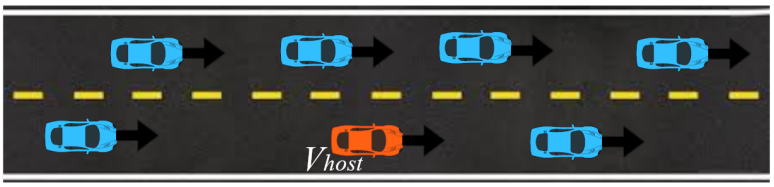


Figure 2.4: Simulation scenario. The orange car is V_{host}, the surrounding cars are adversarial PID-controlled agents.

V_{host} was shielded from performing unsafe actions by judging if a certain action was deemed too dangerous to perform, and in that case, it was removed from the list of allowed actions. This leads to situations where only safe lane changes can be executed by V_{host}. The additional steps to the decision-making process lead to decreased learning time for the Deep RL training loop. This is due to V_{host} being able to focus entirely on efficient interaction with the environment, while not having to worry about taking actions that terminate the episode prematurely. The way V_{host} was rewarded was engineered such that the lane change is performed efficiently and comfortably. With the obtained policy, V_{host} was able to autonomously make lane changes with a success rate of 95% [9].

As previously mentioned, the authors in [9] make use of shielding of V_{host} to prevent the host from taking dangerous actions, ensuring a safe lane change. This shielding excludes the dangerous actions from the experience tuple by extending the MDP in Section 2.2.1 with the list A^{safe}. In addition, the action obtained by the DDQN must be in the set A^{safe}. This addition leads to the agent never encountering an experience tuple containing

dangerous actions. Lane change is determined to be unsafe if it will lead to time gaps $\tau_{margin} \leq 0.5s$ with the vehicles in the target lane, and if the new following vehicle in the target lane would need to brake more than 2 m/s^2 to avoid a collision.

2.6 Evaluation metrics

This section introduces the Brake Threat Number (BTN) and minimal safe longitudinal distance as evaluation metrics for the falsification experiments.

When evaluating falsification scenarios, multiple metrics can be used to judge the performance of the model. In [8] and [28] the success rate of falsification was used, while [27] relied on experiment-specific metrics that tested the hypothesized improvements of their experiments. Every metric has advantages and disadvantages, which is why the choice of evaluation metric depends on the purpose of the work.

2.6.1 Brake Threat Number

One approach to evaluate scenarios is through an algorithm for calculating the Brake Threat Number (BTN) [38]. The BTN represents a value of how much a trailing vehicle V_{trail} must brake a_x^{req}, relative to the potential maximum brake $a_x^{min^*}$, to avoid collision with a leading vehicle V_{lead}. The potential maximum brake is defined as

$$a_x^{min^*} = \max\left(a_x^{min}, a_{x,trail} + j^{min} t_s\right). \tag{2.13}$$

where a_x^{min} is the maximum possible brake for the vehicle, $a_{x,trail}$ is the initial acceleration for V_{trail} and t_s is the time it takes for V_{trail} to become stationary.

The BTN is defined such that when BTN < 1, the vehicle can avoid a collision, but for BTN $= 1$ crashing is unavoidable. The BTN assumes that the lead vehicle is the only obstacle present. The acceleration for the lead vehicle is assumed to be constant $a_{x,lead}(t) = a_{x,lead}(0)$ until it is stationary if $a_{x,lead}(0) < 0$.

The BTN is defined as

$$BTN = \frac{a_x^{req}}{a_x^{min^*}}, \tag{2.14}$$

where a_x^{req} is the required acceleration to successfully avoid a collision and $a_x^{min^*}$ is the maximum possible deceleration the vehicle can perform before the vehicle comes to a rest. The braking system can be described in three parts: first, a period t_d where the vehicle drives with the constant initial acceleration, and second, the vehicle changes its acceleration with jerk j_{min} until it reaches a_x^{min} or becomes stationary. Then, the vehicle brakes with constant deceleration a_x^{min}.

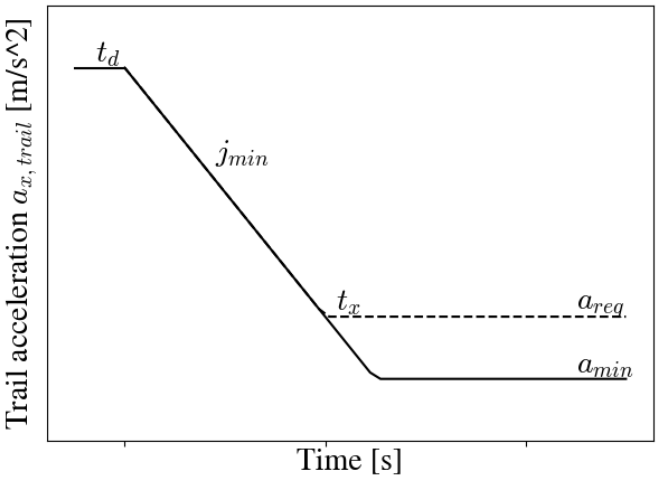


Figure 2.5: Brake system for determining BTN for V_{trail}.

To derive the BTN there must first be derivations of t_s and a_x^{req}.

Define initial trailing vehicle velocity as $v_{x,trail}$. The the vehicle velocity at time $t_s + t_d$ is

$$v_{x,trail}(t_s + t_d) = v_{x,trail} + a_{x,trail}(t_s + t_d) + \frac{j^{min} t_s^2}{2}. \tag{2.15}$$

Solving $v_{x,trail}(t_s + t_d) = 0$ gives

$$t_s = -\frac{a_{x,trail}}{j^{min}} + \sqrt{\frac{a_{x,trail}^2}{j^{min^2}} - 2\frac{v_{x,trail} + a_{x,trail} t_d}{j^{min}}} \tag{2.16}$$

To derive a_x^{req} there are two cases to consider: lead vehicle in motion, and stationary lead vehicle/lead vehicle braking to stationary. First, consider the case where both vehicles are in motion.

The range between the trailing and the lead vehicle is predicted by

$$r(t) = r_0 + \left(v_{x,lead} - v_{x,trail}\right) t + \left(a_{x,lead} - a_{x,trail}\right) \frac{t^2}{2} - j^{min} \frac{t^3}{6} \tag{2.17}$$

where $v_{x,lead}$ and $a_{x,lead}$ are the lead vehicles velocity and acceleration, respectively, $t > 0$ is the time and r_0 is the range at $t = 0$. In the same way, the range rate is predicted by

$$\dot{r}(t) = v_{x,lead} - v_{x,trail} + \left(a_{x,lead} - a_{x,trail}\right) t - \dot{j}_{\max} \frac{t^2}{2}. \tag{2.18}$$

The trailing vehicles acceleration for $t > 0$ is given by

$$a_{x,j}(t) = a_{x,trail} + j^{\max} t \tag{2.19}$$

and when the vehicles are in motion, the needed acceleration to avoid collision is given by

$$a_{x,m}(t) = a_{x,lead} - \frac{\dot{r}(t)^2}{2r(t)}. \tag{2.20}$$

Solving $a_{x,j}(t) = a_{x,m}(t)$ gives $t_x > 0$ which is used to calculate

$$a_x^{\text{req}} = a_{x,trail} + j^{min} t_x. \tag{2.21}$$

This solution is valid if the lead vehicle is still in motion when the vehicles make contact,

$$v_{x,lead} + a_{x,lead} t_c > 0 \tag{2.22}$$

where t_c is the time of contact and is calculated by

$$t_c = t_x - \frac{2r(t_x)}{\dot{r}(t_x)}. \tag{2.23}$$

If (2.22) is not true, then the lead vehicle is stationary by the predicted time of the collision.

If the lead vehicle is stationary or braking until it becomes stationary, then the needed constant acceleration for the trailing vehicle to stop in a range $r_s(t)$ is

$$a_{x,s}(t) = -\frac{\dot{r_s}(t)^2}{2r_s(t)}. \tag{2.24}$$

In this expression the predicted range is

$$r_s(t) = r_0 + r_l - v_{x,trail} t - a_{x,trail} \frac{t^2}{2} - j^{min} \frac{t^3}{6} \tag{2.25}$$

where $r_l = -v_{x,lead}^2/(2a_{x,lead})$ is the range for the lead vehicle stop. The predicted range rate is given by

$$\dot{r_s}(t) = -v_{x,trail} - a_{x,trail} t - j^{min} \frac{t^2}{2}. \tag{2.26}$$

The a_x^{req} can be calculated by solving $a_{x,j}(t) = a_{x,s}(t)$ and inserting t_x into (2.21).

2.6.2 Minimal Safe Longitudinal Distance

Another approach that has been previously used in falsification projects such as [27, 39] is the minimal safe longitudinal distance d_{min}. Let V_{trail} and V_{lead} be two vehicles that are driving in the same lane and direction. Then, d_{min} is derived by calculating the minimal distance needed for V_{trail} to avoid collision in the case that V_{lead} needs to brake with maximal deceleration a_x^{min} [4]. If all of the criteria mentioned above are satisfied, then d_{min} is given as

$$d_{min} = \left[v_{x,trail} t_d + \frac{1}{2} a_x^{max} t_d^2 + \frac{\left(v_{x,trail} + t_d a_x^{max} \right)^2}{2 a_x^{min}} - \frac{v_{x,lead}^2}{2 a_x^{min}} \right]_+ \tag{2.27}$$

where $[x]_+ = \max(x, 0)$, t_d is the reaction time of V_{trail}, $v_{x,trail}$ and $v_{x,lead}$ are the velocities of V_{trail} and V_{lead}, respectively, and the accelerations a_x^{min} and a_x^{max} are the minimum and maximum possible accelerations. Through this metric, a situation is considered longitudinally dangerous if the range r_{trail_lead} between V_{trail} and V_{lead} is smaller than the safe distance d_{min}.

3

Method

This chapter presents information about the experimental setups used to answer the research questions presented in Section 1.1, divided into three main parts. First of all, the simulation environment, including assumptions on the operating vehicles, is presented. The next part contains information regarding the DDQN and the training loop. The last part describes three different experimental setups implemented to answer the research questions, including evaluation methods.

3.1 Problem definition

This section defines the problem to be investigated, and formalises the property to be falsified.

The experiment setups used in this project were based on the setups presented in Section 2.5, where the host vehicle V_{host} is attempting to change lane in a densely packed traffic situation. In a lane-changing scenario, V_{host} is considered successful if it manages to complete a lane change within the limited time frame T. Furthermore, V_{host} needs to not crash in to any other vehicle during the scenario. To accomplish this, in [9] a policy π_{host} which succeeded in 96 out of 100 cases was produced.

However, the performance of π_{host} and the safety shield were measured using simulations where the behaviors of the adversarial vehicles were passive. The performance of π_{host} in experiments with altered behavior of the adversarial vehicles was not concluded in [9]. Therefore, this project aims to perform more robust testing of the safety shield by trying to find edge case scenarios where the policy π_{host} contributes to creating dangerous situations. This will be done through falsification.

With the falsification method, the aim is to find counter-examples that violate a safety requirement. The safety requirement that will be used in this work is specified in Definition 3.

Definition 3 (Property to falsify) *Let x_{host} and x_{ado} be the longitudinal position of V_{host} and V_{ado} respectively, measured at the rear of the vehicles. Let $l_{vehicle}$ be the vehicle length and d_c denote the critical distance separation between two vehicles. Then, the safety requirement for a successful lane change is fulfilled if $\psi_{host} \equiv |x_{host} - x_{ado}| - l_{vehicle} > d_c$ holds true for all $t \in [0, T]$.*

In this thesis, $l_{vehicle} = 5$ and $d_c = 0$. With this setting, Definition 3 can be interpreted

as: *The host vehicle's front or rear does not collide with an adversarial vehicle within the time T.*

The problem was investigated by implementing three different experiments, further explained in Section 3.4. The first two experiments include modifications to the behavior of one or several adversarial vehicles, with the goal to falsify the property, presented in Definition 3. In the last experiment, the host's policy π_{host} was improved by making use of the adversarial behavior obtained in the first experiment.

3.1.1 Vehicle constraints

To obtain the most informative and realistic counter-examples to the falsification property defined in Definition 3, constraints are introduced to the simulation environment. Firstly, there are hard constraints that represent the limit on what are feasible parameters in the situation. These constraints are applied on every vehicle $V_{host}, V_{ado}, V_{trail}, V_{lead}, V_1, ...V_n$ in the simulation. Then, to encourage normal human driving for V_{ado}, soft constraints were introduced. These constraints can be exceeded, but doing so is penalized as described in Section 3.2.2.

1. Velocity
   (a) There exists max and min bounds on the velocity. The max and min bounds are defined as $v_x^{min} \leq v_x \leq v_x^{max}$. In the simulation the absolute velocity bounds, bounds that cannot be violated, are denoted by v_x^{min} and v_x^{max}.
   (b) The soft velocity bounds, that can be violated but with a penalty, are $\gamma_1 v_x^{lim} \leq v_x \leq \gamma_2 v_x^{lim}$, where v_x^{lim} is the road speed limit, γ_1 and γ_2 are the tolerance limits for diverging from the speed limit.
   (c) It is assumed that the speed limit v_x^{lim} is the speed limit for both the start lane for V_{host}, and its target lane.
   (d) The vehicles are only moving in the longitudinal direction, except for when V_{host} performs a lane change.
2. Position
   (a) At the initial state, the lateral position of the vehicles are determined as $y_{ado}(0) > y_{host}(0)$, since the host is initialized in the original lane and the ado in the target lane. Furthermore, V_{ado} has a constant lateral position $y_{ado}(t) = y_{ado}(0)$.
3. Acceleration
   (a) The longitudinal acceleration has absolute constraints $a_x^{min} \leq a_x \leq a_x^{max}$ where a_x^{min} is the minimum bound and a_x^{max} is the maximum.
4. Jerk
   (a) The change in acceleration, jerk, is limited as drivers do not normally drive with high amounts of jerk due to it being uncomfortable. The jerk is soft constrained by $|j| < j^{max}$.
5. Distance

(a) Assumption on the desired distance between a vehicle and its lead vehicle, such that $|x_i - x_j| \geq d_s$ where $i, j \in \{ado, trail, lead\} \wedge i \neq j$ and $d_s = \tau^{safe} v_{x,i}$ is the safe distance separation. This assumption is based on rational thinking of keeping safe distance to vehicle ahead, when driving on a highway.

Table 3.1: Table of parameter values used in the experiments.

Parameter	Value	Description
v_x^{min}	0 m/s	Minimal longitudinal velocity
v_x^{max}	50 m/s	Maximal longitudinal velocity
v_x^{lim}	20 m/s	Road speed limit
γ_1	0.8	Lower tolerance limit when diverging from v_x^{lim}
γ_2	1.2	Upper tolerance limit when diverging from v_x^{lim}
a_x^{min}	$-5\,\mathrm{m/s^2}$ [40]	Minimal longitudinal acceleration
a_x^{max}	$3\,\mathrm{m/s^2}$ [40]	Maximal longitudinal acceleration
j^{max}	$2\,\mathrm{m/s^3}$ [40]	Maximal longitudinal jerk
τ^{safe}	0.5 s	Comfortable time gap between vehicles

3.2 Formulating the problem as MDP

This section provides information on how the ado MDPs are set up and why it is implemented in this way. This includes the action space and the reward function.

A policy π_{ado} for V_{ado} was obtained using DDQN, to systematically falsify the property defined in Definition 3. At each time step t, a set of observations O taken from multiple agents in the environment as shown in Table 3.2 are stored. The observation includes speed and acceleration of V_{ado}, in combination with speed, distance and acceleration of V_{host}, the nearest vehicle ahead of ado V_{lead} and the nearest vehicle behind V_{trail} ado, indexed with i.

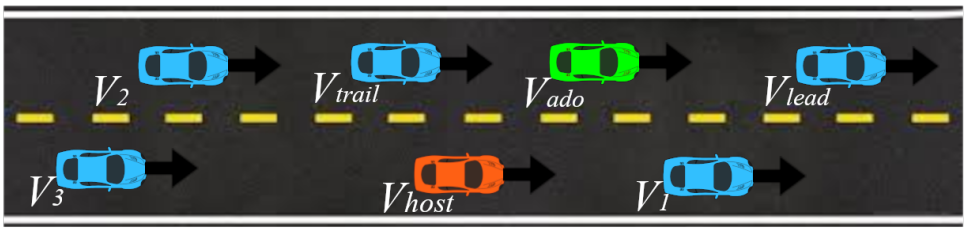


Figure 3.1: Visualization of the environment. V_{host} in orange, ado vehicle V_{ado} in green and other vehicles in blue. The observations are taken from the labeled vehicles.

Table 3.2: Observations taken each time step. i in this table is index for $trail, lead$ and $host$. $v_{x,i}$ is measured by the vehicles actual velocity in comparison to the road limit of $20\,\mathrm{m/s^2}$.

Observation	Description
$v_{x,ado}$	ado velocity
$a_{x,ado}$	ado longitudinal acceleration
$v_{x,i}$	longitudinal velocity
$a_{x,i}$	longitudinal acceleration
$d_{x,ado,i}$	longitudinal distance to ado
$d_{y,ado,i}$	lateral distance to ado

To grasp the intention of the surrounding vehicles, the state space $\mathcal{S}$ contains states s_t which includes observations of the last three consecutive time steps, given by

$$s_t = \{O_t, O_{t-1}, O_{t-2}\}, \tag{3.1}$$

where each observation is separated by 0.5 seconds [9].

3.2.1 Action space

The action space used in the experiments was created with a jerk-based approach using three actions in Table 3.3. This action space aims to put a hard constraint on the jerk, while still being able to reach the whole spectrum of allowed accelerations as presented in Section 3.1.1. The first action A_1 is to hold consistent speed using the same speed control as the PID-controlled vehicles are using in [9]. The second action A_2 decreases the acceleration value by 1. If the acceleration would fall below a_x^{min}, it instead takes the value a_x^{min}. Finally, action A_3 increases the acceleration value by 1, but never exceeds a_x^{max}.

Table 3.3: Action space used by V_{ado}.

Action	Longitudinal Acceleration
A_1	$PID(s, a)$
A_2	$\max_a (a_x^{min}, a_x(t-1) - 1)$
A_3	$\min_a (a_x^{max}, a_x(t-1) + 1)$

This action space was decided on after comparison with another action space, defined as

$$A = \{-5, -4, -3, -2, -1, 0, 1, 2, 3\}, \tag{3.2}$$

where the agent was able to take an action each time step, accessing the whole allowed acceleration space. Consequently, enabling a higher jerk than the action space presented in Table 3.3. The comparison showed that due to the larger complexity of the discrete action space, it performed worse in evaluation than the final action space used in this project.

As the jerk constraint is based on human comfort and not vehicle physics, a variation of the discrete action space might be preferable to find a wider variety of scenarios.

This set of actions is in stark contrast to the semantic actions that V_{host} uses [9]. V_{host} chooses a target vehicle to follow, which lets low-level controllers adjust acceleration to smoothly keep a set time gap τ to the vehicle to follow. The purpose of instead using the action space presented in Table 3.3 is to enable V_{ado} to change accelerations instead of changing which vehicle it should keep a set distance to. This opens up a lot more possibilities for different scenarios, which in turn also can make the policy less consistent. This is supported by the purpose of this project, as the aim of this project prioritizes finding unique edge cases over consistent results.

3.2.2 Reward function

The reward function presented in this section was derived with an iterative process. The process consisted of adding, removing and evaluating different components in the reward function. The final optimized reward function consists of three parts: velocity rewards R_v, distance reward R_d and the falsification reward R_f. The first two parts act as support to the agent while learning, while the last part is the final goal. The total reward $R(s, a, s')$ the agent obtained in state s taking action a in the environment each time step t was defined by

$$R(s, a, s') = R_v + R_d + R_f. \tag{3.3}$$

The implementation and motivation for each reward part of the total reward function are described in this Section.

3.2.2.1 Velocity reward

The first reward was defined to support a realistic driving pattern with respect to velocity. In a real-world setting, the vehicles acting on a road seldom drive with a constant velocity v_x identical to the speed limit on the road, v_x^{lim}, but rather within a range in the vicinity of the speed limit, further described in Section 3.1.1 item 1 (b). Thus, the agent receives a penalizing reward if its velocity is not within the desired velocity range, further presented in (3.4). The penalty is designed to linearly increase as the difference between the velocity and desired velocity increases.

$$R(s, a, s')_v = \begin{cases} -\min(\alpha_1 \cdot (\gamma_1 v_x^{lim} - v_{x,ado}), \alpha_2) & \text{if } v_{x,ado} < \gamma_1 v_x^{lim} \\ -\min(\alpha_1 \cdot (v_{x,ado} - \gamma_2 v_x^{lim}), \alpha_2) & \text{if } v_{x,ado} > \gamma_2 v_x^{lim} \\ 0 & \text{otherwise,} \end{cases} \tag{3.4}$$

where α_1, α_2, γ_1 and γ_2 are constants. The value of α_1 was chosen to scale the difference between the velocities, in relation to α_2. The maximum reward the agent could obtain each time step was restricted by α_2, implemented to ensure that each reward is weighted appropriately in the total reward function, see (3.3). Furthermore, the value of γ_1 and γ_2 was chosen by extensive parameter search. The final values resulted in a velocity range as close to the velocity limit as possible, and at the same time yielded informative falsification

scenarios. This addition to the reward function was of great importance to obtain an agent that behaved realistically. Without the implementation, uninformative falsifying scenarios were found. This could for example be scenarios where the agent behaved with a killer instinct - colliding into the host with an unrealistic high velocity.

3.2.2.2 Distance reward

The next reward is engineered to reinforce the agent to maintain a safe time gap τ^{safe} to the trailing and leading vehicle in the same lane, based on the vehicle assumption presented in Section 3.1.1 item 5a. The parameter τ^{safe} was determined by parameter search. The time gap between the leading and trailing vehicles was defined as

$$\tau_{ado,lead} = d_{ado,lead}/v_{x,ado} \tag{3.5}$$

$$\tau_{ado,trail} = d_{ado,trail}/v_{x,trail}, \tag{3.6}$$

where $d_{ado,i}$ is the longitudinal distance from the ado agent to vehicle i defined as $d_{ado,i} = |x_{ado} - x_i|$, where $i = \{lead, trail\}$. The reward was defined to yield a penalty linear to the magnitude of the time gap, and defined as

$$R(s,a,s')_d = \begin{cases} -(1-\tau_{ado,lead}) & \text{if } \tau_{ado,lead} \leq \tau^{safe} \\ -(1-\tau_{ado,trail}) & \text{if } \tau_{ado,trail} \leq \tau^{safe} \\ 0 & \text{otherwise.} \end{cases} \tag{3.7}$$

The constant τ^{safe} was chosen by what we perceive to be a reasonable value, as well as an analysis of the environment. The selected value results in relatively short distances.

3.2.2.3 Falsification reward

Furthermore, the agent is guided into falsifying the property 3 by obtaining a large positive reward when succeeding with the falsification. In addition, the agent receives a negative reward if the agent acted in a way that lead to a crash not including V_{host}. The mathematical formulation is given by

$$R(s,a,s')_f = \begin{cases} -5 & \text{if crash not including host} \\ \max(100 - 4 \cdot t_{alc}, 0) & \text{if crash including host} \\ 0 & \text{otherwise,} \end{cases} \tag{3.8}$$

where t_{alc} is the time step difference between the collision and when V_{host} committed to lane change. Each value in the reward function is derived to obtain desired ado behavior. If the agent succeeds with falsifying Definition 3, that is colliding with the host, it receives a large reward in comparison with other parts of the reward. The reward is engineered in such a way that a collision close to when the host committed to lane change results in a higher reward. The motivation behind this is that the faster the collision occurs after the lane change, the more likely it is that the lane change was a main contributing factor to why the collision occurred. On the contrary, if the lane change happened long before the collision, it is more likely that the lane change only was a minor contributor to the collision.

3.3 Safety Falsification using DDQN

This section describes the training setup, using DDQN, to guide the agent into finding counter-examples to the falsification property.

In order to guide the agent into falsifying Definition 3, the RL-algorithm DDQN, presented in Algorithm 2, was implemented. The state variables of the MDP presented in Section 2.2.1 were used by the agent to interact with the environment.

3.3.1 Training details

The agent was trained for n time steps, during which the agent makes use of the ϵ-greedy policy to decide on which action a_t to execute. The agent selects a random action with probability ϵ, and otherwise the action with the highest Q-value. The selected action is executed during k time steps, thus a new action is chosen every k^{th} time step. The value of ϵ decreases linearly over the first $\epsilon_{fraction} \cdot n$ time steps, decaying from ϵ_{start} to ϵ_{final}. The internal state of the simulator changes due to the taken action, yielding an experience tuple e_t, given by

$$e_t = (s_t, a_t, r_t, s_{t+1}, t_{terminal}), \tag{3.9}$$

where the selected action a_t executed in state s_t yields the reward r_t, resulting in the agent transitioning to state s_{t+1}. The value $t_{terminal} \in \{0, 1\}$ denotes whether or not the action taken results in termination of the episode. The episode is terminated if the maximum number of time steps T is reached, or if the falsification is succeeded. The experience tuple e_t is stored in the experience replay buffer D, which acts as a buffer for the training data. The samples used for training were drawn using the prioritized replay scheme, with a batch size of N.

The end goal during training is to minimize the Huber loss, defined as

$$l = \begin{cases} \frac{1}{2}(y - Q(s, a; \boldsymbol{\theta}))^2 & \text{if } |y - Q(s, a; \boldsymbol{\theta})| \leq \delta \\ \delta \cdot (|y - Q(s, a; \boldsymbol{\theta})| - \frac{1}{2}\delta) & \text{otherwise,} \end{cases} \tag{3.10}$$

where Q is the action network, $\hat{Q}$ is the target network, δ is some parameter and y is the target, defined by $y = r + \gamma \max_{a'} \hat{Q}(s, a'; \hat{\boldsymbol{\theta}})$.

The parameters of the action network were adjusted to minimize the error using the optimization algorithm Adam, with the learning rate η. The updated weights were obtained by back propagation, see Equation (2.10). The gradients calculated by the optimizer were normalized so the norm of the gradients was less or equal to the clipping value Γ. The gradients remain unchanged if the norm is less or equal to Γ. This is implemented to prevent the infamous problem of exploding gradients.

By definition of DDQN-algorithm, two networks are used during training. The target network $\hat{Q}$ was updated every C time step by $\hat{Q} \to Q$, where Q is the action network.

Table 3.4: Parameters used during training with DDQN.

Parameter	Value	Description
N	32	Batch size
D	50 000	Replay buffer size
C	500	Target network update frequency
γ	1	Discount factor
η	0.0005	Learning rate
ϵ_{start}	1	Start exploration factor
ϵ_{final}	0.02	Final exploration factor
$\epsilon_{fraction}$	0.6	Determines during how many training steps until ϵ_{start} reach ϵ_{final}
Γ	10	Gradient clipping value
n	500 000	Number of training steps
δ	1	Defines range for MSE and MAE in Huber loss
k	10	New action every $k = 10$ frame, which corresponds to 0.5s.

3.3.2 Model architecture

The DDQN model architecture is illustrated in Figure 3.2. The input to the neural network consisted of a sequence s_t, in which the three last observations $O_i, i \in \{t, t-1, t-2\}$ (see Section 2.2.1 for further information) were included. The sequence acts as a distinct state, used by the neural network to estimate the action that maximizes the future reward. The first layer of the network is a convolutional layer followed by a ReLU activation layer. This is followed by two hidden fully connected layers, with 256 and 32 neurons each and another ReLU activation layer. The output layer is a fully connected layer, with a single output neuron for each valid action.

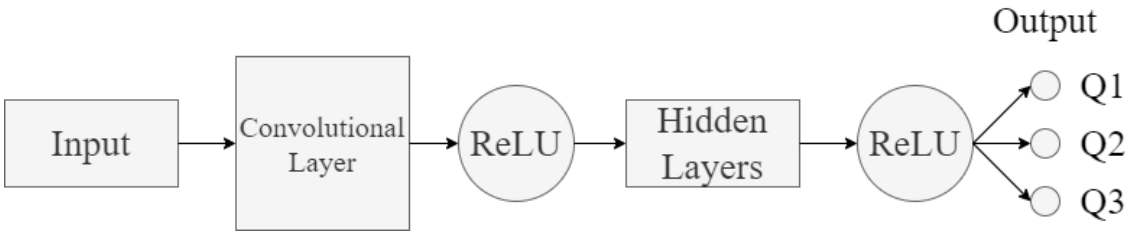


Figure 3.2: Flowchart image of the model architecture. The network takes inputs of size 3x14 into a convolutional layer which convolves into 64 feature maps of size 12 by applying 3x3 filters with stride 1 to the input. This is followed by a ReLU activation function. Thereafter the input is flattened into 768 neurons which enter two fully connected hidden layers with 256 and 32 neurons respectively. Following is another ReLU activation and an output layer which gives the resulting Q-values.

The model architecture, as well as the optimized model parameters, were entirely taken from [9], except for the number of input and output units. The input depends on the state space, which differs for host and ado. Furthermore, the number of output units depends on the action space. Note that optimizing the network and its architecture is not included in the scope of this thesis.

The training loop was implemented in Python using the deep learning package TensorFlow [41].

3.4 Experimental Setup

This section presents the three experiments conducted in this thesis to answer the research questions defined in Section 1.1.

There exist numerous possible experimental setups to use with falsification to find and evaluate critical scenarios. Using the simulation and training details specified in the previous section, three distinct experimental setups were implemented. These three setups consist of training one ado vehicle in isolation, training two agents in a multi-agent system, and finally training V_{host} in an environment with the ado policy, obtained in the first setup, was in place.

3.4.1 Training one ado vehicle

The first situation examined in our work is when V_{ado} induces a collision with V_{host}, by optimizing the policy π_{ado} under which V_{ado} acts. The policy π_{ado} was achieved using the training setup presented in Section 3.2. The host policy π_{host} obtained during training with the safety shield [9] was utilized by V_{host}.

With this setup, experiments using different initial longitudinal positions between the ado and host vehicle were implemented. To obtain a policy $\pi_{ado,ahead}$, V_{ado} was initialized ahead of V_{host} in the target lane. Similarly, to obtain $\pi_{ado,trail}$, V_{ado} was initialized behind V_{host} in the target lane. At last, the initial position was randomized between ahead and behind V_{host} each episode, with the intention to obtain a generalized policy $\pi_{ado,mixed}$. The random initialization was implemented to measure if the framework can be used to train more general policies.

3.4.2 Training multiple ado vehicles

Since the previous presented experimental setup only consists of one ado agent, while the others are PID-controlled, this system is commonly known as a single-agent system. This single-agent system can be transformed to a sort of multi-agent system by enabling several adversarial vehicles to act and be trained using DDQN. Enabling multiple agents to act in the same environment, with a mutual goal, could possibly produce different kinds of critical scenarios which cannot be achieved with a single ado agent. Hence, a multi-agent system consisting of two ado agents - one initialized behind and one ahead of V_{host} in the target lane - was implemented.

In each run, one of the two ado agents acted according to a previously trained policy $\pi_{ado,ahead}$ or $\pi_{ado,trail}$, while the other agent acted according to a policy π'_{ado} which was optimized during the run. The policies $\pi_{ado,ahead}$ and $\pi_{ado,trail}$ were extracted from the setup presented in Section 3.4.1. All other vehicles included in the simulation were PID-controlled. The simulation parameters, other than initial positions, action space and reward function for both of the ado agents were identical during training.

3.4.3 Retrain host vehicle using trained ado

This experiment consisted of retraining V_{host}, at the same time as one ado vehicle operates according to a policy either $\pi_{ado,ahead}$, $\pi_{ado,trail}$ or $\pi_{ado,mixed}$, obtained with the setup presented in Section 3.4.1.

As previously mentioned, V_{host} operates using the policy π_{host} in the other experimental setups, obtained using the same setup as in [9]. In this experiment, the host was retrained starting from policy π_{host}, and finally resulting in a new host policy π'_{host}. This while one ado vehicle was acting according to one of the polices obtained in the first experimental setup. By retraining the host, the intention is to test how the ado policies can be used to change the decision making ability of V_{host}.

The configuration for retraining V_{host} was identical to the configuration presented in [9]. The one and only exception is a slight modification to the reward function. In the previous version, V_{host} was penalized if it crashed with V_{ado} long after completing a lane change. This was adjusted from (3.11) to have a time limit as presented in (3.12) to prevent behavior where the host never commits to a lane change.

$$R_{crash} = \begin{cases} -5, \text{if host vehicle crashes} \\ 0, \text{otherwise.} \end{cases} \tag{3.11}$$

$$R_{crash} = \begin{cases} -5, \text{if host vehicle crashes within 4 seconds of committing to lane change} \\ 0, \text{otherwise} \end{cases} \tag{3.12}$$

3.5 Evaluation

This section introduces the evaluation metrics used to evaluate the results from the experiments. The metrics include falsification success rate, BTN, minimal safe distance and the comparison to the results obtained using the setup presented in [9].

3.5.1 Falsification Success Rate

The main goal of falsification is to provide evidence of faulty behavior by finding a scenario where a specification ψ is violated. To assess how consistent the ado policy π_{ado} performs, the success rate of falsifying the specification ψ_{host} as in Definition 3 is introduced in (3.13).

$$P(\psi_{host} = \text{False}) = \frac{\sum_{\text{all episodes}} \begin{cases} 1 & \text{if } \psi_{host} = \text{false} \\ 0 & \text{otherwise.} \end{cases}}{\text{all runs}} \tag{3.13}$$

As the initial environment differs between episodes, the success rate (3.13) indicates how consistently the policy π_{ado} acts in a way that leads to the specification ψ_{host} being falsified.

3.5.2 Brake Threat Number

As a collision can occur in a multitude of different ways, a method to evaluate the different scenarios is required. Dangerous situations are not always consequences of a dangerous action committed by V_{host}.

Therefore, one metric chosen for evaluating the lane-change scenario was brake threat number (BTN) as described in Section 2.6.1. If a scenario has BTN < 1, it indicates that when the BTN was calculated, it was still possible to avoid the crash. The closer the BTN gets to 1, the harder it is to avoid the collision. An important detail of the BTN is that the calculations are done with the assumption that the leading vehicle will remain at constant acceleration.

The position, velocity and acceleration of the host and ado vehicle at the time step where V_{host} commits to a lane change t_{commit} was recorded. To reduce the effects of fluctuations in the parameters, the parameter values at the next time step $t_{commit+1}$ were also recorded. In the calculation of the BTN the parameter average over the two time steps was used. If V_{ado} was the trailing vehicle, then the BTN was calculated for V_{ado}. Otherwise, the BTN was calculated for V_{host}. As vehicles act the exact moment they observe information in the simulation, the time delay t_d used in calculations was chosen to be 0.

In experiment 3.4.2, there could be two kinds of situations when V_{host} commits to a lane change. One situation is when both of the trained adversarial vehicles are positioned either ahead or behind V_{host}. In this case, the BTN was only calculated between the V_{host} and the adversarial vehicle that was closest to V_{host}. The other situation was when the two trained adversarial vehicles were positioned at each side of V_{host}. This situation was evaluated by calculating both BTNs and recording the maximum of the two values.

In [38] the BTN was presented together with the Steering Threat Number (STN) which is another threat assessment metric which calculates how much a vehicle must steer to avoid a collision. STN was not included in this thesis due to the constraints in lateral movement as presented in Section 3.1.1. If STN is calculated together with BTN, then the total threat assessment is given by $\min(STN, BTN) \leq BTN$. If this method were included instead, then it could possibly have yielded lower scores than what only BTN gives.

3.5.3 Minimal Safe Longitudinal Distance

When committing to a lane change the expectations normally are that the surrounding vehicles will continue driving at the same speed as before. In some cases though, the surrounding vehicles might need to brake with the maximum force. Therefore, to take the worst-case scenarios into account for maximal safety, the minimal safe longitudinal distance d_{min} was also used as an evaluation metric.

To calculate d_{min}, the velocities and positions of the vehicles V_{host} and V_{ado} were recorded and inserted into (2.27) at the time t_{commit}. Similar to the calculation of BTN, the time delay t_d was chosen to be 0, simplifying the calculation of d_{min} to

$$d_{min} = \left[\frac{v_{x,trail}^2}{2a_x^{min}} - \frac{v_{x,lead}^2}{2a_x^{min}} \right]_+, \tag{3.14}$$

where $[x]_+ = \max(x, 0)$. This was calculated for all experiments to present how often the lane change was longitudinally dangerous in the worst-case scenarios.

3.5.4 Retrained Host Performance

The performance of $\pi_{host}^{'}$ acquired when retraining in simulations with an adversarial vehicle following π_{ado} from experiment 3.4.1 must be evaluated fairly. To accomplish this, the same metrics used to evaluate the performance in [9] were used. The metrics consist of lane changing success rate, the average time to finish a lane change and crashing rate. These metrics will be tested in the original environment without any trained adversarial agents as in Section 2.5, in addition to testing in the environment with a trained adversarial present from Section 3.4.1.

4

Results

This section presents the results for the reinforcement learning based falsification experiments presented in Section 3.4.

The results presented in this section, for all three experiments, were obtained by evaluating the resulting policies over 1 000 episodes. The evaluation scenarios were created with the same seed for all the experiments. This means that the initial values, except for which vehicle the ado policy controls, are the same. These initial values were excluded during training to ensure that the evaluation scenarios are unseen.

The results are presented by showing data and criticality from the observed crashes, using BTN as the primary metric. The BTN for all the crashes is visualized by plotting BTN against ΔT_{commit_crash}, defined as the time difference between when the host committed to lane change and the time of the crash. In the experiments where V_{ado} was behind V_{host}, the BTN was calculated for V_{ado}, in the other case, the BTN was calculated for V_{host}. In addition to this, visual interpretations of critical scenarios in each setting are shown. Furthermore, acceleration, velocity, and distance profiles are presented for critical scenarios in comparison with non-critical scenarios.

4.1 Training one ado vehicle

This section includes results for the setup where one ado vehicle was trained to falsify Definition 3. The results are presented for three different initial values on the position of V_{ado}. The obtained policies are denoted $\pi_{ado,trail}$, $\pi_{ado,lead}$, $\pi_{ado,mixed}$ to separate the policies for each initial position. All of the experiments used the same range of initial parameter values as well as the same reward function.

All training of the different policies was done over 500 000 time steps, which creates 10 000 - 15 000 episodes depending on the average episode length.

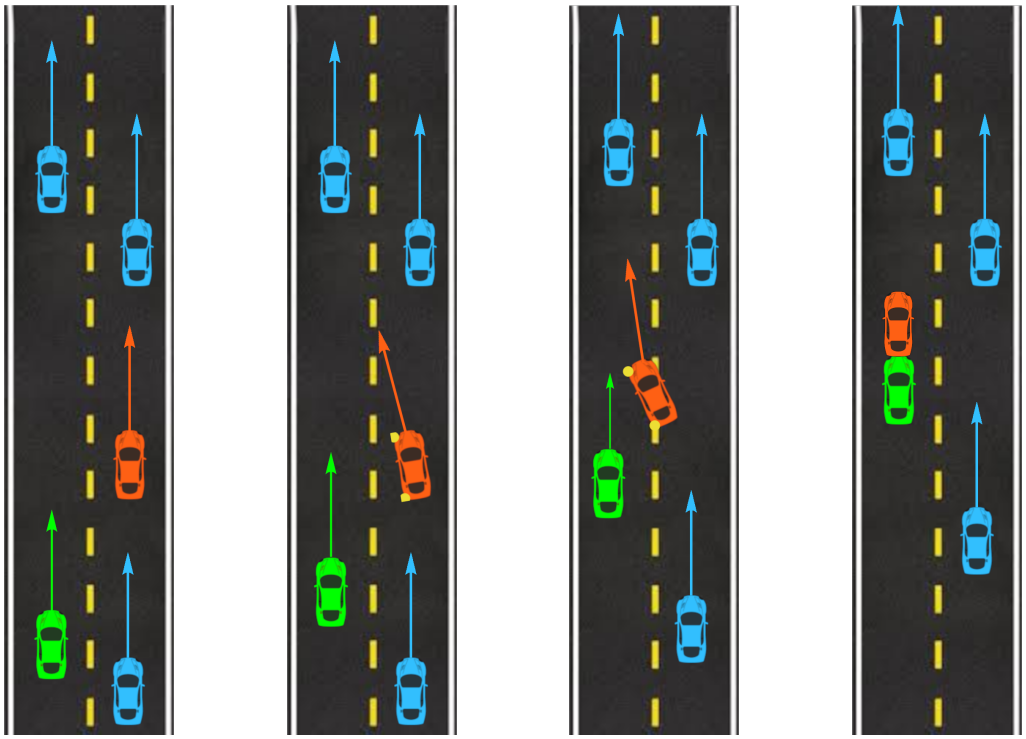
A summary of the results is first presented in Table 4.1 followed by a detailed description later. The crash rate for the different initial positions differed slightly. Regarding the BTN, three times as many crashing scenarios were found with BTN = 1 for V_{ado} initialized behind, compared to when the vehicle was initialized either ahead or with mixed initialization each episode.

Table 4.1: Table containing evaluation values from each of the different policies when training one adversarial vehicle.

Evaluation metric	Policy π_{ado}		
	$\pi_{ado,trail}$	$\pi_{ado,lead}$	$\pi_{ado,mixed}$
Crash rate [%]	69.8	87.2	59
BTN = 1	94	31	33
$r_{host\_ado} < d_{min}$	66	148	14

4.1.1 Trailing vehicle in target lane as ado

In this experimental setup, the starting position of V_{ado} in the target lane was behind V_{host}. Through training as described in Section 3.4.1, $\pi_{ado,trail}$ was formed. An example of a sequence of events triggered by actions from V_{host} and V_{ado} that leads to a collision in the observed episodes is shown in Figure 4.1. In the beginning of the scenario in Scene 4.1a, V_{ado} is driving with a higher velocity than V_{host}. In Scene 4.1b, V_{host} has evaluated the situation to result in a safe lane change, thus signaling to turn and committing to the lane change. In Scene 4.1c, V_{host} changes lane, resulting in a crash in Scene 4.1d. The behavior of V_{ado} before V_{host} commits to lane change leads V_{host} to evaluate the scene as safe when it resulted in a dangerous lane change.



(a) Agents on road, V_{ado} with higher velocity.

(b) V_{host} commits to lane change.

(c) V_{host} performs lane change.

(d) Lane change completed, resulted in crash.

Figure 4.1: Scenario of car crash with V_{ado} initialized from behind. The green vehicle represents V_{ado}, while the orange vehicle represents V_{host}. The blue vehicles are PID-controlled. Each vehicles respective arrow represents their trajectory vector.

The policy $\pi_{ado,trail}$ resulted in 698 crashing scenarios during evaluation, which corresponds to a crash rate of 69.8%. The performance of the crashes are plotted in Figure 4.2, visualizing BTN as a function of ΔT_{commit_crash}. As seen in Figure 4.2, the BTN is 1 in majority of crashes when ΔT_{commit_crash} is between 5 and 7. These crashing scenarios, when BTN equals 1, sum up to 94 scenarios. The BTN of the remaining scenarios is less or equal to 0.4. Consequently, in these scenarios, V_{ado} is able to avoid the collision by braking.

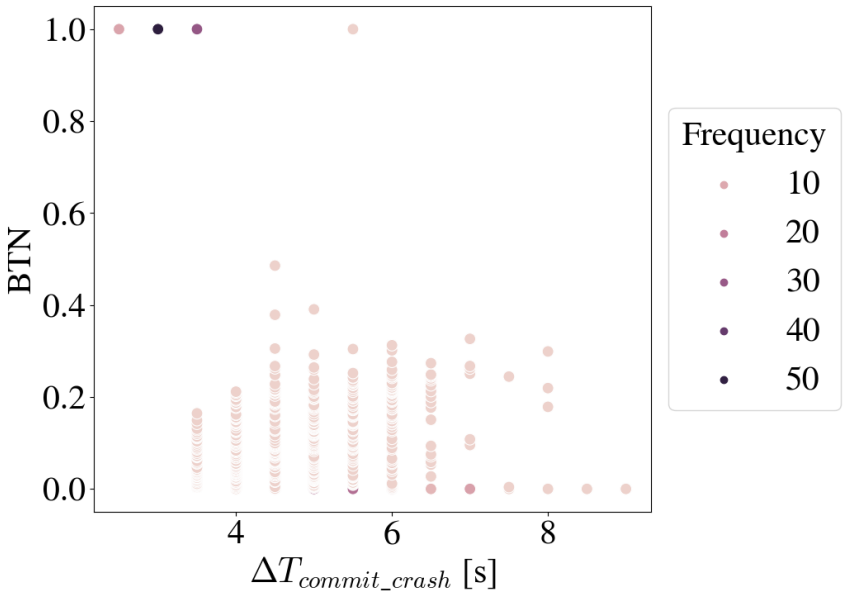


Figure 4.2: BTN against the time difference from that host committed to lane change until the until collision occurs, when V_{ado} is initialized behind of V_{host}.

In addition to BTN, the number of episodes where $r_{host\_ado} < d_{min}$ summed up to 66 (9.5% of crashes).

Furthermore, the acceleration, velocity and distance profiles for V_{ado} and V_{host} as a function of time t is shown in Figure 4.3. The data used in the plot are extracted from scenarios where BTN=1 and $\Delta T_{commit\_crash} = 3.5$, see Figure 4.2 for reference. The leftmost plot visualizes the median velocity and acceleration, together with their respective deviations. The rightmost plot visualizes the distance r_{host_ado} between V_{host} and V_{ado}. The two vertical lines represent the time where V_{host} committed and completed the lane change.

During the whole time frame, V_{ado} drives with a higher velocity than V_{host}. This behavior leads to a situation where a crash is unavoidable since V_{ado} is the trailing vehicle. This is in contrast to when BTN$\neq$1, presented in Figure 4.4, where both vehicles maintain similar velocity until t_{commit}. Another difference between the two situations is the distance r_{host_ado} before V_{host} committed to lane change. When BTN=1, (see Figure 4.3), the distance is $r_{host\_ado} \approx 27$ meters, compared to $r_{host\_ado} \approx 20$ meters when BTN$\neq$1. These distances are observed 2.5 seconds before t_{commit}. The large distance allows for V_{ado} to reach higher velocity without colliding with any vehicle. V_{ado} has in these situations either managed to create this distance by decelerating, or by having different initial conditions.

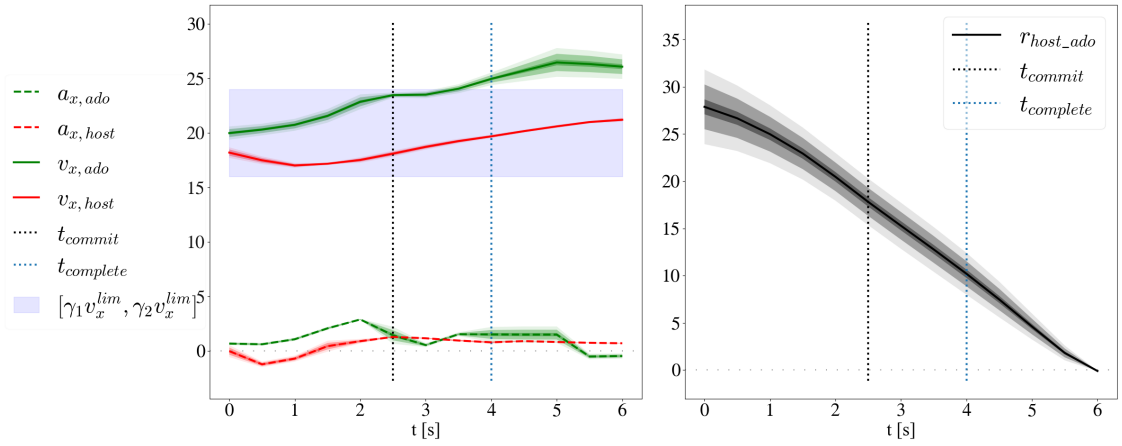


Figure 4.3: $a_{x,ado}, a_{x,host}, v_{x,ado}$ and $v_{x,host}$ as a function of time in a crashing scenario in the left plot. r_{host_ado} as a function of time in the right plot. The median values are presented with the solid and dashed lines, while the corresponding colored regions represent the 25-75, 35-65 and 45-55 percentile from light to dark. The transparent light blue area shows the range for the soft velocity bounds as presented in Section 3.1.1. The black and blue dotted lines shows t_{commit} and $t_{complete}$ respectively.

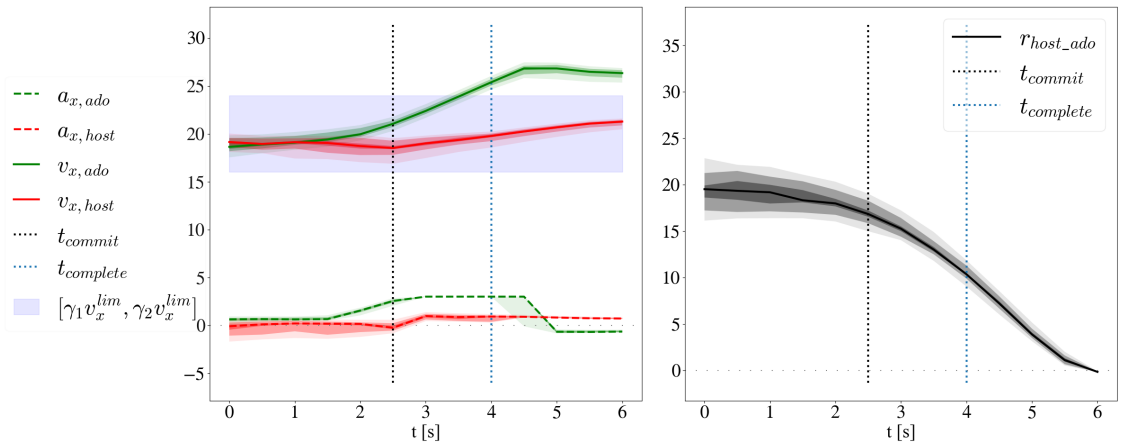
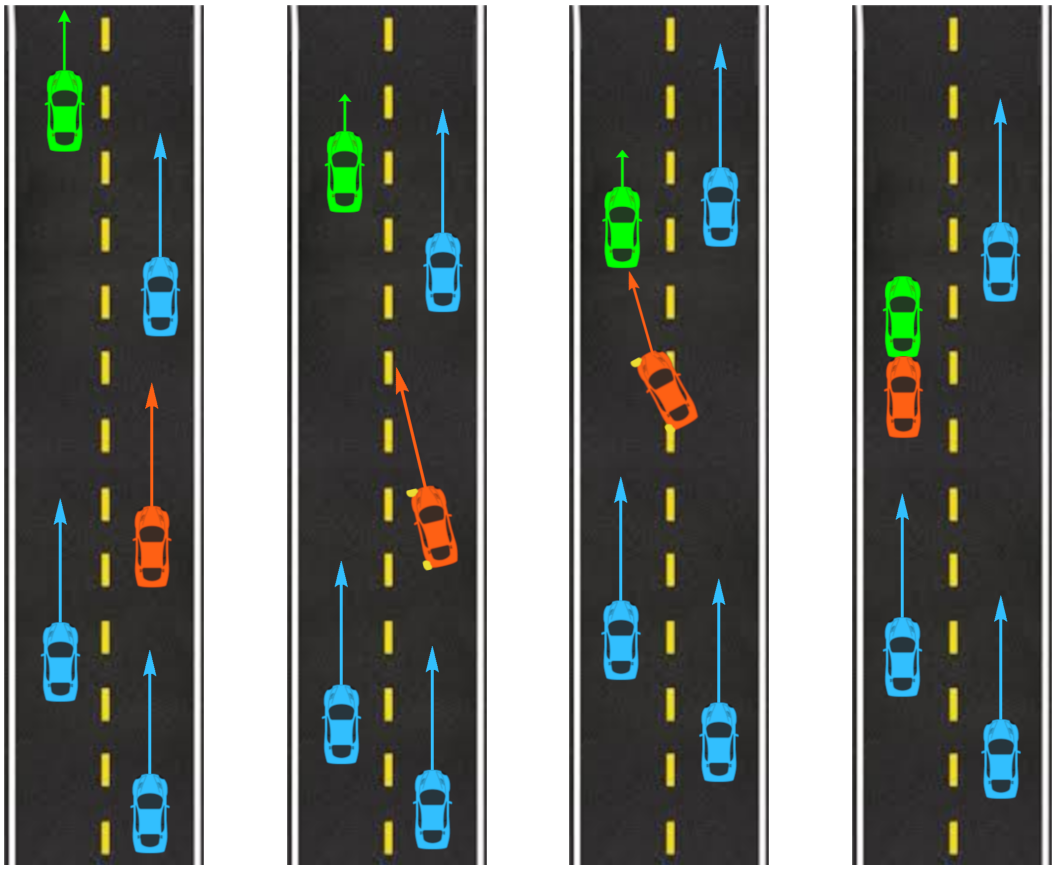


Figure 4.4: Acceleration, velocity and distance plots for $\Delta T_{commit\_crash} = 3.5$ and $BTN \neq 1$. Legend is the same as in Figure 4.3.

4.1.2 Leading vehicle in target lane as ado

In this section, the results for the experiment where V_{ado} was initialized ahead of V_{host} are presented. The ado policy that was obtained during training is denoted as $\pi_{ado,lead}$. An example of a scenario that leads to a collision when V_{ado} is driving according to $\pi_{ado,lead}$ is shown in Figure 4.5. At the start of the scenario, in Scene 4.5a, V_{ado} and V_{host} are driving in separate lanes with $v_{x,host} > v_{x,ado}$ and $a_{x,ado} < 0$. In Scene 4.5b, V_{host} has signaled to turn and evaluated the situation as safe, thus committing to changing lane. Meanwhile $a_{x,ado}$ is still negative. In Scene 4.5c the lane change is performed, resulting in a crash in Scene 4.5d. The scenario shows that due to π_{host} evaluating the situation as safe for changing lanes, V_{host} is being put in a position were it cannot avoid collision.

During the evaluation loop, $\pi_{ado,lead}$ resulted in V_{host} crashing in 872 of 1 000 episodes. This gives a crash rate of 87.2 %. The performances of these crashes are shown in Figure 4.6. The plot visualizes the BTN as a function of ΔT_{commit_crash}. Adding all scenarios where the BTN was 1, 31 scenarios were found.



(a) Agents on road. **(b)** V_{host} commits to lane change. **(c)** V_{host} performs lane change. **(d)** Lane change completed, resulted in crash.

Figure 4.5: Scenario of car crash with V_{ado} initialized from ahead. The green vehicle represents V_{ado}, while the orange vehicle represents V_{host}. The blue vehicles are PID-controlled. Each vehicles respective arrow represents their trajectory vector.

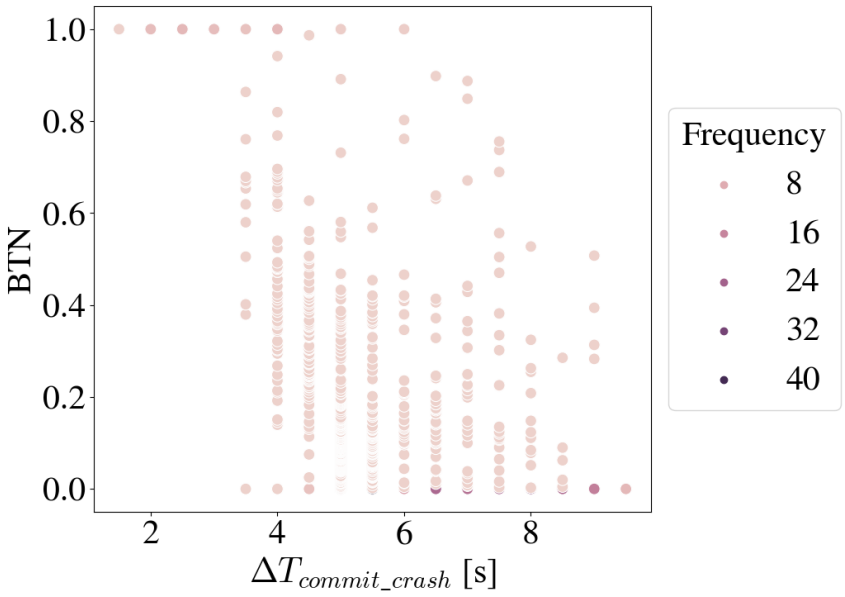


Figure 4.6: BTN against the time difference from that host committed to lane change until the until collision occurs, when V_{ado} is initialized ahead of V_{host}.

In order to visualize the behavior of V_{ado} and V_{host} in the critical scenarios, the accelerations, velocities, and distances between the vehicles are shown in Figure 4.7. In the leftmost plot, the median together with deviations of speed and acceleration values are plotted. In the same way, the right plot visualizes the absolute distance r_{host_ado} between V_{ado} and V_{host}. Two vertical lines are plotted in both figures, representing the time where V_{host} committed to lane change, t_{commit}, and time of completed lane change $t_{complete}$. Figure 4.7 was created using the scenarios where $BTN = 1$ and $\Delta T_{commit\_crash} = 2$. Note that $v_{x,host}$ is at the upper bound of the preferred velocity in the first three seconds of the scenario. This is caused by V_{host} wanting to find space ahead of another adversarial vehicle that is not V_{ado} in the target lane to complete the lane change.

In comparison, for $BTN \neq 1$ and $\Delta T_{commit\_crash} = 4$ the scenarios are shown in Figure 4.8. Specifically, the acceleration profile for $a_{x,ado}$ is differentiates considerably in Figure 4.8 as it reaches a_x^{min} after t_{commit}. This is in contrast to Figure 4.7 where $a_{x,ado} = a_x^{min}$ throughout the scenario until t_{commit}. In Figure 4.8, $v_{x,host}$ is not particularly high or low in the first few seconds of the scenario. This is because the space for making a lane change was created by a yielding adversarial agent that was not V_{ado}.

4.1.3 Trailing or leading vehicle in target lane as ado

In this section, the results for when V_{ado} was initialized either behind or ahead of V_{host} at random are presented, with policy $\pi_{ado,mixed}$.

The evaluation results for $\pi_{ado,mixed}$ is presented in Figure 4.9, visualizing the BTN against ΔT_{commit_crash}. With this setup, a crash scenario occurred in 590 out of 1 000 evaluation episodes, where the BTN was 1 in 33 of those scenarios. The remaining crashing scenarios resulted in a variety of BTN, covering the whole BTN spectrum.

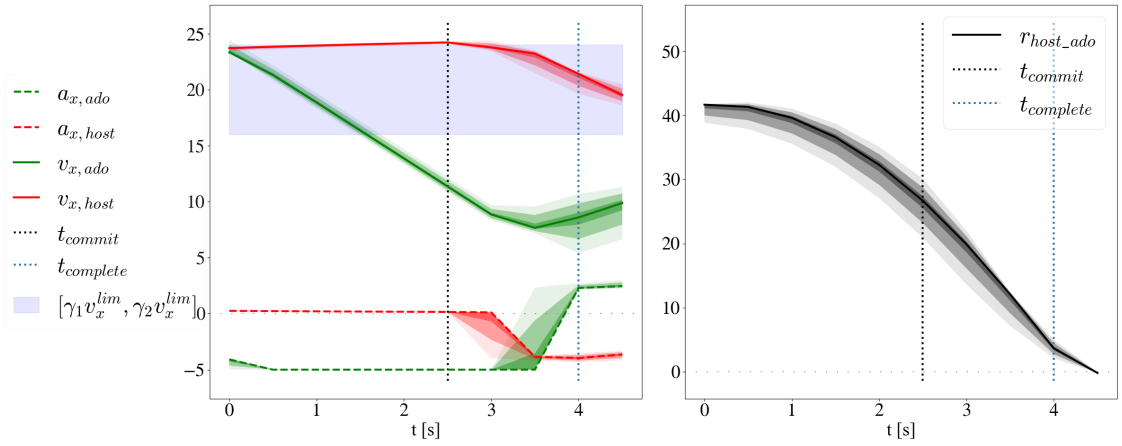


Figure 4.7: $a_{x,ado}, a_{x,host}, v_{x,ado}$ and $v_{x,host}$ as a function of time in a crashing scenario in the left plot. r_{host_ado} as a function of time in the right plot. The median value are presented with the solid and dashed lines, while the corresponding colored regions represent the 25-75, 35-65 and 45-55 percentile from light to dark. The transparent light blue area shows the range for the soft velocity bounds as presented in Section 3.1.1. The black and blue dotted lines shows t_{commit} and $t_{complete}$ respectively.

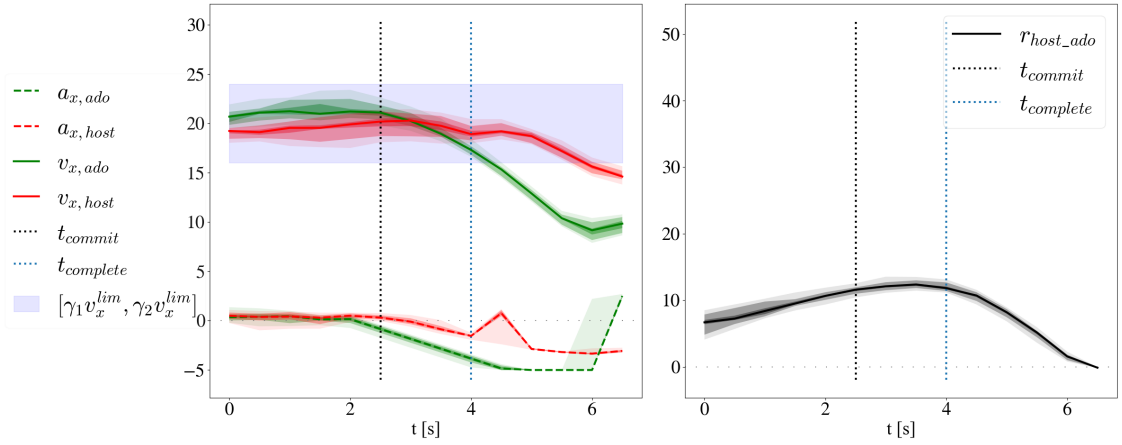


Figure 4.8: Acceleration, velocity and distance plots for $\Delta T_{commit\_crash} = 4$ and $BTN \neq 1$. Legend is the same as in Figure 4.7.

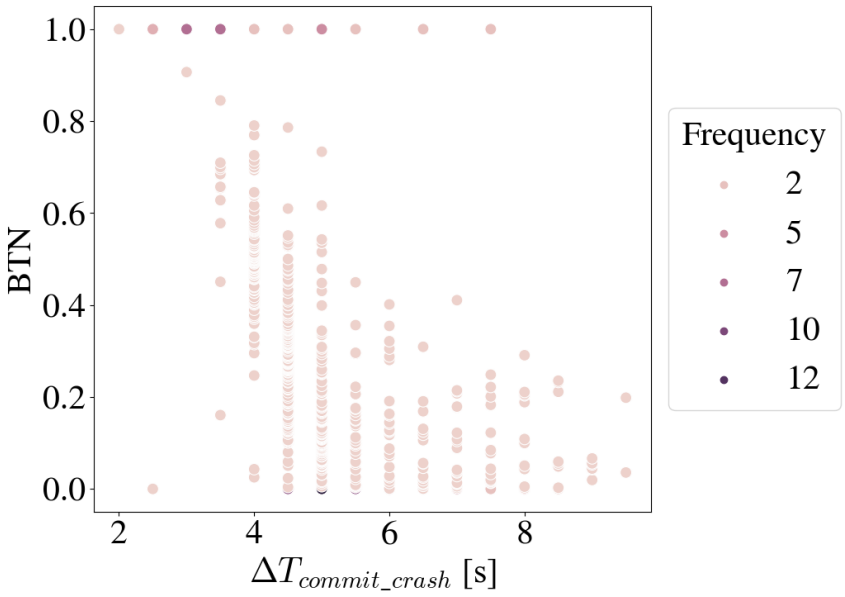


Figure 4.9: BTN against the time difference from that host committed to lane change until the until collision occurs, when V_{ado} is initialized behind and ahead of V_{host} randomly each episode.

4.2 Training multiple ado vehicles

This section includes evaluation results for the experiments where two adversarial vehicles, V_{ado} and $V_{ado}^{'}$, were controlled by an RL policy. The results presented in this section were obtained using the experimental setup described in Section 3.4.2. The experiments uses $\pi_{ado,lead}$ and $\pi_{ado,trail}$ extracted from the experiments in Section 4.1.

Firstly, we present the results where the policy $\pi_{ado,trail}$ used by the vehicle behind the host, $V_{ado}^{'}$, was optimized. The ado vehicle ahead of host V_{ado} in the target lane was operating using policy $\pi_{ado,lead}$. Secondly, the vehicle ahead of the host was trained, now denoted $V_{ado}^{'}$, while the vehicle behind of host, V_{ado} was driving with the policy $\pi_{ado,trail}$.

The training of the different policies was done over 500 000 time steps, which creates 10 000 - 15 000 episodes depending on the average episode length.

A summary of the evaluation results is presented in Table 4.2. BTN was calculated for both the trailing and leading vehicle, and the maximum BTN of these two was registered. In a similar way, the number of episodes where one of the two vehicles satisfies $r_{host\_ado} < d_{min}$ were registered.

Table 4.2: Summary of the results for the multi ado experimental setup. The table shows results for both $\pi^{'}_{ado,trail}$ and $\pi^{'}_{ado,lead}$.

Evaluation metric	Policy $\pi^{'}_{ado}$	
	$\pi^{'}_{ado,trail}$	$\pi^{'}_{ado,lead}$
Crash rate [%]	60.9	89.3
BTN = 1	14	72
$r_{host\_ado} < d_{min}$	81	104

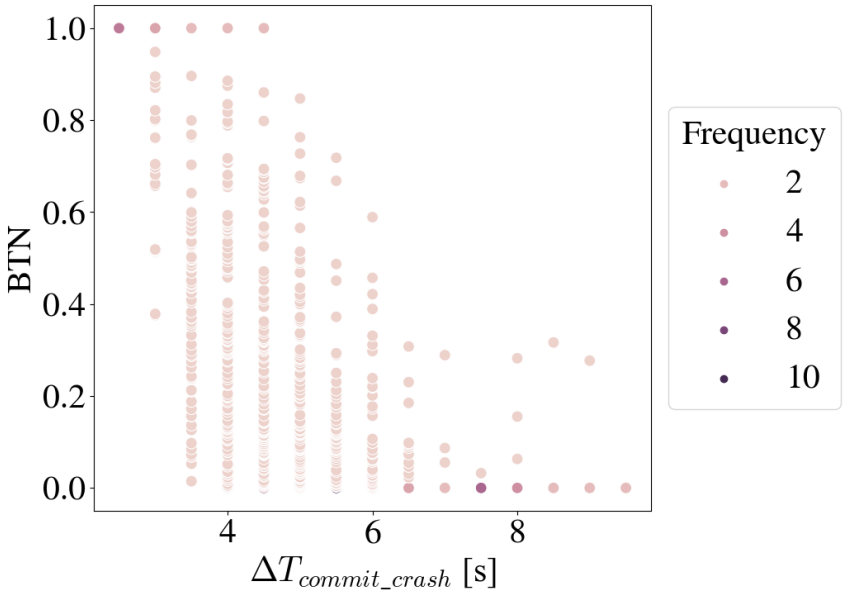


Figure 4.10: BTN against time for the crashing scenarios with V_{ado} ahead, and $V^{'}_{ado}$ behind of V_{host} is trained.

4.2.1 Training policy for trailing ado in the target lane

The results of training $\pi^{'}_{ado,trail}$ by using previous policy $\pi_{ado,lead}$, are presented in this section. The total number of crashes between either V_{ado} or $V^{'}_{ado}$ and V_{host} was 609, out of 1 000 evaluation episodes. In Figure 4.10, the performance of the crashes are shown, using BTN as metric, against ΔT_{commit_crash}. The number of crashing scenarios where BTN=1 is 14.

Furthermore, V_{host} committed to a lane change when $r_{host\_ado} < d_{min}$ in 81 of the crashing scenarios, which corresponds to 13.3% of the crashes.

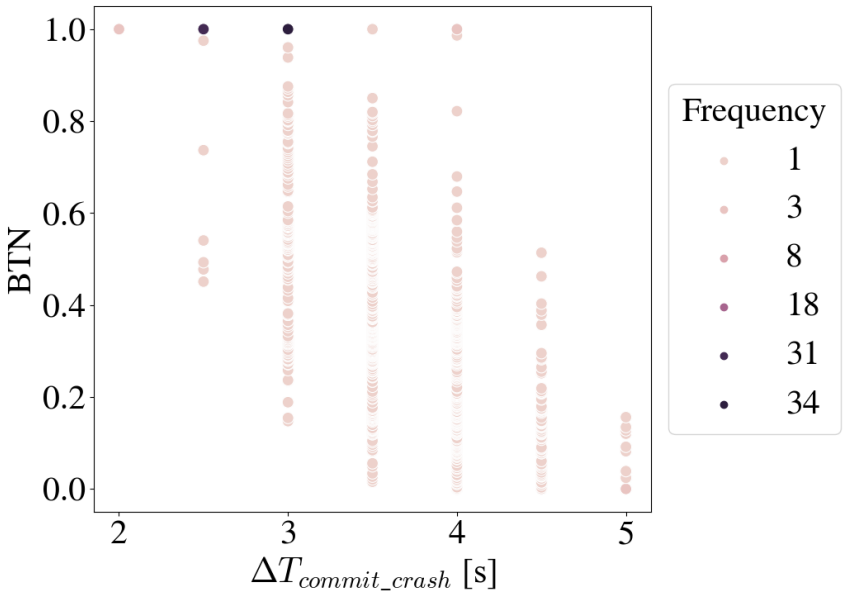


Figure 4.11: BTN against time for the crashing scenarios with V_{ado} behind, and $V_{ado}^{'}$ ahead of V_{host} is trained.

4.2.2 Training policy for leading ado in the target lane

The results for the setup where $\pi_{ado,lead}^{'}$ was trained, and V_{ado} behind of host operates using $\pi_{ado,trail}$ are presented in this Section. To start with, 893 crashing scenarios were found out of 1 000 evaluation episodes. Out of these crashing scenarios, the BTN was 1 for 72 scenarios. The BTN for every crashing scenario is shown in Figure 4.11.

Regarding the third evaluation metric, $r_{host\_ado} < d_{min}$ was satisfied in 104 crashes, which translates to 11.6% of the found crashing scenarios.

4.3 Retrain host vehicle using trained ado

This section presents the experiment results from retraining V_{host} (further described in Section 3.4.3) in an environment where all the ado vehicles are PID-controlled, except for one ado vehicle V_{ado} operating by $\pi_{ado,trail}$ or $\pi_{ado,lead}$.

The retrained host policy $\pi_{host}^{'}$ was obtained by training for 100 000 time steps, corresponding to approximately 3 000 - 4 000 episodes. The original host policy π_{host} [9] was used to initialize $\pi_{host}^{'}$, so that $\pi_{host}^{'} = \pi_{host}$ at the first training step. The training and the evaluation were done with V_{ado} initialized as ahead of V_{host}, behind, and with a mixed position.

As previously mentioned in Section 3.5.4, the retrained host was evaluated in two different simulation setups. A summary of the evaluation results obtained testing the retrained policy in the original environment presented in [9] is presented in Table 4.3. Note that the metrics presented in Table 4.3 were calculated using 100 evaluation rounds, to replicate

the evaluation performed in [9]. Furthermore, a summary of the results evaluating $\pi_{host}^{'}$ with adversarial agents operating under π_{ado} is presented in Table 4.4.

Table 4.3: Table containing evaluation metrics for the original π_{host} and retrained host $\pi_{host}^{'}$. The retrained host was trained with three different ado policies. The evaluation is performed in a simulation comparable to how π_{host} was evaluated, where all the adversarial vehicles are PID-controlled.

Evaluation metric	Original host	Retrained host Policy π_{ado}		
		$\pi_{ado,trail}$	$\pi_{ado,lead}$	$\pi_{ado,mixed}$
Lane change rate [%]	96	100	96	97
Average time until lane change [s]	20.5	19.5	21.2	17.3

Table 4.4: Table containing evaluation metrics for the original host using policy π_{host} and for the retrained host, policy $\pi_{host}^{'}$ trained with three different ado policies.

Evaluation metric	Original host Policy π_{ado}			Retrained host Policy π_{ado}		
	$\pi_{ado,trail}$	$\pi_{ado,lead}$	$\pi_{ado,mixed}$	$\pi_{ado,trail}$	$\pi_{ado,lead}$	$\pi_{ado,mixed}$
Lane change rate [%]	84	93.5	69.8	93.7	79.5	93.2
Crash rate [%]	69.8	87.2	59	90	65.6	91.4
BTN = 1	94	31	33	8	1	5
Average time until lane change [s]	13.1	21.2	13.8	12	22.9	12.7
$r_{host\_ado} < d_{min}$	66	148	14	181	27	6

4.3.1 Retraining host while trailing ado in target lane

When using $\pi_{ado,trail}$, while V_{host} operates using $\pi_{host}^{'}$, the two vehicles crashed 900 out of the 1 000 evaluation rounds. Out of the 900 crashing scenarios, 8 resulted in a BTN of 1. The calculated BTN for all crashing scenarios can be seen in Figure 4.12, as a function of ΔT_{commit_crash}. The majority of crashes resulted in BTN ≤ 0.4.

Furthermore, as described in Section 3.5.4, the host policy evaluated above was also evaluated in the same environment as π_{host}. With that environment, the new policy $\pi_{host}^{'}$ was able to perform a successful lane change in 100% of the 100 evaluation episodes. These lane changes were performed, on average, 19.5 seconds after the start of the episode. The lane change rate and the average time until lane change are slightly improved compared to the original host (see Table 4.3).

4.3.2 Retraining host while leading ado in target lane

In this experiment, the host was retrained in an environment with $\pi_{ado,lead}$. Out of the 1 000 evaluation episodes, the ado and host vehicle crashed 656 times, which corresponds to a crash rate of 65.6%. The performance of these crashes is visualized in Figure 4.13, where the performance is evaluated using BTN. The BTN is plotted as a

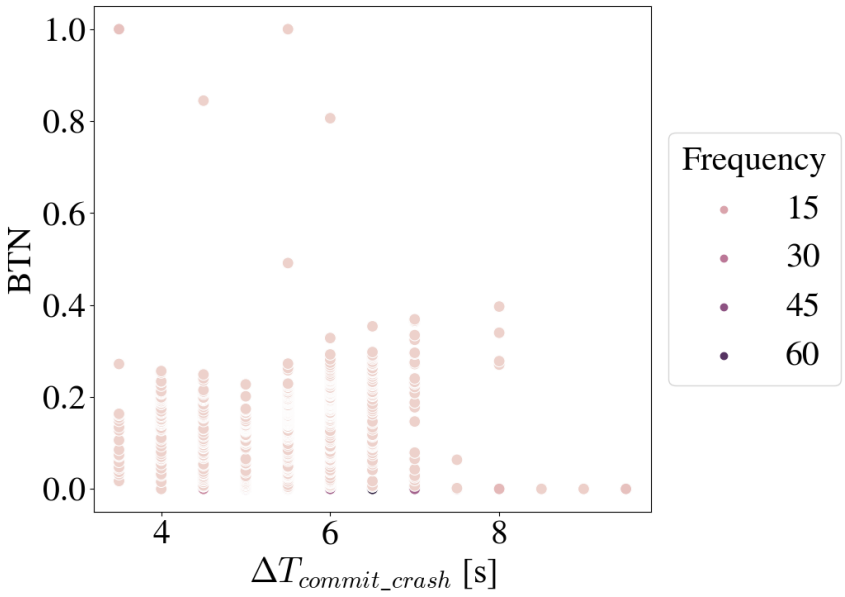


Figure 4.12: BTN against ΔT_{crash_commit} for the crashing scenarios where V_{host} operates with the policy $\pi_{host}^{'}$ and V_{ado} is using $\pi_{ado,trail}$.

function of ΔT_{commit_crash}. Out of the 656 crashes, 1 crash resulted in a BTN of 1, at $\Delta T_{commit\_crash} \approx 2$. The majority of crashes resulted in BTN ≤ 0.5.

The retrained V_{host} was able to change lane in 96% of the 100 evaluation scenarios, see Table 4.3. These lane changes were on average performed after 21.2 seconds. These results are close to identical to the performance of the original π_{host} policy.

4.3.3 Retraining host while ado is trailing or leading in target lane

The performance of the retrained host, calculated using BTN, while V_{ado} is using $\pi_{ado,mixed}$ is presented in Figure 4.14. The vehicles collided in 844 of 1 000 evaluation episodes, which of 5 resulted in BTN=1. As can be seen in Figure 4.14, most of the crashes resulted in BTN$\leq$ 0.5.

With policy $\pi_{host}^{'}$, the vehicle succeeded with a lane change in 97% of the 100 evaluation episodes, taking on average 12.3 seconds to perform the lane change. This is comparable to the performance of the original host, with policy π_{host}. The average time until lane change is slightly improved.

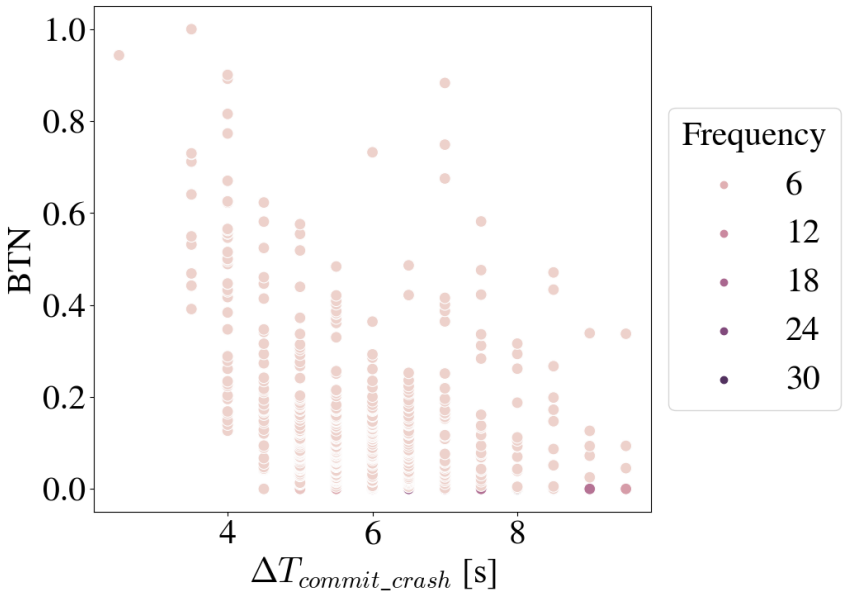


Figure 4.13: BTN against ΔT_{crash_commit} for the crashing scenarios where V_{host} operates with the policy π'_{host} and V_{ado} is using $\pi_{ado,lead}$.

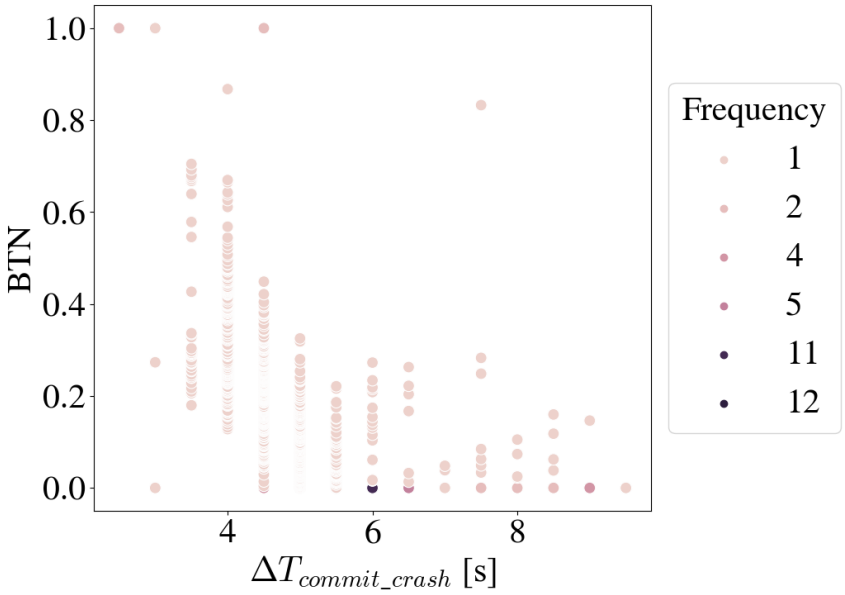


Figure 4.14: BTN against ΔT_{crash_commit} for the crashing scenarios where V_{host} operates with the policy π'_{host} and V_{ado} is using $\pi_{ado,mixed}$.

5

Discussion

This chapter includes a discussion and analysis of the results presented in Chapter 4. In addition, different aspects of the methodology are discussed, in particular possible drawbacks and future improvements.

5.1 Results discussion

This section will discuss the results of the experiments case by case. In general, when working with RL, one issue is consistency. As many of the processes are stochastic, it is difficult to produce consistent results. In addition to this, the RL method is highly vulnerable to hyper-parameter changes which can drastically change the results.

5.1.1 One ado vehicle

The results for the experiments where one ado vehicle is trained are shown in Section 4.1. As has also been shown in previous works, for example [28], Deep RL is able to find counterexamples to the safety validation specification. The values provided in Table 4.1 show that in this framework, the position in which the controlled ado vehicle is initialized plays a key role in how well it performs.

To begin with, the results for policy $\pi_{ado,trail}$ suggest that a multitude of critical scenarios can be found, as can be observed in Figure 4.2. Furthermore, as can be seen in Figures 4.3 and 4.4, the driving profile differs noticeably for scenarios with BTN=1 and BTN≠1. One insight that these plots suggest is that scenarios where V_{host} commits to a lane change when $v_{x,ado} >> v_{x,host}$, which in turns leads to r_{host_ado} shrinking rapidly, there will be an unavoidable crash. This result suggest that improvements can be made in the calculations of the safety shield, as it's purpose is to prevent the situations showed in Figure 4.2.

For when ado vehicle is ahead, as seen in Figures 4.7 and 4.8, there are obvious trends in $a_{x,ado}$ and $v_{x,ado}$ that leads to different values in BTN and ΔT_{commit_crash}. Notably, the most dangerous situations for V_{host} to commit to lane change is when large differences in $v_{x,ado}$ and $v_{x,host}$ are present. In Figure 4.7, $v_{x,host}$ is very high due to V_{host} trying to find space ahead of an adversarial vehicle in the target lane. This behavior is not present in Figure 4.8. The adversarial agent that "blocks" the lane change in Figure 4.7 also enables V_{ado} to have more time to create larger distances as seen on the right side of the plots. From this, the conclusion can be drawn that the severity of the scenarios is directly correlated to not only π_{ado} and π_{host}, but also the other adversarial vehicles

in the densely packed traffic situation. To find more consistent results, either changing the behavior of the PID-controlled adversarial vehicles to be different from [9] or using a cooperative multi-agent RL might be useful.

In comparison to the other two policies $\pi_{ado,trail}$ and $\pi_{ado,lead}$, the policy that was trained when initialized randomly, $\pi_{ado,mixed}$, presented in Section 4.1.3 performed worse in regards to the crash rate metric and $r_{host\_ado} < d_{min}$. This result can be interpreted as in this framework, training a policy to find diverse counter-examples in a more generalized setting might be less effective than training for one particular situation. This is in line with what should be expected, as Deep RL is generally most effective when focusing on a single goal in a single environment. Several papers, such as [28, 42], addresses the known issues with generalization, and how to possibly solve them. Obtaining generalized policies is of great importance, especially in real world applications. Nevertheless, the BTN was similar to the BTN for $\pi_{ado,lead}$. This could be explained through similar arguments as previously, that it has difficulties with generalization, but performs well in a few scenarios.

However, this framework is still shown to be able to find diverse counterexamples when focusing on a single goal as shown in both Sections 4.1.1 and 4.1.2. This result might suggest that the framework also can be used to train ado policies in different environments altogether.

5.1.2 Multiple Ado Vehicles

The results for the experiments where multiple adversarial agents were implemented are shown in Section 4.2. The results show that the approach presented in Section 3.4.2 is a functioning way of implementing multiple agent RL in this framework. In comparison to the results for the single ado agent policy experiment, the experiments with multiple ado policies produced comparable results according to our evaluation metrics, albeit somewhat lower frequency of scenarios with BTN = 1. A possible cause of the decline in performance is due to how RL is fundamentally vulnerable to perturbations in the environment. When the additional control policy $\pi^{'}_{ado}$ was added to the simulation, it added a different behavior which the previously obtained control policy π_{ado} was not used to, causing a decline in performance according to our evaluation metrics.

However, as the new policy $\pi^{'}_{ado}$ did not act in the same way as the PID-controlled agents, this experiment did undoubtedly find unique scenarios that could not be found in the experiments where only one ado policy π_{ado} is in use (Section 4.1). To broaden the scope of possible scenarios that could be found even further, more vehicles could be controlled using control policies.

Multi-Agent Reinforcement Learning (MARL) [43] is a subfield in reinforcement learning that can train multiple agents at the same time. This makes it possible for the agents to cooperate to reach their mutual goals. To improve on our framework, a cooperative MARL approach might produce scenarios where the multiple ado agents cooperate to find unique solutions to the falsification problem which cannot be achieved with the current methodology.

5.1.3 Retrained Host

The results from the experiments from retraining V_{host} is shown in Section 4.3. These results show that the number of situations where the BTN is 1 can be drastically reduced by retraining π_{host}. One thing to consider when analyzing these results is that V_{host} trains in an environment where the ado policy is the same as in the evaluation environment. This methodology may lead to a host policy that is overfitted to the specific kind of dangerous situations that the ado policy produces.

Future work on these experiments could include an iterative process. By having an ado policy be trained in the same way as the new host policy and then training V_{host} once again. This could show if using this method in multiple iterations would result in a host policy that makes falsification harder to perform.

To achieve good results with such a method, increasing the consistency of the model while maintaining the possibility to find unique situations is favorable. In our work, the ado model obtained using the experimental setup presented in Section 3.4.1 is used in Section 3.4.3. Consequently, the results obtained in the latter experiment heavily depend on the result of the first experiment - which underlines the importance of consistent and reliable models. This is of increased significance if an iterative process were to be implemented, as errors generally tend to increase through iterative processes.

5.2 Method discussion

This section will discuss the methodology implemented in this thesis to answer the research questions presented in Section 1.1. This discussion will focus on possible improvements and drawbacks of the method, as well as how the results can be interpreted in relation to the method.

5.2.1 Action space

In our work, several assumptions are made to obtain a simulation environment as realistic as possible. However, it is difficult to be certain that these assumptions reflect the real world, especially in specific scenarios. One assumption we make is regarding the maximum allowed jerk, which is assumed to not exceed $|j| < 2\,\mathrm{m/s^3}$. In [40], it is stated that exceeding the jerk limit could result in an uncomfortable driving experience. However, using this as a hard bound might be too restrictive, especially in maximum braking scenarios. In such scenarios, the comfort of the passengers should be compromised, especially if a critical scenario could be avoided. In addition, the jerk limit in previous research including the BTN [44] invites much larger values of braking. Consequently, modifications to the presented action space in Section 3.2.1 is possible. Changing the allowed jerk could probably change the behavior of V_{ado}, allowing the agent to find even more critical scenarios. The implementation of an action space that does not set a hard bound on the jerk has been tested in this project, as discussed in Section 3.2.1, however without success. One possible reason is the increased complexity of that action space.

5.2.2 Reward function

The reward function was designed having three aspects in mind. First and foremost, the agent must be reinforced to falsify Definition 3. In addition to this, the agent should be guided to find critical scenarios. Finally, the agent is expected to not violate physical constraints, and behave realistically. Formulating a reward function that guides the agent as is intended is one of the many difficulties with RL. In addition, several hyper parameters are included in the reward, all obtained by trial and error using this environment.

Previous research has shown robustness-guided falsification to be feasible, using the STL formulation. During the scope of this project, it has not been possible to investigate this approach in our simulation environment. However, the formulation has the potential to be successful, enabling a more rigorous method of specifying safety specifications.

5.2.3 Double Deep Q-Network

In this thesis, both the host policy π_{host} and the different ado policies π_{ado} are operated by DDQN. While DDQN has been utilized in previous research on similar projects [8, 28], other Deep RL algorithms have also been utilized such as TRPO [27] and A3C [8]. Which Deep RL algorithm works best tends to give different answers based on the problem to be solved. Therefore, experiments using a different kind of algorithm might yield stronger or weaker results. This can only be concluded by extensive testing of different algorithms, in combination with hyper-parameter tuning. Consequently, investigating this in the future is of interest. With our framework, it is possible to implement other Deep RL algorithms, however, some modifications to the framework are required.

When using the original DDQN method, one is limited to using a discrete action space which works well for some cases, especially if semantic actions are utilized. But in problems like in this thesis, where the action space is based on change of acceleration, a policy gradient algorithm can be implemented to make a continuous action space possible. This would enable V_{ado} to find different behavior patterns as the range of possible actions increases. This could also be achieved by implementing the approximate method NAF [45].

5.2.4 Evaluation metric

For evaluating falsification results, there are infinite possibilities of what metrics to be used. The metric BTN used in this thesis, has been used in other research projects [44, 46] often together with the Steering Threat Number (STN). As mentioned in 3.5.2, STN was not included in this project due to restrictions in lateral movement. If the combined metric $F = min(STN, BTN)$ was included, then the overall score for F would have been equal to or lower than the BTN score as $F \leq BTN$. The BTN is designed with the assumption that the lead vehicle acceleration remains constant. This assumption can make the BTN good at assessing whether a situation is dangerous at the time of it being calculated. Especially, this works well for evaluating π_{host} as the lane change might lead to a situation where collision is unavoidable. On the other hand, in the evaluation of π_{ado} it might be much less effective, especially if the BTN is calculated for V_{host} as in Section 4.1.2. This is what leads to higher values of BTN for higher values of ΔT_{commit_crash} in Figure 4.6 in comparison to Figure 4.2.

Instead, the performance of π_{ado} could be judged by the average reward for all episodes. In

this project the main contributor to achieving a high reward was to find a scenario where a collision occur quickly after a lane change was performed by V_{host}. However, the reward function can easily be altered depending on what the experiment wants to accomplish, and the reward function will reflect how well the system performed in that regard.

6

Conclusion

In this thesis, we investigated the possibilities and limitations of using reinforcement learning in falsification of an autonomous vehicle performing lane changes. The goal was to find critical scenarios where the decision-making of the autonomous vehicle may lead to a violation of safety specifications. The specification in this work was defined to be the avoidance of collisions.

To find falsifying scenarios, we proposed a framework for falsification of autonomous driving using Double Deep Q-Network. Firstly, we have shown that the framework is successful in finding critical lane change scenarios by altering the behavior of one adversarial vehicle. Secondly, we have shown that the framework can be extended to train two agents in a multi-agent system, which increased the number of unique scenarios to be found. Lastly, by reusing the policy for the adversarial vehicle obtained in the first experiment, we have shown that lane change decision-making can be improved in regard to safety. This, while still being efficient at performing lane changes.

With that said, the possibility of using reinforcement learning in falsification of an autonomous lane change is demonstrated with successful results. Generally, reinforcement learning algorithms are prone to be sensitive to hyper-parameter. Finding the optimal parameters can be an extremely time-consuming task. In addition to this, defining a reward function that reflects the wanted behavior is a complex task.

Finally, the results presented in this thesis are not optimized to visualize the optimal solution and should be interpreted as a proof of concept. In future work, three main modifications to the framework are proposed. First, research on how different RL-algorithms and hyper-parameters would affect the results is proposed. Secondly, it is of interest to implement MARL to introduce cooperation between multiple agents. Thirdly, investigate if the decision-making of the host vehicle can be further improved by introducing an iterative process to retrain ado and host vehicle.

Bibliography

[1] World Health Organization. Global status report on road safety 2018. Accessed: 2022-05-31. [Online]. Available: https://www.who.int/publications/i/item/9789241565684

[2] J. M. Anderson, K. Nidhi, K. D. Stanley, P. Sorensen, C. Samaras, and O. A. Oluwatola, *Autonomous vehicle technology: A guide for policymakers.* Rand Corporation, 2014.

[3] I. Yaqoob, L. U. Khan, S. M. A. Kazmi, M. Imran, N. Guizani, and C. S. Hong, "Autonomous driving cars in smart cities: Recent advances, requirements, and challenges," *IEEE Network*, vol. 34, no. 1, pp. 174–181, 2020.

[4] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," *CoRR*, vol. abs/1708.06374, 2017. [Online]. Available: http://arxiv.org/abs/1708.06374

[5] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0965856416302129

[6] H. Abbas, G. Fainekos, S. Sankaranarayanan, F. Ivančić, and A. Gupta, "Probabilistic temporal logic falsification of cyber-physical systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 2s, pp. 1–30, 2013.

[7] J. Lidén Eddeland, *Falsification of signal-based specifications for cyber-physical systems : with applications from the automotive domain.* Department of Electrical Engineering, Chalmers University of Technology, 2019. [Online]. Available: https://research.chalmers.se/publication/514410

[8] Y. Yamagata, S. Liu, T. Akazaki, Y. Duan, and J. Hao, "Falsification of cyber-physical systems using deep reinforcement learning," *IEEE Transactions on Software Engineering*, vol. 47, no. 12, pp. 2823–2840, 2021.

[9] D. Liu, M. Brännstrom, A. Backhouse, and L. Svensson, "Learning faster to perform autonomous lane changes by constructing maneuvers from shielded semantic actions," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019, pp. 1838–1844.

[10] H. Abbas and G. Fainekos, "Convergence proofs for simulated annealing falsification

of safety properties," in *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2012, pp. 1594–1601.

[11] S. Ulbrich, T. Menzel, A. Reschka, F. Schuldt, and M. Maurer, "Defining and substantiating the terms scene, situation, and scenario for automated driving," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, 2015, pp. 982–988.

[12] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *CoRR*, vol. abs/1712.01815, 2017. [Online]. Available: http://arxiv.org/abs/1712.01815

[13] M. van Otterlo and M. Wiering, *Reinforcement Learning and Markov Decision Processes*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 3–42. [Online]. Available: https://doi.org/10.1007/978-3-642-27645-3_1

[14] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. Accessed: 2022-02-07. [Online]. Available: http://incompleteideas.net/sutton/book/the-book-2nd.html

[15] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[16] S. Skansi, *Introduction to Deep Learning: from logical calculus to artificial intelligence*. Springer, 2018, pp. 8–12.

[17] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," *Journal of Machine Learning Research*, vol. 15, 01 2010.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[19] L. Rosasco, E. De Vito, A. Caponnetto, M. Piana, and A. Verri, "Are loss functions all the same?" *Neural computation*, vol. 16, no. 5, pp. 1063–1076, 2004.

[20] P. J. Huber, "Robust Estimation of a Location Parameter," *The Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73 – 101, 1964. [Online]. Available: https://doi.org/10.1214/aoms/1177703732

[21] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016. [Online]. Available: http://arxiv.org/abs/1609.04747

[22] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[23] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[25] J. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with

function approximationtechnical," *Rep. LIDS-P-2322). Lab. Inf. Decis. Syst. Massachusetts Inst. Technol. Tech. Rep*, 1996.

[26] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[27] A. Corso, P. Du, K. R. Driggs-Campbell, and M. J. Kochenderfer, "Adaptive stress testing with reward augmentation for autonomous vehicle validation," *CoRR*, vol. abs/1908.01046, 2019. [Online]. Available: http://arxiv.org/abs/1908.01046

[28] X. Qin, N. Aréchiga, A. Best, and J. Deshmukh, "Automatic testing with reusable adversarial agents," 2019. [Online]. Available: https://arxiv.org/abs/1910.13645

[29] K. Kato, F. Ishikawa, and S. Honiden, "Falsification of cyber-physical systems with reinforcement learning," in *2018 IEEE Workshop on Monitoring and Testing of Cyber-Physical Systems (MT-CPS)*, 2018, pp. 5–6.

[30] A. Donzé, "Breach: A toolbox for verification and parameter synthesis of hybrid systems," in *In Computer-Aided Verification*, 2010, pp. 167–170.

[31] C. E. Tuncali, T. P. Pavlic, and G. Fainekos, "Utilizing s-taliro as an automatic test generation framework for autonomous vehicles," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, 2016, pp. 1470–1475.

[32] OpenAI, :, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with large scale deep reinforcement learning," 2019. [Online]. Available: https://arxiv.org/abs/1912.06680

[33] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[34] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015. [Online]. Available: http://arxiv.org/abs/1502.05477

[35] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *CoRR*, vol. abs/1801.01290, 2018. [Online]. Available: http://arxiv.org/abs/1801.01290

[36] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[37] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[38] G. H. Engelsmann, J. Ekmark, L. Tellis, M. N. Tarabishy, G. M. Joh, R. A. Trombley, Jr., and R. E. Williams, "Threat level identification and quantifying system," U.S. Patent 7 034 668B2, Apr, 2006. [Online]. Available: https://patents.google.com/patent/US7034668B2/en

[39] Z. Ramezani, N. Smallbone, M. Fabian, and K. Åkesson, "Evaluating two semantics for falsification using an autonomous driving example," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, vol. 1, 2019, pp. 386–391.

[40] S. Moon and K. Yi, "Human driving data-based design of a vehicle adaptive cruise control algorithm," *Vehicle System Dynamics*, vol. 46, no. 8, pp. 661–690, 2008.

[41] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[42] R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel, "A survey of generalisation in deep reinforcement learning," *CoRR*, vol. abs/2111.09794, 2021. [Online]. Available: https://arxiv.org/abs/2111.09794

[43] L. Buşoniu, R. Babuška, and B. D. Schutter, "Multi-agent reinforcement learning: An overview," *Innovations in multi-agent systems and applications-1*, pp. 183–221, 2010.

[44] M. Brannstrom, J. Sjoberg, and E. Coelingh, "A situation and threat assessment algorithm for a rear-end collision avoidance system," in *2008 IEEE Intelligent Vehicles Symposium*, 2008, pp. 102–107.

[45] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *International conference on machine learning*. PMLR, 2016, pp. 2829–2838.

[46] J. Nilsson, J. Fredriksson, and A. C. Ödblom, "Verification of collision avoidance systems using reachability analysis," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 10 676–10 681, 2014, 19th IFAC World Congress. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667016433100

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY