
Technische Universiteit Delft

# Safe Reinforcement Learning by Shielding for Autonomous Vehicles

Job Zoon

TU Delft

# Safe Reinforcement Learning by Shielding for Autonomous Vehicles

by

Job Zoon

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday June 28, 2021 at 14:00.

| | | |
|---|---|---|
| Student number: | 4393899 | |
| Project duration: | September 14, 2020 – June 28, 2021 | |
| Thesis committee: | Dr. M. T. J. Spaan | TU Delft, supervisor |
| | Dr. F. A. Oliehoek | TU Delft |
| | Dr. ir. E. M. P. Walraven | TNO |
| With help from: | T. D. Simão, MSc | TU Delft |

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

# Abstract

In the past few years, there has been much research in the field of Autonomous Vehicles (AV). If AVs are implemented in our daily lives, this could have many advantages. Before this can happen, safe driver models need to be designed which control the AVs. One technique that is suitable to create these models is Reinforcement Learning (RL). A problem here is that an RL agent usually needs to execute random actions during training, which is unsafe when driving an AV. Two shields are proposed to solve this problem: a Safety Checking Shield (SCS) and a Safe Initial Policy Shield (SIPS). The SCS checks whether an action is safe by predicting the future state after taking that action and checking whether that future state is safe. The SIPS checks whether an action is safe by comparing it to a safe action from a Safe Initial Policy. Based on the safety of the current state and this action, a safe range of actions is created in which the chosen action must fall. Furthermore, two shield-based learning techniques are proposed which are part of the RL algorithm and allow the agent to learn to avoid proposing actions that would be overruled by a shield. For the first method, experiences are fabricated, and for the second method, an alternative loss function is adopted. In the CARLA driving simulator, two scenarios were created to test the systems. In the first scenario, the agent needs to learn to drive straight, and in the second scenario, it needs to learn to not hit other vehicles on a straight road. The two shields are built around a Double Deep Q-Network (DDQN) and compared to it. It is shown that both shielding systems have zero collisions during training and execution, while having a similar or even better performance in terms of efficiency in comparison to the baseline DDQN. Furthermore, it is shown that both shield-based learning techniques effectively enable the agent to learn to not propose unsafe actions.

# Preface

Before you lies my thesis about Safe Reinforcement Learning by Shielding for Autonomous Vehicles. The goal of this thesis is to create a system that ensures safety when training a Reinforcement Learning agent to drive a car. Two types of shields are created and tested for this purpose, of which one is based on existing techniques and the other based on novel ideas. It is shown in a simulator that both of these techniques ensure safety in two traffic scenarios. They perform similar in terms of efficiency to the baseline Double Deep Q-network without shielding.

This thesis project was conducted at the Algorithmics group of the Delft University of Technology as the final part of my Master of Science Degree in Computer Science. For nine months, I worked at the Sustainable Urban Mobility and Safety department of TNO, where I did this research. Unfortunately, I have only visited the office twice during this period because of lockdowns due to COVID-19. All of the work, from reading initial literature to designing the algorithms to writing this thesis, has been done behind my desk in my student room. I am proud that I was able to find the motivation to work on this project for 40 hours each week for nine months while being at home.

I did not do this project all by myself though, I had help from many people which I would like to thank here. I had my first meeting with Matthijs Spaan over a year before starting the thesis project, in which he agreed to be my supervisor and came up with the idea to conduct this thesis at TNO. During the project, we had many meetings in which he always had useful feedback and interesting ideas about my work. Erwin Walraven was my supervisor at TNO, where he was always available for questions or to help with problems. I really enjoyed our weekly meetings. Thiago Dias Simão helped me a lot as well during this project and often came up with relevant literature that I had not found yet. Both Erwin and Thiago extensively reviewed this thesis which was very useful. Frans Oliehoek agreed to take part in my thesis committee and take time to read this thesis, which I am thankful for.

I would also like to thank my parents and sister for supporting me throughout my studies. Finally, I would like to thank Renée and my friends from the DSB for making the past seven years in Delft a great time.

All the years of work during my studies lead to this thesis of which I am very proud. I hope it is a pleasant read.

*Job Zoon*
*Delft, June 2021*

# Contents

# 1

# Introduction

In 2015, the Netherlands decided to become the first test country of Autonomous Vehicles (AV) in Europe[1]. The government now allows vehicles without steering wheels to drive on public roads. This rule is implemented to make it possible to do autonomous driving experiments on Dutch roads and it is not hard to see why: large companies like Waymo, GM and Ford are spending billions of dollars in creating the technique of letting cars drive by themselves [36]. Dutch research institute TNO is interested in these developments as well and has scientists working on the subject of AVs. This thesis is carried out at the Sustainable Urban Mobility and Safety department of TNO.

There is a reason why these companies have an interest in the development of AVs: the introduction of AVs could completely change the way we think about traffic. Imagine a world where you do not have to own a car, where most parking spots in city centres can be transformed to sidewalks, where the number of car accidents is ten times less than today and where you are driven to your destination by only pressing a few buttons on your smartphone. Right now, we are still way off that scenario, but if techniques can be developed which guarantee safety of AVs, the sketched scenario is not so far away at all. One of the techniques that can be used to let AVs learn to drive safe is Reinforcement Learning (RL). This thesis will investigate RL techniques that could contribute to creating safe AVs.

## 1.1. Autonomous Vehicles

An AV controls a part or all of the functions that a human normally controls when driving a vehicle. The most important functions are acceleration and steering, but especially when an AV shares the road with Human-Driven Vehicles (HDV), also other aspects like direction indicators and lights need to be controlled.

Often when people think of AVs, they think of a car without a steering wheel in which the user does not have to do anything in order to get to its destination. In reality, multiple levels of AVs exist [3], which are listed below.

- **Level 0:** No automation: the human drives the vehicle without any help.
- **Level 1:** Driver assistance: one of the car's functions is automated.
- **Level 2:** Partial automation: multiple of the car's functions are automated, but the driver must still pay full attention.
- **Level 3:** Conditional automation: all of the car's functions are automated, but the driver must still pay full attention.

[1] RTL Nieuws: *Nederland eerste testland: zelfrijdende auto's de weg op*

- **Level 4:** High automation: all of the car's functions are automated. Even if the driver does not pay full attention, the car is able to find a safe solution for every potential dangerous situation by for example parking the car. The driver can still intervene.

- **Level 5:** Full automation: the AV drives without the need for a human to ever intervene.

During this research, the focus is on systems that control the important functions of a vehicle, but only in certain scenarios that are tested. These scenarios will be described in Chapter 3. This corresponds to automation level 2. The AV can drive on its own, but the user should be ready to take over control in situations that were not tested in this research. If the AV is in one of the scenarios that were tested, the driver should never have to take control, since the systems created in this thesis should enforce safety in those scenarios.

There is some discussion in the scientific field whether to use the term Autonomous Vehicle or Automated Vehicle. The literature used in this research almost exclusively uses the term Autonomous Vehicle, and therefore it was decided to use that term as well in this thesis. An argument can be made though that only level 5 vehicles are fully autonomous, since all the other levels have human input.

This section will first describe the impact that AVs would have if they are implemented on a large scale. Then, the barriers that need to be overcome before they can be implemented are discussed. After that, the technology behind AVs is shortly explained.

### 1.1.1. Impact

Why are we working on building AVs? Many people enjoy driving their car by themselves and changing our infrastructure and mobility system to use AVs is a huge challenge. This subsection will discuss the advantages of a world with AVs. Research has shown that AVs could have a positive impact on the world we are living in [7, 25, 45]. The benefits of AVs that these papers present are described below.

*Safety*

Since about 90% of traffic accidents are caused by human error, research shows that accidents can be reduced by up to 90% by using AVs [8], although this number is disputed [45], because AVs could pose new safety issues. Still, literature agrees that the introduction of AVs will likely decrease the number of traffic accidents [7, 25, 45].

Yearly, 1.35 million people die worldwide in traffic accidents [1]. It is the leading cause of death of young people [2]. If all cars in the world would be replaced with AVs, many lives could be saved yearly. It is estimated that this could save the world 1.8 trillion dollars [15], which is equivalent to 0.12% of the global GDP.

*Mobility for everyone*

AVs that do not need any interventions from a driver will increase the mobility of people that cannot drive by themselves [7]. Examples of this group are people who are too young to drive or disabled people. Having the possibility of using an AV would give these people independence, can reduce their social isolation and can provide accessibility to essential services.

*Parking*

It is estimated that 31% of the space in central business districts in 41 major cities worldwide is devoted to parking [62]. It does not matter to which city you go, cars on the side of streets are always part of the scenery. When AVs are used, they could drop off their passengers and then drive to a remote parking area outside the city. Also, fewer vehicles are needed when AVs are used since it is easier to share vehicles in that situation. Both of these effects result in fewer parking spots needed

in metropolitan areas [25]. This could open up space for wider sidewalks, more houses or outdoor seating of restaurants and bars. Especially during COVID-19 times, social distancing would be easier if there was more room available on the streets. A reduction in parking spots would result in a more pleasant and healthy environment within city centres.

*Reduced traffic congestion and emissions*
AVs can contribute to a reduction in traffic congestion [7, 25]. 25% of congestion in the United States is caused by traffic accidents [64]. We already showed that the introduction of AVs reduces the number of accidents, so this will also reduce the amount of congestion. Furthermore, features like adaptive cruise control or vehicle-to-vehicle communication can reduce the congestion even more by using the space on the roads more efficiently. This can reduce the amount of congestion by another 8-13%, depending on how well traffic smoothing algorithms are implemented [25]. As a result, the total emissions of traffic might also be reduced, since vehicles will spend less time on the road when there is less congestion.

*Reduced stress and improved productivity*
When people do not have to drive while sitting in a car, additional time is created to work, sleep or relax while travelling. This reduces stress and increases productivity [45]. When the time sitting in a car can be spent doing useful tasks, it matters less to travel further to work. Therefore people can live further from work and cities might become less crowded.

### 1.1.2. Barriers

When reading about the advantages of the implementation of AVs, one might wonder why our roads are not filled with AVs yet. Unfortunately, there are still some barriers [25], which can be categorized into social/ethical and technical problems. While the social and ethical problems are relevant for the adoption of AVs, the technical side is more interesting for this research. Barriers of both categories are described below.

*Social and ethical*
Right now, an AV would cost way too much money for most people [25]. The sensors, communication technology and software needed make AVs too expensive for large scale use. As with all new technologies, the costs will go down over time, especially when the production goes up. Still, there is a cost bump to get over before AVs are available to everyone.

Even if AVs are available to everyone, legislation is needed that allows AVs to drive on public roads. At this time, legislation for autonomous driving differs per country. It can take quite some time before qualitative legislation about important subjects like this is written and accepted, so governments should start working on this soon.

One of the reasons legislation for this subject is complex is the liability when an AV is involved in an accident. Even when AVs are extremely safe and are held to a higher standard than human drivers, some accidents will still occur, possibly with death as result. In most situations early on when only few AVs are on our roads and when these AVs are safer than HDV's, humans will be the cause of accidents with AVs and they will be liable. Therefore AVs should store sensor data to show their contribution to accidents. However, there will be accidents that the AV caused. Who is then responsible for compensation of the victims? The person sitting in the AV, who did not cause the accident, the software developer that made a mistake when developing AV software or the company that the developer is working for? These questions need to be answered before AVs are allowed on a large scale.

Furthermore, situations exist in which an AV has to make a choice between two bad situations. To be prepared, a project called The Moral Machine [9] is working on this problem by gathering

large amounts of data about decisions that people take in difficult situations. An example of such a difficult situation is when an AV needs to choose between hitting three adults and two children walking through a red light and killing them, or steering away and killing the three adults and two children who are sitting in the vehicle. Knowing how an AV must act and who is responsible after an action in such situations is crucial for legislation and acceptance of AVs.

Finally, privacy is an issue in the field of AVs [25], as in all fields where customer's data is used and shared. In this case, travel data like routes, destinations and times of day can be used by companies that create AV systems to generate navigation to avoid congestion, while minimizing the use of energy. Whilst this has some advantages, using personal data is controversial, so legislation should take privacy into account as well.

*Technical*
The potentially most disastrous danger of AVs is a breach of electronic security [25]. Hackers, terrorists or hostile nations could target AVs, letting them cause accidents. Especially if all AVs are connected to each other and act based on each others observations, it could be possible to let all connected AVs crash at the same time. Cyber security companies are working on guaranteeing electronic security for AVs. Although security is crucial for AVs to be implemented on a large scale, this subject is not part of this thesis.

The focus of this research is on a completely different part of the technical challenges: the technology behind driving an AV. It is important that the driving behavior of AVs is as safe as possible, and a lot of research is still missing in this field. There are only a few test AVs available, which makes it hard to tell how AVs influence traffic streams and how safe they are. Most techniques are developed in simulators on simple scenarios. If a technique works well in a simulator, it is not guaranteed that it also works well in real life. Furthermore, new techniques that are still being developed can always be safer than existing techniques. Overall, there is a lot of research that still needs to be done to ensure a safe implementation of AVs in our daily lives.

### 1.1.3. Technology

As stated in the previous section, one of the mayor technical challenges before AVs can be implemented in our daily lives is to create reliable systems to drive the AVs. This section will shortly describe the setup of such systems. An autonomous driving system, the system that controls an AV, can be divided into five components [38]. These components are listed below.

1. **Scene understanding** aims to capture the environment of an AV and translates it to a representation that a computer can use. Examples of technologies that can be used for this are camera's, radar, LIDAR and ultrasonics.

2. **Perception and localization** tries to detect objects and lanes. Different techniques, for example supervised machine learning, can be used.

3. **Scene representation** maps the surroundings of the AV by using the output of the previous step. The mapping is used as input of the AV's decision making process.

4. **Planning and deciding** creates a high level driving policy based on the scene and the AV's destination. The route, path and speed of the AV are planned and optimized.

5. **Control** consists of the actual actions the AV takes, of which the most important two are acceleration and steering. These control actions must execute the plan that was created in the previous step.

This research focuses on mapping component three to component five. It is assumed that the system built in this research receives perfect information about the environment. How the sensors

work and how objects are detected are different fields in the AV research area. Furthermore, the scenarios that are used in this thesis are simple enough that no planning and deciding is necessary.

For all of the five components, many different techniques exist. This research focuses on the technique of RL to map the scene (component three) to safe vehicle control actions (component five). This can be seen as an end-to-end approach, skipping the planning and deciding step. It has been shown that such an end-to-end approach can work quite well [10, 11, 40].

## 1.2. Reinforcement Learning

RL is one of the three main categories in the field of Machine Learning (ML), next to supervised and unsupervised learning. It deals with sequential decision making problems. Basically, an agent is put into an unknown environment and explores this environment by executing actions. Based on the action and the state that the agent is in, the agent receives feedback from the environment in terms of a reward. It can learn a policy in a sequential decision making problem by learning from the rewards it receives after executing actions. The process of RL consists of an agent repeatedly picking actions and gathering rewards, eventually learning good behavior within its environment. A more detailed description will be given in Chapter 2.

RL is used for a wide range of applications [44]. It can be used to learn computers to play games. An example of a successful RL agent is AlphaGo Zero [63], a computer program that learned to play the game Go by using RL without previous knowledge. AlphaGo Zero has beaten AlphaGo, a computer program that was trained by using RL with human expert knowledge and Monte Carlo Tree Search, 100 times in 100 games. Since Go has a gigantic search space and because it is hard to evaluate positions, it was a great achievement of plain RL to be able to perform so well in this game.

Other examples of fields where RL is used successfully are robotics, natural language processing, computer vision and, most relevant for this research, intelligent transportation systems. RL can be used for intelligent traffic lights, but also for AVs, the subject of this thesis. There has been much research in the field of RL for AVs [38]. Different approaches exist: from end-to-end learning where the RL algorithm uses a low-level state space and directly outputs an action, to very specific task learning, for example when an agent only needs to learn steering actions for a lane change. In this research, the focus is on an RL algorithm that takes a high-level representation of the environment as input and outputs the desired throttle and steering actions.

Usually, an RL agent explores the environment by executing random actions. This does not work when an AV is trained in real life, since the AV could cause traffic accidents and the equipment might be damaged when it takes random actions. A subfield of RL called Safe RL deals with problems in which safety of the agent needs to be ensured.

### 1.2.1. Safe Reinforcement Learning

Safe RL is a part of the RL field in which it is important to ensure system performance or to respect safety constraints during training and/or execution [27]. For games, Safe RL is not relevant since it does not matter if an RL agent does not perform well during training or is not consistent enough, as long as the final learned policy performs well. For the field of AVs, Safe RL is more relevant, because there are some safety constraints that need to be respected when training AVs.

There are two approaches to Safe RL: changing the optimization criterion or changing the exploration process [27]. This research focuses on changing the exploration process. Specifically, a method called shielding [6] is analyzed and used to ensure that safety constraints are enforced during the training process.

When using shielding, a shield checks whether the actions proposed by the agent are safe or not. If they are not safe, they are overruled with a safe action. This thesis investigates how shields can be applied in the field of AVs. To do this, one shield that is based on literature is tested on AV scenarios, and one novel type of shield is proposed and tested on the same scenarios. These shields are based

on two models. The first model can determine whether a given action is safe or not based on the state. The second model can propose a safe action for every state in which a safe action is available.

Usually when shielding is used, the shield is employed both during training and execution of the policy that is learned with RL. In this thesis, two techniques are investigated which enable the agent to learn what actions not to take based on the actions that the shield overrules. We call this shield-based learning. Shield-based learning allows the shield not to be employed during execution, since the agent should have the knowledge to never propose actions that would trigger the shield anyways.

## 1.3. Problem Statement

Section 1.1.1 has shown that the implementation of AVs in our daily lives can have a huge positive impact. One of the techniques that can be used to control AVs is RL. RL is a suitable technique because of the structure of the problem of letting AVs drive: this is a sequential decision making problem where states need to be mapped to actions, which is exactly the type of problem that RL can solve. It is not trivial to model the problem of controlling AVs as a Markov Decision Process (which will be described in Section 2.1.1). This makes regular planning approaches less suitable. Furthermore, because of the complexity of the problem, it is hard to create a rule-based model manually. RL does not have these issues. Unfortunately, there is a problem when using RL for AVs.

To learn a policy with RL, training is needed. During the training phase, the AV drives around in an environment and its experiences are saved. To learn the best action for all possible states, it is important that many different experiences are sampled. Usually this is done through random exploration: the agent performing random actions in different states to gather many different experiences. In the field of AVs, this is a problem because it is not safe to let a vehicle in the real world perform random actions. This could cause unsafe situations or damage the equipment.

Some effort has been put into creating models which can determine whether AV control actions are safe in the current situation [29] and models that can propose a safe action for every situation, but are not necessarily optimal in terms of efficiency [68]. These kinds of models do offer insightful knowledge about safety when building an AV system, but they are not sufficient to let a car drive as efficient as possible. This thesis aims to use these types of models in a shielding based approach to safely train AVs using RL. Since no models exist for the exact scenarios that are used in this thesis, both models need to be designed and implemented manually. The first model, which can judge action safety, is used for an existing type of shielding, while the second model, that can propose safe actions, is used for a novel type of shielding.

Summarized, the aim is to solve the safety issue by using two shielding approaches. These approaches are built by using two different models. When employing these shields, it should be possible to train an AV with RL without it having any collisions. In a simulator, it will be tested whether the shields ensure safety and how this impacts the RL process. If shields can ensure safety, this would solve the problem of not being able to safely train AVs using RL. The research questions that formalize the problem can be found in Section 1.4.

## 1.4. Research Questions

The objective of this research is to use two models for two shields to guarantee safety during the training of AVs using RL. The following main research question follows from this objective:

**Research Question.** *Given a model that can determine action safety and a model that can propose safe actions for Autonomous Vehicles, how can shielding be used to learn a policy with Reinforcement Learning in such a way that safety is guaranteed during training and execution of the policy?*

To answer this question, five subquestions are formulated. In this section, all of the subquestions are stated and discussed.

**Subquestion 1.** *What is a suitable baseline Reinforcement Learning system to train an agent to drive autonomously?*

The basis of this research is an RL algorithm that trains an agent to drive. Only when this algorithm is implemented, it is possible to implement shields on top of it and check their effects. The algorithm should be suitable for the field of AVs and for the shielding approaches. The goal is not to create the best possible RL system for AVs, but the baseline system should perform well.

**Subquestion 2.** *How can a model that can determine safety for actions be integrated with an existing shielding approach to ensure safety during training?*

The first shield is based on literature and uses a model which can determine safety for actions. This model needs to be designed for the scenarios in this thesis and it needs to be integrated with an existing shielding approach. During this design process, the guarantee of safety must be kept in mind. The second subquestion relates to this process.

**Subquestion 3.** *How can a model that can propose safe actions for every situation be used as a novel type of shield which ensures safety during training?*

The second shield uses an existing model which can propose a safe action for every situation in which a safe action is available. During the design of this novel type of shield, it must be ensured that safety is guaranteed. The third subquestion relates to this second type of shield.

**Subquestion 4.** *What techniques can be used to enable the agent to learn from overruled actions by the shields in such a way that execution of the final policy is safe even if the shields are not employed anymore?*

We do not only care about safety during training, but also about safety during execution of the final policy. While the training of the agent should be kept safe by the shields, it could be beneficial to not have a shield employed anymore during execution. Two techniques are proposed that allow the agent to learn from overruled actions by the shield.

**Subquestion 5.** *How do the baseline RL algorithm and the two types of shields compare in terms of safety and efficiency during training and execution?*

Finally, we are not only interested in creating shields, but also in how well they perform and whether they solve the problem that is investigated in this thesis. They should be compared against each other and against the baseline RL algorithm. Both safety and efficiency need to be tested. The comparison shows how the shields influence the RL process and whether they can contribute to safe training for AVs.

## 1.5. Contributions

Since there has not been much research into Safe RL for AVs, the first contribution of this thesis is to test shields in the field of AVs. In a recent survey about Deep RL and Deep Imitation Learning for AVs, only 5 out of the 64 papers investigated consider the safety aspect of AVs [76]. Shields have barely been tested on AVs yet. This thesis will show the effect of shields on the AV training process. This contributes to the range of Safe RL techniques that have been tested on AVs and it extends the range of fields in which shields have been tested.

Furthermore, a new type of shield is proposed in this thesis: a shield based on a model that can propose safe actions for every situation where a safe action is available. This type of shield can not only be used for AVs, but also in other safety critical applications. This thesis shows how a safe initial policy can be used as a shield for an RL algorithm. The type of method that is proposed in this thesis has not been found in existing literature, so it is a contribution to the field of shielding. If it performs well, it could be tested in other fields as well.

Other than that, a contribution is made to the learning process when a shield is employed. Usually, when an agent is trained with a shield employed, the shield keeps being employed during execution of the final policy [6], so the agent does not have to learn to avoid proposing actions that would be overruled by a shield. This research tests two different methods that enable the agent to learn from those overruled actions, one where fabricated experiences are used and one where an alternative loss function is used. Both of these methods are tested to check their influence. The alternative loss function has not been found in existing literature.

Finally, if the methods perform well, this would be a confirmation that shields as a technique are useful for safety critical applications. Since AVs are such an application, this could be a small step towards being able to train AVs in real life using RL. Right now, this is not possible due to safety constraints. If it would be possible to train AVs in real life, this could potentially result in better performing AVs, which would contribute to bringing the large scale implementation of AVs a bit closer to our daily lives. As described earlier in this chapter, this could have a huge positive impact on our society.

## 1.6. Outline

After this introduction chapter, Chapter 2 discusses background information about RL, Q-Learning, Deep Q-learning, Safe RL, shielding and related work to this thesis. Safe RL research for AVs is classified as related work. After handling all the background information, the problem of this thesis is formalized in Chapter 3. The evaluation criteria are defined in this chapter, as well as the aim and scope of the thesis. The model within which RL can be applied will be formalized in this chapter as well. Then, Chapter 4 contains all information about the two proposed shields: the Safety Checking Shield and the Safe Initial Policy Shield. The two shield-based learning techniques, fabricating experiences and using an alternative loss function, are described in this chapter as well. After all techniques have been explained, Chapter 5 describes the setup of the experiments that have been conducted for this research. The choice of simulator, the RL algorithm and experimental setup are discussed. Then, Chapter 6 lists the expectations about the experiments and then shows and analyzes the results. Finally, Chapter 7 contains a conclusion and some future work recommendations.

# 2

# Background

This chapter contains background information about the research in this thesis. First, the general concept of Reinforcement Learning (RL) [70] is explained, after which specifically (Deep) Q-learning is described, which is the RL technique that is used in this research. After that, the concept of Safe RL is introduced, specifically shielding, which is the technique that is evaluated in this thesis. The final section of this chapter discusses some related work in the field of Safe RL for Autonomous Vehicles (AV).

## 2.1. Reinforcement Learning

Machine Learning is the field of letting a computer (the machine) use computational methods to learn a task [52]. This task can be to make accurate predictions or to improve the machine's behavior in an environment. The learning process is based on electronic data, which can be in the form of data sets labeled by humans, unlabeled data sets or data obtained by interacting with an environment. First, the machine is trained by processing data, which is the learning process. After training, it should be able to perform better on a task than before.

The three main categories in Machine Learning are Supervised Learning, Unsupervised Learning and Reinforcement Learning. RL is used to let a machine, which we call the agent, learn how to behave in an environment [70]. That is exactly what needs to be learned in the field of AVs: AVs should know how to behave on the road. Therefore this technique is applicable in the field of AVs. This section describes how RL works. First, Markov Decision Processes are explained, which can be seen as the framework in which RL can be applied. Then, the concept of a policy and the training process are discussed.

### 2.1.1. Markov Decision Process

RL problems can be formalized by the Markov Decision Process (MDP) framework [57]. A schematic representation of this framework is shown in Figure 2.1. All aspects of an MDP [70], which are shown in this figure, are described below. An agent, in our case an AV, tries to learn to perform well on a task within this framework.

*State*

A state is a representation of the environment of the RL agent. States are needed to provide the agent with information about its surroundings. The state space can be either discrete or continuous.

In a discrete setting, there is a finite set of states $S = \{s_1, s_2, ..., s_n\}$ of the environment. An example of a problem with a discrete state space is a grid world in which the agent needs to learn to walk to a destination. In that case, the location of the agent could be used as the representation for the state. There is a finite number of locations in a grid world, so this would make the state space discrete.

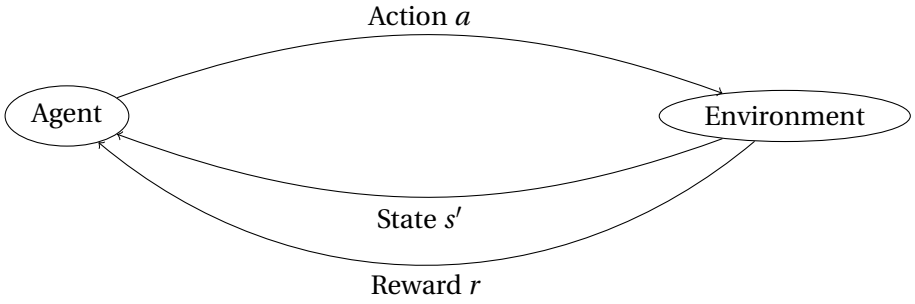



Figure 2.1: Schematic representation of a Markov Decision Process. An agent sends an action $a$ to the environment, which responds with a reward $r$ and transitions to a new state $s'$.

In a continuous setting, the state space consists of a number of variables $(\sigma_1, \sigma_2, ..., \sigma_n)$, in which the state variables $\sigma$ can take on an infinite amount of values, although they might be restricted to a certain range. The environment in the field of AVs is so complex that a continuous setting is more suitable than a discrete setting. Possible variables that could be included are the locations and velocities of the AV and of other vehicles. It would be infeasible to model this in a discrete state space.

*Action*
Actions are the things that the agent can do. They are the only part of an MDP that the agent directly controls. Actions have an effect on the environment and therefore they partially determine the state of the environment. Again, they can be discrete or continuous.

In a discrete setting, there is a finite set of actions $A = \{a_1, a_2, ..., a_n\}$ from which the agent can choose one at every step. For example, when the agent needs to learn to walk to a destination in a grid world, the actions could be to walk one step up, down, left or right. In that case there would be four actions that the agent can choose from, of which one is executed at each step.

In a continuous setting, the action space consists of a number of variables, for which the agent needs to pick values. In an AV context, the action space could consist of an acceleration and a steering variable, for which the agent needs to pick two values at each step.

*Transition function*
As stated before, an action influences the state. Specifically, by applying an action $a$ in a state $s$, the environment makes a transition to a new state $s'$. The transition function $T$ maps $s$ and $a$ to probabilities of ending up in a new state $s'$: $T(s, a, s') \rightarrow [0, 1]$. A system is only truly Markovian if $T(s, a, s') = P(s'|s, a)$ holds: if the next state is only dependent on the previous state and action, and not on actions that were taken or states that have been visited before the previous step. In this research, it is assumed that the transition function is deterministic. This means that $T(s, a, s')$ is 1 for one future state $s'$ and 0 for all other possible future states $s'$.

*Reward*
Finally, the agent receives rewards for being in a state or executing an action in a state, depending on the system. In this thesis, the reward is only based on the state $s'$ that the agent ends up in after taking an action. The reward function $R$ maps every possible state to a reward $R : S \rightarrow \mathbb{R}$. Rewards are crucial for the learning part of RL algorithms. Based on the magnitude of the reward, the agent can learn which actions are good in which states. The goal of the agent is to maximize its return, which is the reward accumulated over time.

Together, the state space, action space, transition function and reward function make up the MDP. In RL, the agent can at every timestep observe the state $s$, execute an action $a$, after which it gets a

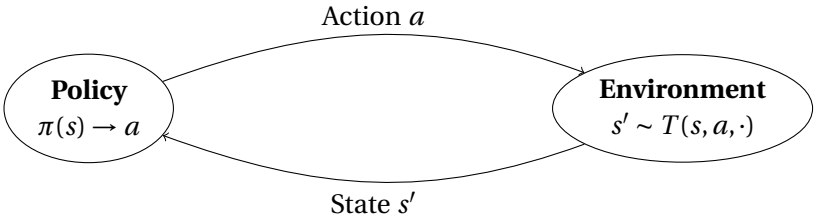



Figure 2.2: Schematic representation of a policy interacting with an environment. At every step, the policy $\pi$ decides which action to take based on the state. The transition function $T$ of the environment then produces a new state $s'$ based on the current state $s$ and chosen action $a$.

reward $r$ and transitions to the next state $s'$ based on the transition function $T$, which is unknown to the agent. These variables form a $(s, a, r, s')$ tuple which is called an experience. The agent's learning process is based on experiences.

The learning process usually consists of multiple episodes. An episode is a sequence of states, actions and reward which ends when a termination criterion is met. Examples of termination criteria are reaching a goal state and going over the time limit.

### 2.1.2. Policy

Within an MDP, a policy $\pi$ is a mapping from every possible state to an action $\pi : S \rightarrow A$. In other words, a policy is a strategy in an environment. When an agent follows a deterministic policy, it will take action $a = \pi(s)$ for the state it encounters at every timestep. This process is shown in Figure 2.2.

The goal of RL is to learn a good policy for an environment. A good policy is a strategy that maximizes the optimality criterion. In this thesis, a discounted, infinite horizon optimality criterion is used, which is shown in Equation 2.1. This equation shows the expected discounted return. For every step $e$, a discount factor $\gamma^e$ is multiplied with the reward from that step $r_e$, where $0 \leq \gamma < 1$. Because of the $\gamma^e$ factor, the expected rewards closer in the future are more important than the expected rewards further in the future. If $\gamma$ is set to a low value, the future rewards do not matter much. If it is set to a high value, the future rewards are almost as important as the current reward.

$$E\left[\sum_{e=1}^{\infty} \gamma^e r_e\right] \tag{2.1}$$

In the context of AVs, a good policy is a policy that drives safely and efficiently for every possible traffic situation. A policy can be learned through training an agent, as described in the next subsection.

### 2.1.3. Training

RL is the process of learning a policy through training. During training, the agent is put in an environment and will sample experiences $(s, a, r, s')$ by executing one action per step. It can learn a policy based on these experiences. The training process is shown in Figure 2.3. Usually many episodes, which are sequences of steps, are simulated to train an agent.

An often used strategy to pick actions during the training process is called $\epsilon$-greedy, which can be seen as the policy that is used during training. The $\epsilon$-greedy strategy is a policy that is used to sample an adequate set of experiences to learn a good final policy. $\epsilon$-greedy balances the concepts of exploration and exploitation.

Exploration aims to find a wide range of experiences. This is necessary since a good policy can only be learned when enough different experiences are found. When the agent chooses for exploration at a timestep, it usually randomly selects an action. When for every state a lot of random actions have been executed, it is likely that most of the environment's search space has been expe-

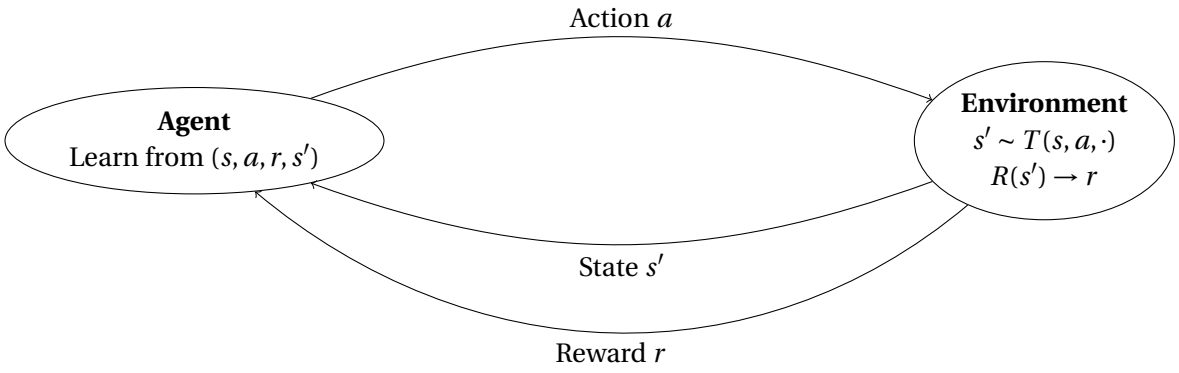



Figure 2.3: Schematic representation of an agent interacting with an environment during training. At every step, the agent decides which action to take based on the state. The transition function $T$ of the environment then produces a new state $s'$ based on the current state $s$ and chosen action $a$. The reward function $R$ produces a reward $r$ based on $s'$. The agent updates its policy based on the experience $(s, a, r, s')$.

rienced during training. This ensures that the agent has experienced the actions that belong to a good policy and therefore can learn a good policy on its own.

Different from exploration, exploitation aims to exploit the knowledge that has already been learned by the agent. During an exploitation step, the agent picks the estimated best action instead of a random action. This should result in sampling experiences with higher rewards and therefore learning a better policy. Exploration is the broad search of many different experiences to see what regions of the search space are promising, while exploitation is used to find a good policy within those regions.

At the start of training, it is beneficial to mainly explore the search space, because the agent has not learned anything yet. Later on, exploitation becomes more useful. In this thesis, an $\epsilon$-decreasing strategy is used, in which the choice between executing a random action (exploration) or the estimated best action (exploitation) is based on parameter $\epsilon$. $\epsilon$ starts at 1 and decreases slowly during training. If a randomly drawn number from a uniform distribution between 0 and 1 is lower than $\epsilon$, exploration is chosen. When it is higher, exploitation is chosen. So $\epsilon$ balances exploration and exploitation during the whole training process, starting with a higher exploration and ending with a higher exploitation.

During the training process, an RL algorithm is used to learn a policy from the gathered experiences. There are many learning algorithms and each of them has a different way of learning a policy from experiences [70].

## 2.2. Q-learning

One of the learning algorithms that can be used to train an agent is Q-learning [72]. Q-learning is a dynamic programming type of method, which incrementally improves its evaluations of how well actions perform in states. It is a type of Temporal Difference learning, which is model-free learning that learns by bootstrapping from estimates of the value function [65]. It is shown that Q-learning converges to the optimal action values when for every state all actions are sampled enough times [71].

Since Q-learning was first described in 1989 [72], many versions of it have been proposed. This section will first describe how regular Q-learning works, after which Deep Q-Networks (DQN) and Double Deep Q-Networks (DDQN) are explained. DDQN is the specific method that is used in this research.

**Algorithm 1** Q-learning

Initialize $Q$
**for** episode $k \in \{1, ..., K\}$ **do**
  observe $s$
  **for** step $e \in \{1, ..., E\}$ **do**
    **if** $u \sim U(0,1) < \epsilon$ **then**
      $a \leftarrow \text{rand}(A)$
    **else**
      $a \leftarrow \text{argmax}_a Q(s,a)$
    $s' \sim T(s,a,\cdot)$
    $r \leftarrow R(s')$
    $Q'(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \max_a Q(s',a) - Q(s,a)\right)$
    $s \leftarrow s'$
  decrease $\epsilon$

### 2.2.1. Regular Q-learning

As for all RL algorithms, Q-learning aims to learn a policy from experiences. It specifically aims to estimate Q-values for every action in every state. Q-values are a way to value actions in states: the higher the Q-value, the better the action is in that state. For regular Q-learning, both the state and action space need to be discrete. This means that the Q-values for every state-action pair can be saved in a table of size $|S| \cdot |A|$, the size of the state space times the size of the action space. This table is called a Q-table.

The Q-values in the Q-table can be initialized randomly or all be set to 0. During training, whenever an experience $(s, a, r, s')$ is fed to the Q-learning algorithm, the Q-value belonging to $(s, a)$ is updated using the Bellman update rule, which is shown in Equation 2.2.

$$Q'(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right) \tag{2.2}$$

In Equation 2.2, $Q'(s,a)$ is the new Q-value for a state-action pair. The updated value is calculated by the right side of the equation. $Q(s,a)$ is the current Q-value for the state-action pair. $\alpha$ is the learning rate, which determines how strongly Q-values are updated at every timestep. $r$ is the reward that was received after performing action $a$ in state $s$. $\gamma$ is the discount factor, which determines the importance of the future rewards. $\gamma$ should be between 0 and 1. At 0, the future rewards do not matter at all to the agent. When it is set close to 1, every future reward is almost as important as the current reward. $\max_a Q(s',a)$ is the maximum Q-value over every action in the new state $s'$, which is reached after taking action $a$ in state $s$.

The Bellman equation is used to calculate a new Q-value for a state-action pair by slightly updating the old Q-value, based on the reward $r$ and the estimated value of the new state $s'$. During training, a lot of experiences $(s, a, r, s')$ are fed to this Bellman equation, which then updates the according Q-values in the Q-table. As stated before, if this Q-table is fed enough diverse experiences, it will converge to perfect Q-values. The training process of regular Q-learning is shown in Algorithm 1.

The resulting policy is quite straightforward: for the current state, the Q-values of all actions are checked in the Q-table. The action with the highest Q-value is executed. This greedy policy is also used during the exploitation part of training.

### 2.2.2. Deep Q-learning

Regular Q-learning only works for discrete state and action spaces. This is fine for simple MDP's, but for most real life applications this is not sufficient. One way to solve this is by discretizing a

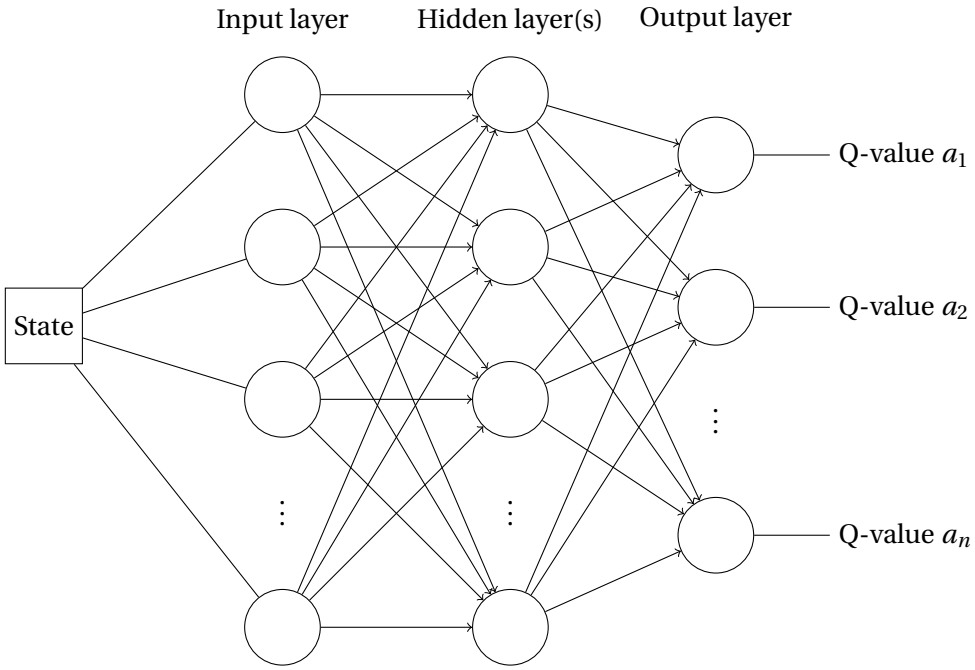



Figure 2.4: Schematic representation of a Deep Q-Network. The continuous state variables are used as the input of the neural network, which outputs estimated Q-values per action.

continuous state or action space, but this might be infeasible for complex spaces. If it is possible to discretize a complex space, it can result in a huge number of states, which can cause a decrease in performance. This is related to the curse of dimensionality: the extreme growth in complexity of problems as the number of variables increases.

For complex state spaces there is a solution: a variant of Q-learning exists that uses a continuous state space and a discrete action space, which is called Deep Q-learning [50]. In Deep Q-learning, the Q-table from regular Q-learning is replaced by a Deep Neural Network (DNN), called a Deep Q-network (DQN). A DNN is a function approximator which in this case tries to approximate the Q-table from regular Q-learning. Figure 2.4 shows an example of a DQN. Instead of having a discrete state space, the state can now be represented as a matrix of continuous variables. The state variables are the direct input of the DQN. The output of the DQN are estimated Q-values for every action. Based on the Bellman equation (see Equation 2.2), the difference between the output values and the calculated Q-values can be determined. The DQN is then updated to make the state to Q-value mapping more accurate. Thus, instead of calculating a Q-value for every state-action pair in a tabular form, a DNN is used to estimate the Q-value of every action based on the state. The pseudocode of a DQN is shown in Algorithm 2.

During training of a DQN, a replay memory is used. This memory stores experiences that the agent has collected so far. Only when the replay memory has reached a certain size, the DQN will start training on the experiences. At every training step, a random batch of the experiences is sampled. This prevents the network from overfitting on the first few experiences.

Two networks are used when training a DQN: an online network and a target network. The online network is constantly updated during training, while the target network is only updated once in a while, copying the weights from the online network to the target network. This is needed for stability during training. As shown in Equation 2.2 in the $\max_a Q(s', a)$ term, the Q-values that the DQN needs to learn are partially dependent on the Q-values of the DQN itself. If the DQN needs to learn a function based on outputs that keep changing quickly, it is hard to get stable results. Therefore a target network is introduced, which keeps stable for a number of episodes. $\max_a Q(s', a)$

**Algorithm 2** Deep Q-Network

1: Initialize replay memory $D$
2: Initialize online network $Q$ with random weights $\theta$
3: Initialize target network $Q_t$ with weights $\theta_t \leftarrow \theta$
4: **for** episode $k \in \{1, ..., K\}$ **do**
5: observe $s$
6: **for** step $e \in \{1, ..., E\}$ **do**
7: **if** $u \sim U(0,1) < \epsilon$ **then**
8: $a \leftarrow \text{rand}(A)$
9: **else**
10: $a \leftarrow \text{argmax}_a Q(s, a; \theta)$
11: $s' \sim T(s, a, \cdot)$
12: $r \leftarrow R(s')$
13: $d \leftarrow e == E$
14: $D \leftarrow D \cup (s, a, r, s', d)$
15: Sample minibatch of $(s_i, a_i, r_i, s_{i+1}, d_i)$ from $D$
16: **if** not $d_i$ **then**
17: $Q$ gradient descent step on $((r_i + \gamma \max_{a'} Q_t(s_{i+1}, a'; \theta_t)) - Q(s_i, a_i; \theta))^2$
18: **else**
19: $Q$ gradient descent step on $(r_i - Q(s_i, a_i; \theta))^2$
20: $s \leftarrow s'$
21: decrease $\epsilon$
22: Every couple of episodes: $Q_t \leftarrow Q$

is calculated by using the target network, so for state $s'$, the highest output Q-value of the target network is used. This means that the online network can learn a function without the outcome of this function changing at every learning step.

Experiences are saved with a fifth variable which indicates whether the experience was the last step of the episode or not. This variable $d$ is set to true if it was the last step of the episode and to false otherwise. It determines how the Q-network is updated. If an experience was the last of the episode, this means that no more steps follow after the step from that experience and therefore no more rewards can be gathered after that step. The Q-value for the state-action pair of that step should then be equal to the reward from that experience without an extra term that represents future rewards. The difference in the way the Q-network is updated based on $d$ is shown in lines 16-19 in Algorithm 2.

### 2.2.3. Double Deep Q-Network

DQN has an issue of overestimating Q-values. A solution is to use a Double Deep Q-Network (DDQN) instead of a DQN [69]. DDQN is the RL technique that is used in this research. It is very similar to DQN with just a slight difference.

The difference lies in line 17 of Algorithm 2 in the $\max_{a'} Q_t(s_{i+1}, a'; \theta_t)$ term. In a DQN, this part is calculated by using Q-values of only the target network $Q_t$. In a DDQN, both the online network $Q$ and the target network $Q_t$ are used. From the online network, the index of the estimated best action $a$ for state $s'$ is determined. The Q-value of that action $a$ for state $s'$ is then obtained from the target network. The online network picks the best action and the corresponding Q-value is obtained from the target network.

## 2.3. Safe Reinforcement Learning

Safe RL is a branch of RL that deals with RL problems in which safety constraints need to be respected during training or execution, or problems in which a reasonable system performance needs to be guaranteed [27]. Training AVs in the real world is such a problem: it is essential that safety constraints are enforced when training AVs using RL, otherwise collisions could occur. Safe RL can be roughly divided into two categories: modifying the optimization criterion and modifying the exploration process [27]. This section describes both of these categories. At the end of the section, the specific method that is used in this thesis, shielding, will be explained in more detail.

### 2.3.1. Optimization criterion

The goal of RL is to find a policy which achieves the maximum performance for some criterion. In regular RL, this criterion is usually to maximize the expected return, which was explained in Section 2.1.2, but it can also be to minimize a cost metric. This is not always suitable for risky tasks, since the return or cost might not consider how risky a policy is. Therefore the first Safe RL approach is to change the optimization criterion to include the concept of risk. Criteria that consider risk can be categorized into four categories [27].

A worst case criterion can be used [31, 55], for which a policy achieves its maximum performance when the return, the accumulated reward over time, is as high as possible in the worst case scenario. This ensures that the policy always has a reasonable performance, since the return is maximized particularly when the agent is in a bad scenario. It also reduces the variability of the policy. The second type of criterion that can be used for Safe RL is the risk-sensitive criterion. In this criterion, the return and a risk factor need to be balanced. The optimal policy does not only yield the highest return, but it also avoids risky situations. The return and risk can be balanced in an exponential function [32] or as a weighted sum [49, 60]. The third type of criterion is a constrained optimization criterion [35, 53]. In this case, the objective is to maximize the return subject to constraints: one or more variables that need to be higher or lower than given bounds. Finally, there are a couple of other optimization criteria that can be used for Safe RL, for example risk metrics from the field of financial engineering [48].

### 2.3.2. Exploration process

The second Safe RL approach is to modify the exploration process. This is the approach that is investigated in this thesis. As stated in Section 2.1.3, usually strategies like $\epsilon$-greedy are used as the policy during training, which randomly explore states and actions. The use of these strategies has two disadvantages: a large amount of time is wasted by exploring irrelevant states and actions, and using these partially random strategies may lead to unsafe situations. The Safe RL techniques to modify the exploration process can be categorized into using risk-directed exploration and incorporating external knowledge [27].

When using risk-directed exploration [30, 41], a risk metric is used which determines the risk of different actions. Actions with a lower estimated risk have a higher probability of being executed. Meanwhile, the regular optimization criterion is used, which is usually maximizing the return. Since this type of exploration is stochastic, it is hard to provide safety guarantees when using this technique.

The other approach is to incorporate external knowledge. This can be done by providing the agent with initial knowledge [23]. This initial knowledge can consists of already known information about the task or experiences that were gathered earlier. It can be used to initialize the agent by teaching it to only explore the regions of the state and action space that the initial knowledge showed to be safe and valuable. A technique that goes a bit further into using initial knowledge aims to derive a policy only based on the initial knowledge [4]. This technique is called Apprenticeship Learning. A set of experiences can be used to learn a policy in an offline manner. This completely

eliminates any chance of unsafety during training, since training does not require any interaction with the environment. It does require enough experiences to train on, which have to be generated in some way. Finally, external knowledge can be used in the form of teacher advice. At every step during training, the agent can ask for help [20] or be granted advice from a teacher whenever the teacher thinks this is necessary [19]. The teacher can help the agent to stay safe during exploration. In this thesis, teacher advice in the form of a shield is implemented and tested. The technique of shielding will be described in Section 2.3.3.

### 2.3.3. Shielding

The Safe RL technique that is used in this thesis is called shielding. It is an approach in which a teacher, in this case a shield, provides advice to the agent whenever the teacher feels this is necessary. Such a teacher can be human or a computer model and it can be implemented in multiple forms [27]. It can give advice to the agent at different times: by interrupting it, during pauses of the agent, or by providing the agent with information that it can read whenever it has time. The advice can come in different forms: for example visually or textually. In some cases the teacher is only able to give advice, in other cases it has the power to overrule actions.

A shield is a kind of teacher that specifically looks at the safety of actions. Its goal is to guarantee safety during training, by only letting the agent perform safe actions. This can be done in two ways [6]. At every timestep, the shield can either pick a set of safe actions from which the agent can choose one, or it can let the agent choose any action and then overrule it if it is unsafe. The first way is called preemptive shielding, the second way post-posed shielding. In both ways, the shield has a specific moment during a training step during which it checks actions, and it has the power to either remove some action options or overrule actions. In this thesis, post-posed shielding is used: overruling actions that are unsafe.

In Figure 2.5, the use of a shield during training is shown. Instead of directly sending an action to the environment, the agent creates a sorted list of all actions based on preference. One by one, the shield checks these actions until it finds a safe one. That action is then sent to the environment. This means that the shield does not change the action if the preferred action of the agent is safe. Only if that action is not safe, it will look through all other actions to find a safe one.

The way in which a shield determines whether an action is safe or not depends on the problem and implementation. The shield should allow the agent to learn an optimal policy while enforcing logical constraints. Whenever one of these constraints is violated by an action, the shield interferes. This means that the model that is used for the shield should be able to recognize it when an action violates a constraint.

Shields should have two properties: correctness and minimum interference [6]. The model that

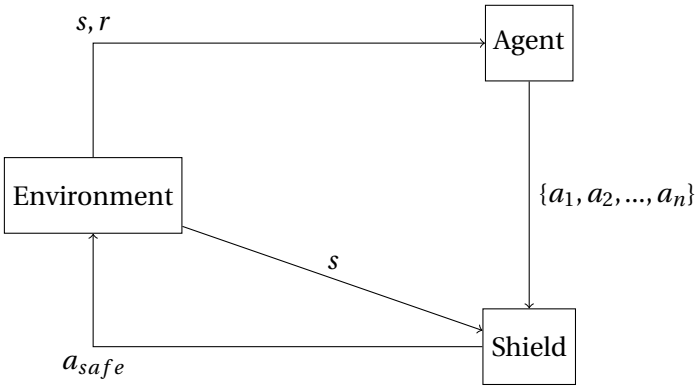



Figure 2.5: Schematic representation of a shield during training. At every step, the agent makes a sorted list of actions in descending order of preference based on $s$. The shield picks the first action $a_{safe}$ that is safe from this list.

determines whether an action violates the safety constraints should be correct. Whenever an action is unsafe, it should always be detected. Furthermore, the shield should have minimum interference: it should only interfere whenever an action is unsafe.

In this research, two different models are used to create two shielding approaches: one model that can determine if an action is safe and one model that can propose a safe action for every state in which this is possible. Much of the existing shielding research uses some kind of model that directly checks whether an action is safe [6]. If the action is not safe, it is overruled. Such a model can be based on existing information and assumptions [33], like in this research, or on predicted risk and probabilities of staying safe [12, 14, 39]. These types of methods are similar to the shield based on the model that can check whether an action is safe or not, which will be used in this thesis.

Using a model for a shielding approach that can only propose safe actions is less common. There is one method though, called Model Predictive Shielding [43], which uses such a model. In this approach, a model is needed that can guide the agent back to a safe state. If the agent enters an unsafe state, the safety model takes over and overrules the proposed actions of the agent with safe actions. In this thesis, a slightly different approach is proposed which uses such a safe action proposing model to create a more flexible shield.

Whenever an action is overruled by the shield, its experience cannot be used for the RL agent to train on, because it does not exist since the action has not been executed. The traditional way to deal with this is to use the actual experience, but replace the actual action with the initial proposed action by the agent. This means that the agent will keep proposing unsafe actions if they yield a high reward after being overruled. By using this method, the agent does not learn to avoid unsafe actions itself, which means that the shield has to be employed both during training and execution. In this thesis, we will look at two alternative methods that do enable the agent to learn to avoid unsafe actions.

## 2.4. Related work

Because of the fact that Safe RL is a relatively recent research area, there has not been much research yet into using Safe RL for AVs. There has been some research though into Safe RL methods for AVs, usually on specific AV tasks [12, 14, 26, 33, 39, 46, 56, 73]. All of these papers were written quite recently: in 2018 or later. This means that there is still a lot of research to do on the subject of Safe RL for AVs. This section first describes some general Safe RL approaches for AVs, after which a couple of shielding approaches are discussed.

### 2.4.1. Safe Reinforcement Learning for Autonomous Vehicles

Some of the existing research uses different Safe RL methods than the ones that are proposed in this thesis. For example, Fulton and Platzer [26] use a model to estimate the current state of the agent. If the observed state does not match with this estimated state, the chosen strategy is abandoned and the agent is guided to a safe part of the state space. This is tested on Adaptive Cruise Control for AVs. Lütjens et al. [46] propose MC-Dropout and Bootstrapping for awareness about the agent's uncertainty. These methods are embedded in a Safe RL framework, which changes the AVs navigation based on uncertainties. This is a risk-directed type of exploration, which was described in the previous section. The system is built for navigation around pedestrians. Wen et al. [73] propose a different Safe RL method for AVs: Parallel Constrained Policy Optimization. This method extends the actor-critic architecture with a third network that estimates a risk function. Based on this risk function, the behavior of the agent is altered and safety is ensured. It is tested on two AV scenarios: lane-keeping and an intersection.

### 2.4.2. Shielding for Autonomous Vehicles

There are a couple of papers which use a shielding type of approach in the field of AVs, although none of them uses the term shielding. Chen et al. [14] propose a regret-based approach. At every step, a regret term is calculated based on probabilities of collisions and the cost of slowing down. Based on this regret function, a safe set of actions is created from which the agent can choose. A DDQN is used as the RL algorithm. This method can be categorized into the first type of shielding: a shield that selects a safe set of actions and then lets the agent choose which action from that set to execute. This method is tested on a highway lane change scenario.

Bouton et al. [12] suggest a different approach, in which a probabilistic model computes the probability of reaching the goal state safely for every state-action pair. A threshold is used to determine which of the actions have a high enough chance of success. The agent can then choose one of these actions. The method is tested on an intersection scenario.

A third interesting option is proposed by Krasowski et al. [39] and uses a system that plans motions of other traffic participants. A shield is then used to create a safe subset of actions, based on the other traffic participants' motions, from which the agent can choose. If no safe action exists, a verified fail-safe controller is used. This controller can bring the agent back to a safe state if this is possible. The method was tested on a lane-changing scenario.

Niu et al. [56] propose a two-stage Safe RL system. In the first stage, a model-free RL algorithm needs to learn to avoid danger at a low speed, while a rule-based shielding type of model checks its actions. In the second stage, the agent needs to learn to drive at a high speed. Now, this rule-based model is replaced with a model based on data, which again acts like a shield. This method is tested in a racing simulator with complex racing tracks.

Finally, Isele et al. [33] use the term masking instead of shielding, but they propose the same type of method as shielding. Unsafe actions are masked from the agent, so the agent can only pick a safe action. This method is tested on a simple T-junction scenario.

As described above, all papers that use a shielding approach in the area of AVs use the first type of shielding. This means that the types of shields that are designed in this research have not been tested in the field of AVs before. Most of the shields use models which can check whether an action is safe or not, based on different techniques. None of the existing research found in the field of AVs uses a model which can propose safe actions to base a shield on.

When looking at the results of all papers mentioned in this section, they all present promising and safe results. However, they only present these on very specific traffic scenarios, whereas the goal of some of the papers is to show how well Safe RL generalizes to more complex scenarios. This might indicate that the methods are not performing well enough to generalize to every possible traffic scenario. It is much easier to only make a learning algorithm work on a very specific scenario. It would be interesting to see if one of the Safe RL techniques would work on every possible traffic scene that can be encountered.

Furthermore, all of the methods described above have only been tested in simulators, although the goal of using Safe RL is to enable AVs to train in real life. AVs have to be tested in real life to be able to check whether Safe RL actually achieves its goal of making RL for AVs safe. This can only be done when the research community starts to work on Safe RL systems that can handle any possible traffic scenario. Hopefully this will soon be possible by combining the best performing Safe RL systems that work on specific traffic tasks.

# 3

# Problem Definition

In the previous two chapters, the concepts of Autonomous Vehicles (AV) and Reinforcement Learning (RL) were introduced and explained. In Section 1.3, the problem that is investigated in this thesis was shortly introduced. This chapter will more formally define this problem. After that, the evaluation criteria are listed, which are used to determine the performance of the solutions of the problem. Then, the aim of the thesis is formalized as an optimization problem. After that, the scope of the thesis for which the proposed solutions are tested is described. At the end of this chapter, the model definition is given, which contains the state and action space, the reward function and the RL algorithm that is used.

## 3.1. Problem

As introduced in Chapter 1, the implementation of AVs in our daily lives could have many advantages. Before we can use AVs, it is crucial that safe driver models exist that control the vehicles. A driver model $D$, as defined in Definition 3.1, uses information about the environment to decide which actions to take. These actions should be safe at all times.

**Definition 3.1.** *A driver model D controls an AV by mapping sensory inputs of the vehicle to control actions like accelerating, braking and steering.*

One suitable way of creating such a driver model is by using RL. The policy learned through RL can be used as a driver model. A policy maps states to actions, which in this case is a mapping from sensory inputs of a vehicle as the states, and vehicle control actions as the actions. As explained in Section 2.1.3, training is needed when using RL to learn a policy that can be used as a driver model. During the exploration part of training, the RL agent takes random actions to learn which actions work best by observing the rewards it receives. Based on these rewards it learns to behave in a certain way. During the exploitation part of training, the agent executes actions that it estimates to be the best for current states. The policies during training balance the exploration and exploitation aspects. These policies are different for every $K$ episodes (Definition 3.2). The final policy $\pi_K$ should only select actions that maximize the expected return.

**Definition 3.2.** *$\pi_k$ is the policy that is used during episode $k \in \{1..K\}$, where $K$ is the total number of episodes in an RL training process. A training policy balances random exploration and exploitation of the agent's estimated best actions. $\pi_K$ is the final policy and can be used as driver model $D$.*

Clearly, it is impossible to make any guarantees on the safety of a policy when a vehicle is executing random actions during exploration. It could drive safely, but most likely it will hit other vehicles, drive on the wrong side of the road, hit objects and cause other vehicles to crash. This

will only stop when the car breaks down. When taking estimated best actions instead of random actions during the exploitation part of training, the AV is likely to drive better, but there are still no guarantees on safety at all. The agent might have learned to crash the car into the nearest tree to avoid more accidents, instead of good driving behavior.

Consequently, at any point of training an RL agent to learn to drive, it is likely that unsafe situations occur. Therefore it would be irresponsible to use regular RL to train a vehicle to drive in real life. This would cause accidents before the agent would learn anything. It is possible to do the training process in a driving simulator, but this poses a new issue: it is unsure whether the behavior learned in a simulator generalizes to the real world. There is a whole research area related to this issue [75]. By being able to train the agent in the real world, the whole issue of transferring behavior from a simulator to the real world can be avoided.

Unfortunately, as stated before, it is not possible to train an AV agent in the real world when using regular RL because of safety issues. This means that the technique of regular RL can not be employed to create driver models. In this thesis, the aim is to use RL to train an AV while guaranteeing safety, by making the assumption that one of two models is available. The first model should for every action be able to decide whether that action is safe or not. The second model should be able to propose a safe action for every state in which a safe action is available. Given these two models, this research looks at possibilities to guarantee safety during the RL training process, while making sure that the RL agent is able to learn an efficient policy. Both safety and efficiency of the policy will be defined in Section 3.2.

## 3.2. Evaluation criteria

To be able to check whether proposed solutions are effective to solve the problem, criteria are needed which indicate the performance of these solutions. Specifically, two Key Performance Indicators (KPI) are needed which indicate safety and efficiency, the two most important elements to judge the performance of an AV. An AV needs to be safe, because nobody would use a vehicle which has a high chance of causing collisions. It also needs to be efficient. The safest options for a vehicle is to never move away from its parking spot, but that would indicate a bad performance. Therefore efficiency needs to be tested, which indicates whether the transportation component of the AV is performing sufficiently. This section defines the concepts of safety and efficiency.

Traffic safety consists of many different elements: driving below the speed limit, not braking too hard, following traffic rules, slowing down before sharp corners and many others. Two KPI's are often used in the field of AVs to indicate safety: the proximity to other vehicles [16, 33] and the number of collisions [12, 39, 46]. This thesis uses the number of collisions for the definition of safety, since all of the safety elements come down to avoiding collisions. The indicator *col*, as defined in Definition 3.3, is used to indicate safety. *col* should be as low as possible.

**Definition 3.3.** $col(\pi)$ *is the number of collisions divided by the total kilometers driven when executing policy* $\pi$.

A car not only needs to be safe but also needs to drive to a destination. Therefore an efficiency indicator has to be defined. Just like safety, efficiency can consist of multiple components. Often used KPI's in the field of AVs for efficiency are the distance traveled [21, 28], whether a destination was reached [39] and the average speed [28, 39]. The average speed *sp*, as defined in Definition 3.4, indicates efficiency in this research. The higher *sp*, the better, as long as the vehicle's behavior is safe.

**Definition 3.4.** $sp(\pi)$ *is the average speed in m/s of the AV when executing policy* $\pi$.

The return (sum of rewards) is also often used in the field of AVs to evaluate performance [14, 16, 28, 39]. The return can show whether an RL agent learns something during training, but it does

| | Return | Safety *col* | Efficiency *sp* |
|---|---|---|---|
| Training | ✓ | ✓ | ✓ |
| Evaluation | | ✓ | ✓ |

Table 3.1: The evaluation criteria that are used during training and evaluation to evaluate the RL agent's performance.

not directly indicate performance. Therefore the return is used in this thesis to show the learning process during training, but not to indicate performance of the policies that were learned through RL. The reward function will be defined in Section 3.5.3. In Table 3.1, an overview of the evaluation criteria for both training and evaluation is shown.

## 3.3. Aim

Now that safety and efficiency are defined, the goal of this thesis can be formalized. The aim is to train an RL agent to drive while only using safe policies during training and execution. The concept of a safe policy is defined in Definition 3.5. It can either be shown that a policy is safe by proving that the agent can never have a collision when using this policy, or it can be shown by extensive simulation. Two models, one that is able to judge action safety and one that can propose safe actions, are used to ensure that the policies are safe. While the policies during training should only be safe, policy $\pi_K$ should also be as efficient as possible.

**Definition 3.5.** *A policy $\pi$ is safe if it is shown that $col(\pi) = 0$.*

The goal can be defined as an optimization problem based on *col* and *sp*. This optimization problem is shown in Equation 3.1. Every policy that is used during training should be completely safe: $col(\pi_k) = 0$ for every episode $k$. It is crucial that the AV never crashes, otherwise it should not be allowed on the road. That is why this constraint is introduced. Within this safety constraint, the goal is to let the final policy drive as efficient as possible, so to maximize $sp(\pi_K)$.

$$\begin{aligned} &\text{maximize} && sp(\pi_K) && \\ &\text{subject to} && col(\pi_k) = 0 && \forall k \in \{1..K\} \end{aligned} \tag{3.1}$$

Based on this optimization problem and the introduced definitions, it is possible to rephrase the research question: *Given a model that can determine action safety and a model that can propose safe actions for Autonomous Vehicles, how can shielding be used to learn a policy with Reinforcement Learning that maximizes $sp(\pi_K)$, while ensuring that $col(\pi_k) = 0$ for every episode $k$?*

## 3.4. Scope

Now that the problem is stated, the scope needs to be defined. Two scenarios are created on which the problem should be solved in this research. The first scenario is very simple and can show whether the problem can be solved perfectly in a simple setting. The second scenario is a bit more challenging and can show whether it is possible to solve this problem in a more complex setting. The two scenarios are described in the following two subsections.

### 3.4.1. Scenario 1: One lane without traffic

The task in scenario 1 is quite simple: the AV only needs to learn to drive straight on a straight road. Whenever the end of the road is reached, the simulation is ended. The vehicle is always spawned in the right direction at the start of the road. The optimal policy is to use full throttle and no steering at all times. The environment is not uncertain in any way, so the best sequence of actions is the same for every episode. This scenario is very simple, it is easy to learn the optimal policy within a short time. Scenario 1 is used to ensure that the different techniques proposed in this thesis have the ability to reliably learn a simple task within a short time.

### 3.4.2. Scenario 2: One lane with traffic

In the second scenario, the agent needs to learn to not hit other vehicles while driving on a road. Staying on the road is not part of the learning task: the vehicle can not steer and therefore it is unable to leave the road. On average, around ten vehicles are on this road at every episode. This can differ, meaning that the AV should be able to drive safely for both an empty road and a crowded road. Scenario 2 is harder than the first scenario: the best action depends on the current state. In some of the episodes, the road is empty and the vehicle can use full throttle and get a maximum reward. In other episodes, the road is jammed and the best behavior is standing still to not hit another vehicle. When the AV is spawned, it is ensured that it can not hit anything within two timesteps, which is the time needed to come to a halt after being spawned with an initial velocity of 7 m/s. This guarantees that the AV always has the opportunity to avoid hitting another vehicle. Scenario 2 is used to check whether the problem of this thesis can be solved in a more challenging environment.

## 3.5. Model definition

After the description of the scope of the thesis, the actual model in which the AV needs to operate can be defined. This section will describe parts of the Markov Decision Process that is designed for this research. The state and action space will first be defined for both scenarios, which were described in Section 3.4. After that, the reward function is defined, which is the same for both scenarios. The transition function is unknown and depends on the simulator that will be used. This transition function should adhere to the laws of physics. At the end of this section, the choice of RL algorithm, which is the Double Deep Q-Network (DDQN), is explained.

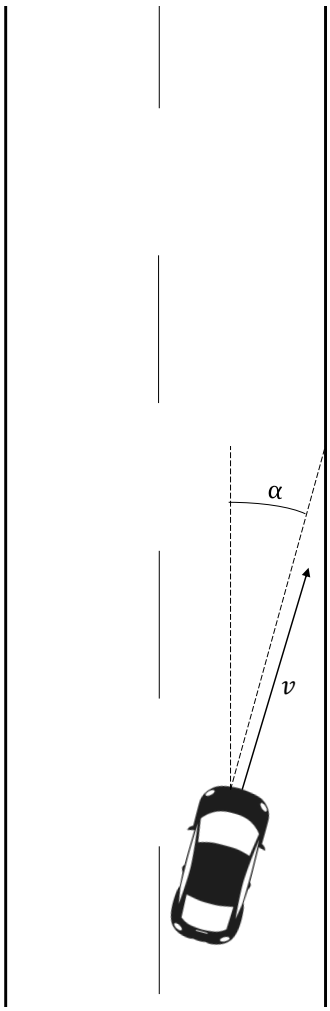
### 3.5.1. State space

The state space differs for the two scenarios. For scenario 1, less information is needed than for scenario 2. The state space for both scenarios is continuous, meaning that it can consists of multiple continuous variables. A continuous state space is chosen because it would be hard to model a relatively complex state space like the one for AVs in a discrete manner. The state space should be constructed in such a way that the agent gets enough information from the environment to be able to learn and execute a policy that can safely and efficiently drive within the situations that are presented to it.

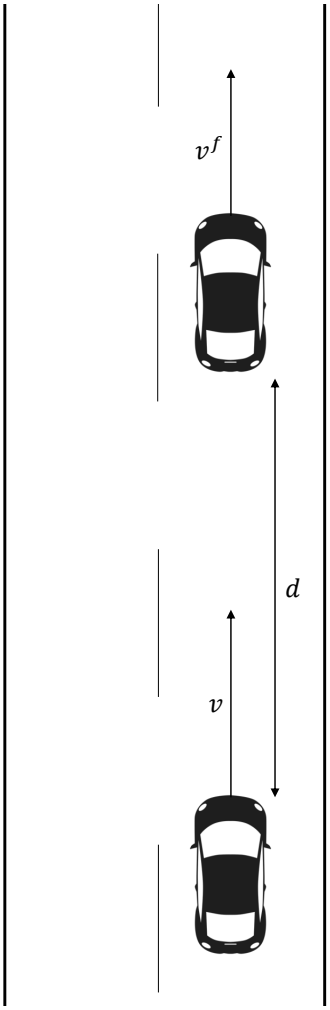
For scenario 1, the state space $s^1$ only consists of the current velocity $v$ in m/s of the AV and the relative angle $\alpha$ in degrees between the AV and the road, as shown in Equation 3.2. The velocity is needed for the DDQN to predict the reward and Q-values for actions, since this depends on the future velocity of the vehicle. The future velocity is dependent on the current velocity and action, which means that the agent needs to receive its current velocity to generate accurate Q-values. Other than that, the AV should be aware when it is not driving straight on the road, since that would cause the agent to leave the road, which is what the agent needs to learn to avoid in scenario 1. The relative angle between the AV and the direction of the road $\alpha$ is included in the state space to provide the agent with this information. If $\alpha$ is not zero, the AV is not driving straight on the road. A representation of the environment of scenario 1, containing the state space variables, is shown in Figure 3.1a.

$$s^1 = [v, \alpha] \tag{3.2}$$

In scenario 2, there are multiple vehicles that share the road with the AV. Information about the vehicle that drives in front of the AV needs to be included in the state space. The AV does not need to learn to stay on the road in this scenario, so the relative angle $\alpha$ can be omitted. As shown in Equation 3.3, three variables are used for scenario 2: the velocity in m/s of the AV $v$, the distance in meters between the AV and the vehicle in front of it $d$ and the velocity in m/s of the vehicle in front of the AV $v^f$. With these three variables, it should be possible for the AV to learn a policy that




(a) Environment of scenario 1, containing the velocity $v$ and the relative angle to the road $\alpha$.




(b) Environment of scenario 2, containing the velocity $v$, the velocity of the leading vehicle $v^f$ and the distance between the two $d$.

Figure 3.1: Schematic representations of the environments of scenario 1 and 2, containing the variables that are used as the state space.

is efficient while maintaining enough distance to the vehicle in front of it. A representation of the environment of scenario 2, containing the state space variables, is shown in Figure 3.1b.

$$s^2 = [v, d, v^f] \tag{3.3}$$

### 3.5.2. Action space

Just like the state space, the action space differs per scenario. The action space defines exactly what an agent can do within an environment. As mentioned earlier, the action spaces that are used in this research are discrete because of the way that the shields are built. In both scenarios, the agent needs to be able to control longitudinal movement: acceleration and deceleration. Eleven options for longitudinal actions were chosen: $\{-1.0, -0.8, -0.6, -0.4, -0.2, 0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$. -1.0 means a full brake, 1.0 means a full throttle and 0.0 means not using the pedals at all. This number of actions $n$ should be high enough to ensure that the agent can choose from enough actions to learn a flexible policy, but also low enough to limit the complexity of the problem.

In scenario 1, lateral control is also part of the problem. The agent needs to learn to stay on a

| | $a_{longitudinal}$ | $a_{lateral}$ | $n$ |
|---|---|---|---|
| **Scenario 1** | $\{-1.0, -0.8, -0.6, -0.4, -0.2, 0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$ | $\{-1.0, 0.0, 1.0\}$ | $11 + 2 = 13$ |
| **Scenario 2** | $\{-1.0, -0.8, -0.6, -0.4, -0.2, 0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$ | | 11 |

Table 3.2: Action spaces for the two scenarios divided into longitudinal and lateral actions.

straight road, which only makes sense when it is able to actually leave the road. On top of the longitudinal actions, the agent is given two lateral action: -1.0 and 1.0. When the agent chooses one of these two lateral actions, the throttle or brake cannot be used, which is equal to the longitudinal action of 0.0. -1.0 stands for fully steering to the left and 1.0 for fully steering to the right. In total there are $11 + 2 = 13$ different actions in scenario 1, eleven longitudinal actions and two lateral actions. For scenario 2, only longitudinal control is needed. In Table 3.2, an overview of the action space for the two scenarios is shown, as well as the number of actions $n$. Note that it depends on the type of vehicle used how many degrees the vehicle turns when it is fully steering right or left and how much it accelerates or decelerates when fully or partially using the throttle or brake.

### 3.5.3. Reward function

An AV needs to be safe and efficient. The reward function should be designed in such a way that the agent is able to learn to behave safely and efficiently. Therefore a safety and an efficiency component are included in the reward function $R(s')$, which is shown in Equation 3.4. The reward is based on the next state $s'$ of the AV after executing an action, specifically on the velocity $v$ and whether the AV has had a collision. $R(s')$ is the same for both scenarios. In scenario 1, leaving the road is counted as a collision. If $\alpha$ is not zero while $v$ is higher than 0, it means that the agent is leaving the road. In scenario 2, a collision with another vehicle is counted as a collision. The AV collides with the vehicle in front of it if $d$ becomes zero.

$$R(s') = \begin{cases} \text{-1} & \text{if collision} \\ \frac{v}{v_{max}} & \text{otherwise} \end{cases} \tag{3.4}$$

The reward function $R(s')$ consists of two options. If the AV has had a collision, the reward is set to -1. Otherwise, the reward is based on the velocity $v$ of the AV. $v$ is divided by the maximum velocity $v_{max}$ of the AV to ensure that the reward based on velocity always has a value between 0 and 1. The two options in the reward function always outputs a reward between -1 and 1. Using a normalized reward like this is good practice when using Deep RL [51]. The reward function is inspired by the standard reward function that is used in highway-env [42], an environment for decision making for AVs. Keeping the reward function simple makes it easier to determine why the agent behaves in a certain way.

### 3.5.4. Reinforcement Learning Algorithm

The aim of this research is to check how the proposed shields can influence traffic safety of an RL algorithm, not to create the best possible RL system to train driver models. Nevertheless, it is beneficial to use a state-of-the-art RL algorithm, since that will make the results of this thesis more applicable to recent research in the field of RL for AVs. One state-of-the-art RL algorithm is the Double Deep Q-Network (DDQN), which performs very well on different RL problems, for example OpenAI's Cartpole problem [13, 34]. It is also used often as the RL technique in the field of AVs [14, 54, 58, 74].

As explained in Section 3.5.1, the chosen technique has to be able to use a continuous state space, since it would be unfeasible to map a complex AV environment to a discrete state space. Other than that, the shielding technique, which was described in Section 2.3.3, that is used in this research requires a discrete action space. The DDQN, which was described in Section 2.2.3, was

chosen to use in this research as the baseline RL algorithm. It works with a continuous state space, a discrete action space and it is a well performing RL technique.

# 4

# Shielding

This thesis aims to solve the problem of Autonomous Vehicles (AV) not being able to train safely when using Reinforcement Learning (RL). Two shielding techniques are proposed to solve this problem. This chapter describes the two shields: the Safety Checking Shield and the Safe Initial Policy Shield. These shields are based on a model which can check safety for every action and a model that can propose a safe action for every state respectively. For both methods, the general ideas and implementation details are discussed. Furthermore, two methods are proposed to enable the agent to learn from actions that were overruled by a shield: fabricating experiences and using an alternative loss function. This chapter aims to answer the second, third and fourth subquestion, which were stated in Section 1.4.

## 4.1. Safety Checking Shield

The Safety Checking Shield (SCS) is based on post-posed shielding: letting the agent pick an action out of all possible actions, checking that action and overruling it if it is unsafe [6]. This type of shielding has been described previously in Section 2.3.3. Figure 2.5 shows how it works. At every step, the agent sends a list of all actions to the shield, sorted in descending order on preference of the agent. In our case, this preference is based on the estimated Q-values of the actions. A discrete action space is required to be able to receive a sorted list of actions. The Double Deep Q-Network (DDQN), which is used in this research, uses a discrete action space. During an exploration step, this list is sorted in a random order. Based on the current state of the agent, the shield checks the actions in order of the list on safety. When it finds a safe action, that action is sent to the environment. This means that the preferred action of the agent is only overruled when it is not safe, and in that case the safe action with the highest Q-value is chosen.

The main purpose of this section is to describe how the SCS checks an action's safety, specifically in the domain of AVs. The methods used for this are not based on existing literature, since there has not been much use of shielding in the field of AVs. Most other literature has proposed shields that work in a different manner [12, 14, 39], so the methods used in that literature cannot be used for the SCS. The only similar type of shield, called a mask [33], is tested on a T-junction scenario, which means that it is not usable for the scenarios that are used in this thesis. All of these different methods of shielding for AVs have been described in Section 2.4.2.

The SCS that is proposed in this chapter consists of two models, which together can judge whether an action is safe or not under some given assumptions, depending on the environment. The shielding process is shown in Figure 4.1 and the pseudocode of it can be found in Algorithm 3. As mentioned earlier, the agent sends a sorted list of all actions to the shield. Furthermore, the shield uses the current state of the environment as input. For every action, the future state after taking that

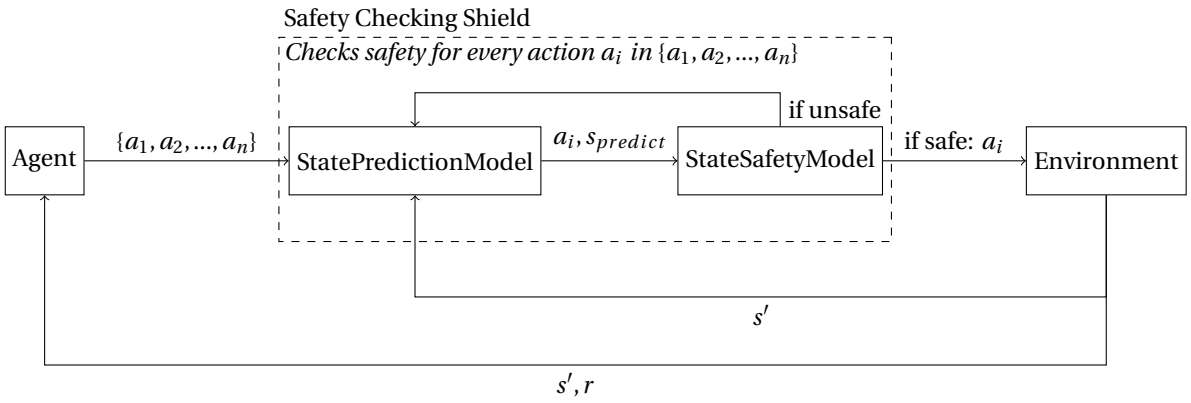



Figure 4.1: Schematic representation of the SCS during training. At every step, the agent sends a sorted list of actions to the shield. Based on the current state, the shield predicts the next state for action $a_i$. If this next state is considered safe by the StateSafetyModel, $a_i$ is sent to the environment. If it is not considered to be safe, the next action $a_{i+1}$ is checked.

**Algorithm 3** Safety Checking Shield

```
s ← current state
{a_1, a_2, ..., a_n} ← sorted list of actions based on the agent's preference (Q-values)
for a ∈ {a_1, a_2, ..., a_n} do
    s_predict ← StatePredictionModel(s, a)
    if StateSafetyModel(s_predict) then
        return a
```

action is predicted by the StatePredictionModel. It is then checked whether that future state is safe or not by a different model, the StateSafetyModel. If the predicted future state is safe, the action that resulted in that state is considered safe. The first safe action is sent to the environment.

This section will start with a brief safety analysis that introduces the requirements of the two models and analyzes the safety of the shield given that these requirements are met. After that, the two models and their requirements are more thoroughly discussed. Finally, the implementation details for the scenarios that are used in this thesis are presented. It will be shown that the implementations meet the requirements of the models and therefore result in a shield that ensures safety of the AV on the chosen scenarios.

### 4.1.1. Safety Analysis

The SCS is built upon two models, the StatePredictionModel $M_{SP}$ and the StateSafetyModel $M_{SS}$. The first model predicts the future state of the vehicle after taking an action. The second model checks whether that state is safe or not. If both models are accurate, it is evident that the SCS is safe. In that case, the $M_{SP}$ would produce the exact future state, which could then reliably be checked by the $M_{SS}$. It is unrealistic though to create completely accurate models for the complex problems that they need to work on, partly because the future state can not be predicted correctly all of the time in a stochastic environment. That is why a couple of requirements are introduced in this section, which ensure safety of the SCS if they are met.

The $M_{SP}$ should for every state variable in $s$ not predict it to be safer than it might be in the actual future state. Furthermore, the $M_{SS}$ should always assume the worst case scenario about the environment. It should only estimate the predicted state by $M_{SP}$ to be safe if there is even a safe trajectory for the agent when everything goes wrong. The next two subsections will explain these requirements in more detail.

When the two models meet these two requirements, the $M_{SP}$ will produce a predicted state which is never safer than the actual future state. The $M_{SS}$ will then use this predicted state, which is

as unsafe or unsafer than any state that the agent can end up in, and check if there is a safe trajectory, even if everything goes wrong. If a safe trajectory exists, it means that even the unsafest state that the agent can end up in is safe. That means that the initial action must be safe. The shield only allows actions that were estimated to be safe under these requirements and therefore the shield should ensure safety. The agent should never end up in a state where no safe action is available, since actions are only allowed if they put the agent in a state where a safe trajectory is present.

### 4.1.2. StatePredictionModel

The first part of the shielding process is to predict the future state $s_{predict}$ of the environment after one step, based on an action and the current state. The model that makes this prediction is called the StatePredictionModel $M_{SP}$. Based on $s_{predict}$, a second model determines whether the action that resulted in $s_{predict}$ is safe. Therefore it is important that $s_{predict}$ is predicted accurately.

$M_{SP}$ aims to map the current state and an action to an accurate predicted new state: $M_{SP}(s,a) \rightarrow s_{predict}$. We call the different variables within a state $\sigma$. For example, in the state space of scenario 1, which was described in Section 3.5.1, $\sigma_1$ would be the velocity of the vehicle and $\sigma_2$ the relative angle to the road. In a continuous state space, the StatePredictionModel changes zero or more variables $\sigma$ in $s$ to create $s_{predict}$. For every $i$, variable $\sigma_i^{predict}$ in $s_{predict}$ should be as close as possible to $\sigma'_i$, the variable in the future state $s'$. $M_{SP}$ should be able to make an accurate prediction for every variable $\sigma_i^{predict}$. If the prediction is not close to the actual state, the shield might overrule too much actions, which hurts the learning process of the agent. Note that $\sigma'_i$ cannot be observed before executing an action, which is why $\sigma_i^{predict}$ is needed.

For some fields and tasks, models are available that can make a perfect prediction of $\sigma_i^{predict}$ for every variable $i$ in $s$. If the predictions are not perfect though, every predicted variable should be biased towards the risky side. To illustrate the concept of risk, an example is given. In a task where the goal is to never let a tank overflow, one of the variables is the fullness of the tank. If this variable cannot be predicted 100% accurately, $M_{SP}$ should always overestimate this parameter. If it would be underestimated, this could give the StateSafetyModel a false sense of safety, because it would underestimate the danger of the situation that the agent would end up in after taking an action. This could lead to violating a safety constraint.

Summarized, the goal of $M_{SP}$ is to map $s$ and $a$ to $s_{predict}$, where every variable $\sigma_i^{predict}$ in $s_{predict}$ is as close as possible to the actual future state variable $\sigma'_i$, while ensuring that $\sigma_i^{predict}$ is never estimated to be less risky than $\sigma'_i$.

### 4.1.3. StateSafetyModel

The StateSafetyModel $M_{ss}$ needs to determine whether $s_{predict}$ is safe or not for every possible $s_{predict}$: $M_{ss}(S_{predict}) \rightarrow$ {True, False}. $s_{predict}$ is considered to be safe if it is possible to not violate safety constraints in that state. At least one action should be available in $s_{predict}$ that would result in an agent's trajectory in which safety constraints are not violated.

$M_{SS}$ is basically a function which calculates whether violating safety constraints is unavoidable in $s_{predict}$. This is the case when the safety constraints would be violated for every action that can be taken after reaching $s_{predict}$. If there is no safe action available, $M_{SS}$ returns false. If one or more actions actions exist that will not result in violating safety constraints, $M_{SS}$ returns true.

It is very dependent on the actual task how the StateSafetyModel is implemented. Note that we usually do not know the actual transition function, so the StateSafetyModel should be implemented in such a way that uncertainties about the environment will not result in dangerous situations. For every uncertainty, the worst-case scenario should be assumed. For example, in the case of AV's, if the distance to another vehicle in $s_{predict}$ can be between 10 and 20 meters, the StateSafetyModel should always estimate it to be 10 meters since that would be the most dangerous situation. By

always assuming the worst-case scenario, the StateSafetyModel might estimate safe states to be unsafe, but it will never estimate unsafe states to be safe. This ensures that the agent is never put in an unsafe situation in which safety constraints can be violated.

### 4.1.4. Implementation

This subsection describes the specific implementations of $M_{SP}$ and $M_{SS}$, which were discussed in the previous two subsections, on the two scenarios that were described in Section 3.4. The implementation of $M_{SP}$ should map the current state and action to a predicted state: $M_{SP}(s,a) \rightarrow s_{predict}$. The implementation of $M_{SS}$ should map $s_{predict}$ to a boolean, indicating whether $s_{predict}$ is safe. With the two models, the current state and an action are mapped to a boolean which indicates whether the action is safe in the current state. The safety constraint that needs to be respected in the two scenarios is having no collisions. Both leaving the road or colliding with other vehicles or objects count as collisions. The two models should result in a shield that overrules all actions that could cause a collision.

There are some requirements that the implementations of the $M_{SP}$ and $M_{SS}$ need to fulfill if they are not completely accurate. Section 4.1.2 showed that $M_{SP}$ should either predict every state variable $\sigma$ in $s$ accurately, or it should estimate it to be riskier than the actual future state variable. For the $M_{SS}$, it is required that uncertainties about the environment never result in underestimating the risk of a situation. For every uncertainty about the environment, the worst-case scenario should be assumed. The implementation details for both scenarios will show how the implementations meet these two requirements. We need to make two assumptions here. First, it is assumed that nothing irregular happens. Examples of irregular events are trees falling onto the road, children jumping before the AV or other cars driving on the wrong side of the road. The SCS is not prepared for such situations. Since the experiments in this research are performed in a simulator, it is guaranteed that these irregular events do not occur. In the real world, we do not have this guarantee. In addition, the assumption is made that we know the exact acceleration and deceleration of the AV for each action.

*Scenario 1*

For scenario 1, the state space consists of two variables: the velocity $v$ and the relative angle $\alpha$ of the vehicle to the road. These variables were shown in Figure 3.1a. Since the task is to learn to stay on the road, only the second variable is important to determine whether a state is safe (in this thesis, there is no difference between having a collision at a high velocity or at a low velocity). $M_{SP}$ maps the current state and action to $s_{predict} = (v_{predict}, \alpha_{predict})$ in which only $\alpha_{predict}$ is important. $\alpha_{predict}$ determines whether the vehicle will be driving straight on the road or not.

For $M_{SS}$, it does not matter how much of the AV has left the road, if it leaves the road even slightly it is counted as a collision. Therefore $\alpha_{predict}$ should be zero if the proposed action keeps the vehicle on the road and larger than zero otherwise, but it does not matter exactly what the value is in that case. $\alpha_{predict}$ is based on the steering action. If the vehicle steers, which happens if the lateral action is not zero, $\alpha_{predict}$ is set to 1. Otherwise, $\alpha_{predict}$ is set to 0. Equation 4.1 shows this simple implementation of the $M_{SP}$. Since $v_{predict}$ is irrelevant, its predicted value is set equal to its current value.

$$M_{SP}(s,a) = M_{SP}((v,\alpha),(a_{longitudinal},a_{lateral})) = (v_{predict},\alpha_{predict}) = \begin{cases} (v,0) & \text{if } a_{lateral} = 0.0 \\ (v,1) & \text{otherwise} \end{cases} \tag{4.1}$$

The requirement for the $M_{SP}$ is that all state variables, in this case $(v_{predict}, \alpha_{predict})$, are either completely accurate or biased towards the risky side. $v_{predict}$ is not used for the shield, so we ignore this variable. $\alpha_{predict}$ is always larger than 0 when any lateral action is taken, which means it never underestimated the risk of leaving the road. Therefore the $M_{SP}$ meets the requirement.

It is straightforward to determine whether the state is safe for scenario 1. As shown in Equation 4.2, the state is safe if $\alpha_{predict}$ is 0, and unsafe otherwise. $M_{SS}$ is a very simple rule for this scenario, based on $\alpha_{predict}$. $M_{SS}$ meets the requirement of always assuming the worst-case scenario. Only when the AV is predicted to drive completely straight on the road, it estimates the state to be safe.

$$M_{SS}(s_{predict}) = \begin{cases} \text{True} & \text{if } \alpha_{predict} = 0 \\ \text{False} & \text{otherwise} \end{cases} \tag{4.2}$$

*Scenario 2*

The state space for scenario 2 consists of three variables: the velocity of the AV $v$, the distance to the leading vehicle $d$ and the velocity of the leading vehicle $v^f$. These variables were shown earlier in Figure 3.1b. $s_{predict}$ is predicted using Equation 4.3. The first variable that is predicted is the future speed $v_{predict}$. First, the acceleration $acc$ after taking the proposed action is multiplied by the size of the timestep $T$ to get the difference in velocity for one step. Then, the difference is added to the current velocity $v$ to end up with the predicted future velocity of the AV.

For the future distance to the leading vehicle $d_{predict}$, the distance driven needs to be predicted. First, the average speed during the timestep is calculated by adding the current speed $v$ to half of the acceleration during the timestep $(T \cdot acc)/2$. This average speed is then multiplied by the length of the timestep $T$ to obtain the distance that is driven. For safety, it is assumed that the leading vehicle does not move within the timestep. Therefore the predicted distance driven is subtracted from the current distance to the leading vehicle $d$. Making the assumption that the leading vehicle does not move means that even if the leading vehicle fully used its brake during that timestep, the predicted distance is still smaller than the actual distance. The shield will thus never overestimate the distance between the AV and the leading vehicle, which is the requirement of $M_{SP}$. For the same reason, $v^f_{predict}$ is set to 0. It would be possible to be a bit less conservative by actually estimating what would be the minimal driven distance and maximum deceleration of the other vehicle within the next timestep. In this scenario, this does not make a large difference because of the low velocities of the vehicles. Usually they can come to a halt within one timestep and within a couple of meters. Therefore is was simpler to directly set these variables to 0 and not bother with the couple of meters difference this could make.

$$s_{predict} = (v_{predict}, d_{predict}, v^f_{predict}) = \left(v + T \cdot acc, d - T \cdot \left(v + \frac{T \cdot acc}{2}\right), 0\right) \tag{4.3}$$

Following from the requirement of the $M_{SP}$, $v_{predict}, d_{predict}$ and $v^f_{predict}$ should all be completely accurate or less safe than the actual future state variables. $v_{predict}$ can be calculated precisely, since it is possible to accurately measure the current speed $v$ and the size of the timestep $T$ is set manually. As stated before, $acc$ is assumed to be accurate, so $v_{predict}$ should be accurate. $d_{predict}$ and $v^f_{predict}$ are both estimated to be more risky than they actually are, since it is assumed that the vehicle in front of the AV does not move within the next timestep. Therefore the predictions of all three variables meet the requirements.

The predicted state $s_{predict}$ is safe if there is a way to avoid a collision. In scenario 2, the ultimate action to avoid hitting another car is to fully use the brake. Therefore $s_{predict}$ is assessed to be safe if the AV does not hit another vehicle when using a full brake until it comes to a halt.

To check whether the AV would not hit the vehicle in front of it, the distance that is needed to come to a halt, called the braking distance $d_{brake}$, is required. Equation 4.4 shows how the braking distance is calculated. The predicted velocity $v_{predict}$ is divided by the maximum deceleration $dec$. This division results in the time needed to come to a halt. This is multiplied with the average velocity during that time, which can be estimated to be $0.5 \cdot v_{predict}$ if we assume that braking linearly decreases the velocity.

$$d_{brake} = \frac{1}{2} v_{predict} \cdot \frac{v_{predict}}{-dec} \tag{4.4}$$

Now that the braking distance $d_{brake}$ is calculated, it needs to be compared to the predicted future distance to the vehicle in front of the AV $d_{predict}$. If the braking distance $d_{brake}$ plus a buffer distance $d_{buffer}$ is larger than the distance $d_{predict}$ to the vehicle in front of the AV, the state $s_{predict}$ is not safe, since it might not be possible to brake in time before hitting the other vehicle. This decision of $M_{SS}$ is shown in Equation 4.5. The buffer distance $d_{buffer}$ is set to 10 meters, which is the minimum distance that the AV should keep to the vehicle in front of it at all times. The buffer distance is used for two reasons: it ensures that the lengths of the vehicles are not an issue in the calculations, and it ensures that the AV keeps an appropriate distance to other vehicles as is desired in traffic. Summarized, $M_{SS}$ first calculates the braking distance by using $v_{predict}$ and then compares it with $d_{predict}$ to output whether $s_{predict}$ is safe.

$$M_{ss}(s_{predict}) = \begin{cases} \text{False} & \text{if } d_{brake} + d_{buffer} \geq d_{predict} \\ \text{True} & \text{otherwise} \end{cases} \tag{4.5}$$

Except for the predicted variables from $M_{SP}$, $M_{SS}$ only depends on $dec$. As stated earlier, $dec$ is assumed to be accurate and therefore $M_{SS}$ is assumed to be completely accurate when the predicted variables are accurate. It was shown previously that these predicted variables are either accurate or estimated on the risky side, which means that $M_{SS}$ will never underestimate the danger of a state. This results in the $M_{SS}$ never allowing an unsafe action.

## 4.2. Safe Initial Policy Shield

The second shield that we propose is called the Safe Initial Policy Shield (SIPS), and is based on a policy that can propose a safe action for every state in which a safe action exists. The SIPS works similar to the SCS in terms of shielding. It uses post-posed shielding, which means that the agent proposes a list of actions, after which the shield checks the actions in an iterative manner until finding a safe action. That action is then sent to the environment. This process was described previously in Section 2.3.3.

The difference between SCS and SIPS is the way in which it is checked whether an action needs to be overruled. SCS uses a model which predicts the future state and a model which checks state safety. If the future state after taking an action, predicted by the first model, is estimated by the second model to be safe, the action is not overruled. SIPS uses two different models. The first model, a Safe Initial Policy (SIP), proposes a safe action for the current state. The second model is called the SafetyRangeModel, and creates a range of allowed actions based on how safe the current state is and on the action proposed by the SIP. The first action from the agent that falls within this safety range is executed. Specifically, a variable $\rho$ is calculated which determines how much the value of the executed action can differ from the value of the safe action from the SIP. This means that the actions should be sortable, possibly in multiple dimensions. For example, in the field of AVs, two dimensions in which the actions can be sorted are acceleration and steering. For every dimension, the chosen action is only allowed if it differs $\rho$ or less from the SIP action. The range that is created here could be used for a preemptive shielding type of approach, but it was chosen to let the agent pick any action and then check whether it is safe or not. This ensures that the type of shielding is equal for the SCS and the SIPS.

The whole SIPS process is shown in Algorithm 4 and Figure 4.2. The agent sends a descending sorted list of actions based on preference to the shield. Furthermore, the SIP proposes a safe action based on the state and the SafetyRangeModel determines the deviation parameter $\rho$. For every action, it is checked whether its value does not deviate more than $\rho$ from the value of the safe action. If this is the case, it is sent to the environment. Otherwise, the next action is checked.

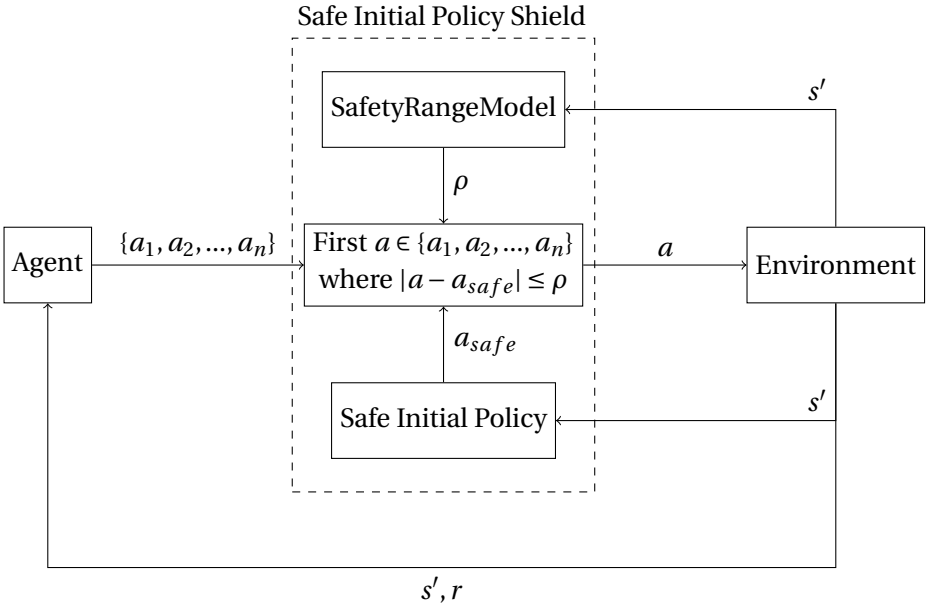



Figure 4.2: Schematic representation of the SIPS during training. The agent sends a sorted list of actions to the shield based on preference. The shield picks the first action that is safe, based on $\rho$ and $a_{safe}$ from the SafetyRangeModel and the SIP, both determined by using $s$.

**Algorithm 4** Safe Initial Policy Shield

```
s ← current state
{a_1, a_2, ..., a_n} ← sorted list of actions based on the agent's preference (Q-values)
a_safe ← SIP(s)
ρ ← SafetyRangeModel(s)
for a ∈ {a_1, a_2, ..., a_n} do
    if |a − a_safe| ≤ ρ then
        return a
```

The main ideas and setup of the SIPS will be elaborated in this section. First, the safety of the SIPS is analyzed. Then, the SIP and the SafetyRangeModel are explained. After that, the exact implementation on the scenarios created in this thesis is described. Finally, some techniques are discussed which are related to this novel method.

### 4.2.1. Safety Analysis

The SIPS uses two models, the SIP and the SafetyRangeModel $M_{SR}$. The SIP should always propose a safe action, and if possible an action that puts the agent in a state where all actions are safe. $M_{SR}$ determines how safe the current state is, and produces a deviation range depending on the state safety. It uses a linear function to determine this state safety, which should be able to generate a value between 0 and 1, where 0 is equal to being in an unsafe state and 1 is equal to being in a fully safe state.

Because of the fact that the SIP should never propose an action that would put the agent in danger in the next step, the agent can deviate from the proposed action if it is not in a safety critical situation. Since the action space should be linear and the $M_{SR}$ safety function is linear between an unsafe and a safe state, the agent is allowed to deviate more from the proposed action the safer its state is. For example, in a state that is almost completely safe, the SIP will propose an action that puts the agent in a completely safe state. The agent is then allowed to deviate a lot from that action, but not so much that it executes an unsafe action. For unsafer states, the agent can deviate less.

The function is designed in such a way that the chosen action can never deviate so much from the proposed action that it becomes unsafe.

The advantage in comparison to always following the SIP action is that the agent can learn a better policy in slightly unsafe situations, since it can try a wider range of actions in such situations. As explained though, every action that puts the agent in immediate danger is overruled.

### 4.2.2. Safe Initial Policy

The SIP needs to be able to propose a safe action in every state where a safe action is available: $\text{SIP}(s) \rightarrow a_{safe}$. Furthermore, it should never propose an action that would be safe for one step but would put the agent in a dangerous situation during future steps. The safe action is used by the SafetyRangeModel to create a range of safe actions that the executed action must fall into. Other than proposing a safe action, the SIP does not have to perform very well on a task. One of the features of the SIPS system is that the RL algorithm is able to learn to perform better on a task than the SIP, while learning to behave safe in safety critical situations by following the SIP's behavior is such situations.

It is very dependent on the actual task what kind of model can be used as a SIP. For a simple task, this could be a simple rule based model which knows which exact action to propose for every unsafe state. For more complex tasks, a smarter model is needed. The only requirement is that it always proposes a safe action whenever a safe action is available in a state. If this requirement is not met, the safety of the SIPS system can not be guaranteed.

As mentioned, it does not matter much how well the SIP performs in terms of efficiency. In safe situations, the proposed action by the SIP is not used in the shielding process, as will be explained in the next section. Therefore it does not matter how efficient the SIP is in safe situations. For slightly unsafe situations, the actions of the agent can still deviate much from the SIP, so the efficiency of the SIP does not matter much here as well. Only in very risky situations the agent always needs to follow the exact behavior of the SIP, which means that it can not learn to behave more efficient than the SIP in such situations. This is not an issue, since it is desired that the agent closely follows the SIP in dangerous situations to ensure safety.

### 4.2.3. SafetyRangeModel

Now that the SIP is defined, which can propose a safe action at every step whenever a safe action is available, a second model is needed which determines how much the value of the agent's action can deviate from the value of the safe action. Specifically, this SafetyRangeModel $M_{SR}$ needs to determine the allowed deviation $\rho$, based on the state: $M_{SR}(s) \rightarrow \rho$.

If a continuous action space with one variable is used, or a discrete action space in which all actions can be sorted in one dimension, $\rho$ consists of one value. For example, all longitudinal actions, which were described in Section 3.5.2, can be sorted based on their acceleration. $\rho$ indicates the amount that the value of the actual action can deviate from the value of SIP action. If a discrete action space is used, the value of the chosen action must fall into the allowed range. If there are multiple variables in a continuous action space or multiple dimensions in which the actions can be sorted in a discrete action space, $\rho$ can consist of multiple values indicating the allowed deviation for every dimension. In this research, $\rho$ only needs to be used for one dimension for every scenario, so a multi valued $\rho$ is not tested in practice. It works exactly the same though as a single value $\rho$. The only difference is that $\rho$ need to be calculated for multiple dimensions instead of one. Note that this only works for multiple dimensions if every dimension influences the safety independently.

To determine $\rho$, first two states need to be determined: one safety critical state $s_{critical}$ and one fully safe state $s_{safe}$. $s_{critical}$ is a state in which the agent is very close to violating the safety constraints and in which the SIP needs to be followed very carefully to avoid actually violating them. $s_{safe}$ is a state in which it does not matter which action the agent takes, it will never put the agent

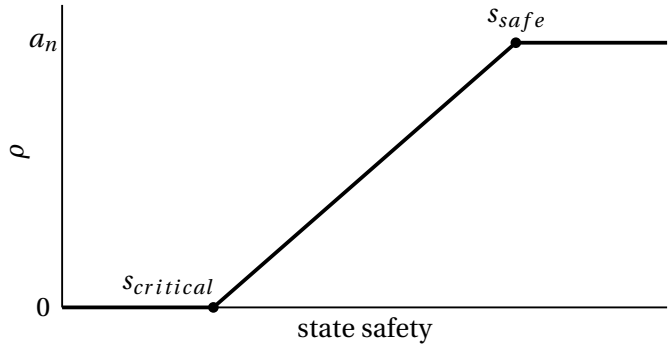



Figure 4.3: Schematic plot of $\rho$ over the state safety.

in immediate danger of violating safety constraints. Note that $s_{safe}$ and $s_{critical}$ do not necessarily have to contain all the state variables, only the ones that are relevant for the safety of the state.

A linear function $f_{ss}$ is needed which measures the safety of a state based on (some of) the variables in the state space. For example, if the task is to avoid overflowing a tank, $f_{ss}$ could be based on the fullness of the tank and the speed at which the tank is filled. If the current state is unsafer than $s_{critical}$, $\rho$ is set to 0, which means that only the action from the SIP can be chosen. If it is safer than $s_{safe}$, $\rho$ is set to $a_n$ (the value of the action $n$), which means that all actions can be chosen. If a state is between $s_{critical}$ and $s_{safe}$ in terms of safety, $f_{ss}$ determines $\rho$. Equation 4.6 shows how $\rho$ is determined exactly. This is also visualized in Figure 4.3.

$$\rho = \begin{cases} 0 & \text{if } f_{ss}(s) < f_{ss}(s_{critical}) \\ a_n & \text{if } f_{ss}(s) > f_{ss}(s_{safe}) \\ a_n \cdot \frac{f_{ss}(s) - f_{ss}(s_{critical})}{f_{ss}(s_{safe}) - f_{ss}(s_{critical})} & \text{otherwise} \end{cases} \tag{4.6}$$

It is dependent on the actual problem how $f_{ss}$ is implemented exactly and how $s_{safe}$ and $s_{critical}$ are determined. It should be ensured that any action can be safely executed when the agent is in $s_{safe}$, while $s_{critical}$ is a state that is as unsafe as possible, while at least one action is still safe. It is not a big issue if multiple actions are safe in $s_{critical}$, this only makes the SIPS a bit more conservative.

### 4.2.4. Implementation

The SIPS needs to models. A SIP, which can produce a safe action $a_{safe}$ based on the current state $s$, and a SafetyRangeModel $M_{SR}$, which produces the deviation parameter $\rho$ based on $s$. With these two models, the range of safe actions is defined, which consists of all actions of which the value is between $a_{safe} - \rho$ and $a_{safe} + \rho$.

This subsection explains the specific implementations of the SIP and $M_{SR}$ for the two scenarios that are used in this thesis, which were described in Section 3.4. Again, the implementations of the models must meet some requirements. The SIP must be able to propose a safe action for every state in which a safe action is available. For the SafetyRangeModel, a linear function over the actions needs to be designed and the two states $s_{safe}$ and $s_{critical}$ need to be determined. In $s_{safe}$, all actions must be safe, while at least one action must be safe in $s_{critical}$. Just like for the SCS, it is assumed that no irregular events happen when the SIPS is employed and it is assumed that we know the exact acceleration and deceleration for each action.

*Scenario 1*
Because of the simplicity of scenario 1, its SIP is also very simple. First, the desired velocity is determined, which is set to eighty percent of the speed limit $v_{limit}$. If the vehicle drives faster than the desired velocity, the brake variable is set to 0.6, which means using a little over half of the braking power. When the vehicle drives slower than the speed limit, the throttle is set to 0.6, which means using a little over half of the throttle power. The steering is always set to 0, which equals no steering. The SIP for scenario 1 is shown in Equation 4.7. Since the only safety constraint in scenario 1 is leaving the road, which is only possible by steering, this SIP is safe because it never uses any steering. Setting the throttle and brake to 0.6 means that the policy is somewhat efficient, but still conservative enough to be able to show that the SIPS system can produce an improved final policy.

$$\text{SIP}(s) = \text{SIP}(v, \alpha) = a_{safe} = (a_{acc}, a_{steer}) = \begin{cases} (0.6, 0) & \text{if } v < 0.8 \cdot v_{limit} \\ (-0.6, 0) & \text{otherwise} \end{cases} \tag{4.7}$$

For every possible state, the agent should not deviate from the steering action $a_{steer}$ from the SIP . It does not matter how much it deviates from the acceleration action $a_{acc}$ though, since every acceleration action is safe. Therefore the chosen dimension on which the SafetyRangeModel works is steering, not acceleration. Within the steering dimension, the AV is always in a safety critical situation, since using any steering action will put the agent in danger. Therefore for every state $s$, $f_{ss}(s) = f_{ss}(s_{critical}) = 0$, which means that $\rho$ is always 0 for steering. As a consequence, the agent must always follow the steering action from the SIP, but it can choose its own acceleration. This means that the SIPS has exactly the same effect as the SCS, all steering actions are overruled and all throttle actions are not. The reason for this is the simplicity of the task. It is expected that the results of using the SIPS or the SCS on scenario 1 are equal.

*Scenario 2*
For the second scenario, an existing Car Following model is used as the SIP, called the Intelligent Driver Model (IDM) [68]. This is the same model that is used in highway-env [42], a gym environment built for AVs. Gym is a toolkit that can be used to develop and compare RL algorithms. The IDM is collision-free in the car following scenario [68]. To be absolutely sure that no collisions happen when the IDM is used, the model's parameters are set very conservatively. IDM has a couple of limitations, one being that it can cause the vehicle to get a negative velocity, so to drive backwards, in certain situations [5]. This is unwanted in the car following scenario, but not an issue in this research since the action space does not allow the vehicle to drive backwards. IDM calculates the desired acceleration based on some variables and parameters. This calculation is shown in Equation 4.8 and Equation 4.9.

$$a_{safe} = acc \left[ 1 - \left( \frac{v}{v_0} \right)^{\delta} - \left( \frac{d^*}{d} \right)^2 \right] \tag{4.8}$$

$$d^* = d_0 + Tv + \frac{v\Delta v}{2\sqrt{acc \cdot dec}} \tag{4.9}$$

In Equation 4.8, $acc$ is the maximum acceleration of the AV, which is set to 3. It is multiplied with the term inside the brackets, which is always lower than or equal to 1. $v$ is the current velocity, $v_0$ is the desired velocity and $\delta$ is the velocity exponent, which is set to 4, based on the value of $\delta$ in the paper that proposed IDM [68]. $d$ is the distance to the leading vehicle. $d^*$ is calculated with Equation 4.9. In this equation, $d_0$ is the minimal distance wanted to the leading vehicle, which is set to 30 meters. $T$ is the desired time gap to the leading vehicle, which is set to 4 seconds. These two parameters are both set to very conservative values to ensure absolute safety and to make sure that IDM as a policy is not already optimal for scenario 2. This also ensures that this policy will never put

the agent in danger in the next timestep, which is one of the requirements as stated in Section 4.2.2, because $T = 4$ is larger than the timestep that is used in the RL system, which is 1.5 seconds. Finally, $\Delta v$ is the difference in velocity between the AV and the leading vehicle, and $dec$ is the maximum deceleration, which was set to 8.

This policy should be safe for scenario 2, especially since some of the parameters were set to very conservative values. That is why the IDM meets the requirement of always being able to propose a safe action whenever a safe action is available. All variables in this function can be obtained from the state $s$, which means that the IDM adheres to the SIP signature $\text{SIP}(s) \rightarrow a$. The discrete safe action is determined by rounding down $a_{safe}$ that follows from Equation 4.8 to the nearest discrete action.

Now that $a_{safe}$ is determined, $\rho$ needs to be calculated using $M_{SR}$. In this scenario, an action with a lower acceleration is always as safe or safer than an action with a higher acceleration. That is why the safe action range is set to $(0, a_{safe} + \rho)$ instead of $(a_{safe} - \rho, a_{safe} + \rho)$, where action 0 is the action with the lowest acceleration.

$f_{ss}$ is for scenario 2 equal to the distance from the AV to the leading vehicle $d$. $s_{critical}$ and $s_{safe}$ depend on the velocity of the vehicle. $s_{critical}$ is calculated using Equation 4.10, in which $dec$ is the maximum deceleration. This equation produces the minimal distance that is needed to come to a halt if the AV fully uses its brake. In this case, the situation is critical since any other action than using a full brake would put the AV in danger of having a collision. The requirement for $s_{critical}$ is that it is as unsafe as possible, while at least one safe action is still available. This is true for this implementation: full braking is still available as the safe action. A buffer distance $d_{buffer}$ is added to make sure that the AV keeps a bit of distance to other vehicles at all times. $s_{safe}$ is calculated using Equation 4.11, in which $acc$ is the maximum acceleration. In this equation, the buffer distance is added to the distance that the AV would drive in a timestep when accelerating fully and the distance that is needed to come to a halt after that. Whatever action the agent would take, it always has enough space to brake when it is further away from the leading vehicle than $s_{safe}$, since it always has enough distance to brake after executing every possible action. This means that every action is safe in $s_{safe}$, which is the requirement. Now $\rho$ can simply be calculated using Equation 4.6, in which $f_{ss}(s)$ is equal to $d$, the distance of the AV to the leading vehicle.

$$s_{critical} = d_{buffer} + \frac{0.5v^2}{-dec} \tag{4.10}$$

$$s_{safe} = d_{buffer} + T \cdot (v + 0.5 \cdot acc \cdot T) + \frac{0.5(v + acc \cdot T)^2}{-dec} \tag{4.11}$$

### 4.2.5. Related techniques

The SIPS is a novel technique that is closest related to Model Predictive Shielding [43], which is a shielding technique in which a safe policy is used to guide the agent back to a safe state if it enters a state where the agent cannot guarantee safety. The difference is that the SIPS contains the SafetyRangeModel, which determines how much the agent's action can deviate from the SIP action, based on how risky the current situation is. This is the main novel idea of the SIPS. In Model Predictive Shielding, the agent must always directly follow the safe policy whenever it ends up in a dangerous situation.

Other than Model Predictive Shielding, SIPS also touches the field of Transfer Learning (TL) [67]. TL aims to improve the learning process of a new task by using knowledge from a related task. In our case, the knowledge from an existing model, which aims to stay safe in the environment, is used to improve learning to be efficient while staying safe. The final policy $\pi_K$ after training an agent when using SIPS should be an improvement of the SIP in terms of efficiency, while achieving a similar performance in terms of safety.

Finally, the SIPS has some similarities to the technique of Safe Policy Improvement (SPI) [66]. SPI aims to learn a new policy based on an initial policy. This initial policy has interacted with the environment, creating a data set of experiences. The new policy is learned by training on these experiences. SPI provides probabilistic guarantees about improving the performance of the initial policy. The difference with the SIPS is that the new policy is created based on experiences gathered by the initial policy, while the SIPS only uses the initial policy directly in dangerous situations. In safe situations, the RL algorithm can interact with the environment itself without the SIPS interfering. Even in slightly unsafe situations, the agent can deviate from the proposed action by the initial policy when using the SIPS. Both of the techniques should be able to improve on the initial policy, although no guarantees about this are made for the SIPS.

## 4.3. Shield-based learning

Every action that the agent wants to take during training is checked by the SCS or the SIPS if these are employed. If the preferred action $a_1$ is overruled, it will not be executed and therefore no experience $(s, a_1, r, s')$ will be put in the replay memory. This means that the agent will not learn that action $a_1$ is unsafe. If at some point the agent would no longer have a shield employed, it would not have gathered the knowledge to be able to learn to avoid unsafe situations. This might result in an unsafe final policy. Other research deals with this by keeping the shield employed during execution [6, 33].

In this research, two different shield-based learning methods are proposed and tested, methods that enable the agent to learn to avoid actions that are overruled by a shield. For the first method, fabricated experiences are produced whenever an action is overruled. For the second method, an alternative loss function is proposed. Both of these methods are built around the DDQN that is used in this research. They would also work with a regular Deep Q-Network. The methods are described in this section. They work for both shields.

### 4.3.1. Fabricated Experiences

For the first method, fabricated experiences are produced. Whenever the preferred action $a_1$ of the agent is overruled by the shield, it still gets added to the replay memory, even though the action is not executed and $s'$ is not known. An experience with action $a_1$ is then fabricated. In this fabricated experience $(s, a_1, r, s')$, $s$ is the actual current state, $a_1$ is the preferred and overruled action, $r$ is a negative reward that is equal to the minimal possible reward and $s'$ is equal to $s$. In our case, $r$ is equal to the punishment of a collision, which is the minimal reward as shown in Section 3.5.3. The fabricated experience is marked as *done*, which usually means that it was the last experience of an episode. This influences how the experience is used to update the Q-network (Algorithm 2, Section 2.2.2). Notice that only $s$, $a_1$ and $r$ are used to update the Q-network when an experience is marked as *done* and $s'$, which does not exist since $a_1$ has not been executed, is not used for the update. By using these fabricated experiences, it is possible for the Q-network to learn from negative rewards that have been assigned to actions that were overruled by the shield. The agent can now learn that executing the overruled action $a_1$ for state $s$ yields a negative reward equal to the negative reward for a collision.

The creation of such fabricated experiences is done by the agent. The agent compares the action it sent to the environment with the action that was executed. If these are not the same, which means that its preferred action was overruled, the agent adds two experiences to the replay buffer instead of one: both $(s, a, r, s')$ and $(s, a_1, -1, s)$ in which *done* is set to true. By adding these experiences to the replay buffer, the agent can learn to avoid unsafe actions in the same way that it would learn to avoid collisions in a regular RL setting without a shield. The process of shielding with fabricated experiences is shown in Figure 4.4. The arrow on the right shows that the shield sends the chosen safe action $a_{safe}$ to the agent, which can then produce a fabricated experience if $a_{safe}$ is not equal to its preferred action $a_1$.

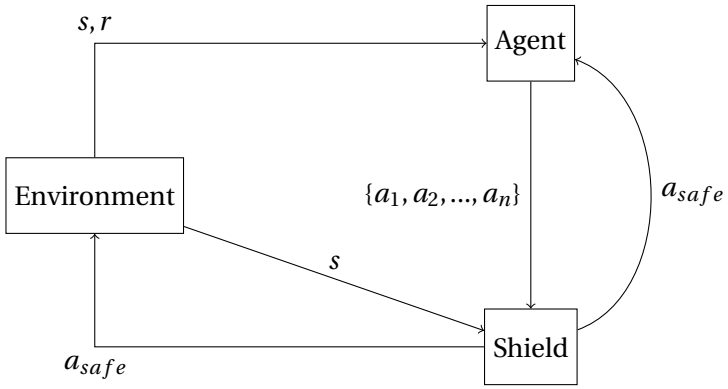



Figure 4.4: Schematic representation of a shield during training when fabricated experiences are used. At every step, the agent creates a sorted list of actions in descending order of preference based on $s$. The shield picks the first action $a_{safe}$ that is safe from this list. The chosen action is sent to the agent as well, which enables the agent to fabricate an experience when necessary.

### 4.3.2. Alternative loss function

For the second method, an alternative loss function is adopted instead of fabricating experiences. This alternative loss function should enable the agent to learn to not propose actions that would be overruled by a shield. The advantage of only altering the loss function is that it is relatively clean, since only the loss function needs to be changed while no changes need to be made to the RL process. This separates the rewards from the constraints. The constraints are now handled within the loss function, while the rewards are only used for learning a well performing policy.

In the regular DDQN, Mean Squared Error (MSE) is used as the loss function, as will be described in Section 5.2.1. In this method, the MSE is extended with a second term to enable the agent to learn to avoid actions that would be overruled by a shield. The general idea is that the agent should learn that it should never propose an action that would be overruled by a shield. In other words, the Q-network should never assign the highest Q-values to actions that would be overruled by a shield. Therefore the loss function punishes Q-values that are relatively high for actions that would be overruled. The alternative loss function is shown in Equation 4.12.

$$Loss(y, \hat{y}, O) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \cdot \sum_{i=1}^{n}\left(O_i \cdot \frac{e^{\beta \hat{y}_i}}{\sum_{j=1}^{n} e^{\beta \hat{y}_j}}\right) \tag{4.12}$$

In Equation 4.12, $y$ is a list of the actual Q-values of the actions, $\hat{y}$ is a list of estimated Q-values by the network and $O$ is a list of binary values, indicating which actions the shield would overrule. All three lists have a length $n$, which is equal to the number of actions. The first part of the equation is the regular MSE. The second part is meant to enable the agent to learn not to assign the highest Q-values to actions that would be overruled. In this second part, $\lambda$ and $\beta$ are hyperparameters which need to be set. $O_i$ is 1 when the shield would overrule action $i$ and 0 otherwise. $\hat{y}_i$ is the predicted Q-value of action $i$ by the Q-network. Basically, the second part of the loss function produces a sum of SoftMax values of every action that would be overruled. If the Q-values of these actions are relatively high in comparison to the actions that would not be overruled, this sum gets quite high which results in a higher loss. This means that the network should assign relatively low Q-values to actions that would be overruled by the shield to minimize the loss. This enables the agent to learn to not propose unsafe actions that would be overruled by the shield.

# 5

# Experimental Setup

Now that the problem and the proposed shields have been discussed, the setup that is used check whether the shields address the research question can be described. This chapter contains all information about the experiments that are carried out to check this. First, the simulator used to execute the experiments is discussed. Then, the specifics of the Reinforcement Learning (RL) system that is used for the experiments are presented. The Deep Neural-Network, hyperparameters and training process are described. At the end of the chapter, the setup of the experiments is discussed.

## 5.1. Simulator

To carry out experiments, an Autonomous Vehicle (AV) is needed on which the different proposed systems can be loaded. Unfortunately, there is no real life AV available for this research. Therefore an AV in a simulator is used to evaluate the proposed systems. It would be possible to build a new simple simulator for the scenarios that are proposed in this research, but it is preferred to use an existing driving simulator. When using an existing one, it is ensured that it is not possible to manipulate the simulator in any way to make the systems perform better.

Multiple driving simulators that can be used for AV research are available, for example SCANeR Studio [22], AirSim [61], OpenDS [47] and CARLA [21]. From these options, CARLA was chosen to use in this research for multiple reasons. First of all, it is one of the most popular recent driving simulators and it is used in much current AV research [14, 28, 38], specifically for RL approaches as well. Furthermore, CARLA offers a Python API, which is the programming language in which the RL system of this research is implemented. Additionally, it is open-source, which is an advantage because of transparency. Finally, it is possible to run co-simulations between CARLA and PTV Vissim, which is the simulator that TNO mainly uses. In such a co-simulation, some of the vehicles are simulated within CARLA and some of the vehicles are simulated within PTV Vissim, while all vehicles are shown in both simulators. By choosing CARLA as the simulator in this research, it is possible for the created systems to control cars in PTV Vissim as well by using these co-simulations.

CARLA is a simulator that has been specifically developed for training and validation of AVs. It is grounded on Unreal Engine 4 [59] and it uses the OpenDRIVE [24] road network framework. When running a simulation, a CARLA instance needs to be open. Through Python, it is possible to specify settings like the map or the weather for the current CARLA instance. Afterwards, vehicles, pedestrians and cyclists can be spawned. There is an autopilot mode available to let most of the traffic drive by themselves.

In this research, all road users are controlled by the autopilot except one, the AV. The AV can be controlled using five commands: steering, throttle, brake, hand brake and reverse gear. Only the first three of the commands are used for the experiments in this thesis. For steering, CARLA requires

a value between -1 and 1, where -1 is a full left steer, 0 is no steering and 1 is a full right steer. Throttle and brake both require a value between 0 and 1 where 0 means no throttle or brake and 1 means full throttle or brake. By setting values for these three variables at every step in the simulation, the AV drives around in the CARLA environment.

CARLA provides multiple sensors through which the AV can sense its environment. Examples of these sensors are camera's, radar and LIDAR. Other than that, all available information from the simulation, like locations and velocities of vehicles and other objects, can be accessed through the API and used by the AV. As stated in Chapter 1, the systems that are proposed in this thesis assume that they receive perfect information from the environment. Therefore the AV does not use any of the sensors, but directly uses simulation data. This includes the velocity and location of the AV and of other vehicles.

A downside to CARLA is that it is quite slow. It is not possible to run many simulations within a short time, because of the fact that simulations run in real time. This might have a negative impact on the learning process: RL algorithms often require millions of samples before being able to learn something. It would take a year to gather one million samples in CARLA when running one simulation. Although this could impact the experiments in this research, only being able to run simulations in real time is realistic. When employing the systems created in this research on real life AVs, they can also only run in real time. Therefore these systems should be able to deal with that and learn a good policy while not having millions of samples available. Only if a working system can be created which learns a good and safe policy in a real time simulator, it would be possible to employ the same system to real world vehicles.

### 5.1.1. Scenarios

As stated earlier in Section 3.4, two scenarios are created in which the AV needs to learn how to behave: one scenario in which the AV needs to drive straight without leaving the road and one scenario in which the AV needs to learn to drive safely with other vehicles in one lane. By using two scenarios to evaluate the Safe RL system, it should be possible to draw more general conclusions about the performance of the system than when only one scenario would be used.

The scenarios are implemented in CARLA. When a simulation is started, the map is loaded. In scenario 2, other traffic is loaded as well. Then, for every episode, the AV is placed at the start of the road on which it needs to learn its task. The control of the AV is then given to the RL system until the AV reaches the end of the road, leaves the road, has a collision with another vehicle or runs out of the episode time limit, which is set to thirty seconds. The AV is then removed from the road and spawned again at the start of the road. This process repeats itself until the final episode.

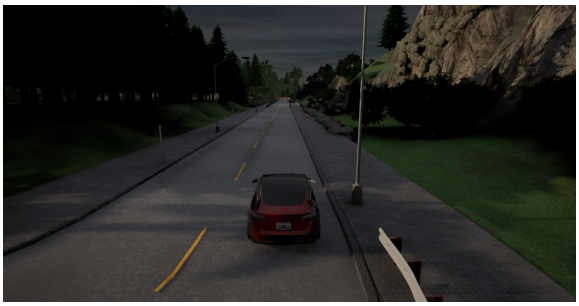
Screenshots of the two scenarios are shown in Figure 5.1. Figures 5.1a and 5.1c show a behind view and a top view of the AV in scenario 1. Figures 5.1b and 5.1d show a behind view and a top view of the AV in scenario 2.

### 5.1.2. Throttle to acceleration function

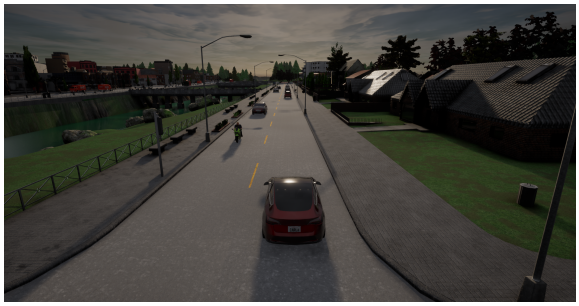
In CARLA, the acceleration and deceleration of a vehicle are controlled by setting a throttle and a brake variable to a value between 0 and 1. It is not possible to directly set the acceleration of a vehicle. Unfortunately, for the shields, it is necessary to know the exact acceleration of the vehicle when executing a certain action. Chapter 4 showed that it is necessary for the shields to know the exact acceleration or deceleration per action to ensure safety. Therefore a function is needed that maps the throttle and brake actions to acceleration values. In CARLA, this function differs per vehicle used.

Fortunately, an intern at TNO has measured the acceleration as a function of the current velocity and the throttle or brake value of the Tesla Model3 car[1], which is used in this thesis. For every velocity below 27 m/s, the acceleration was measured for every throttle and brake value. Figure 5.2

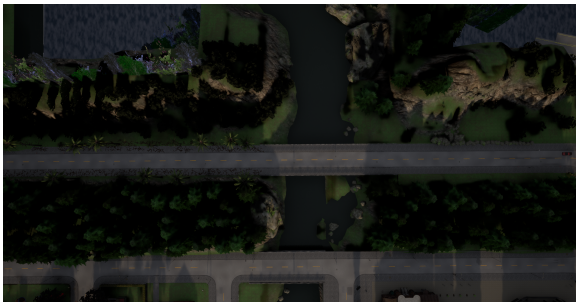
[1] Paul Netto. Carla documentation, *TNO*, Dec 2020



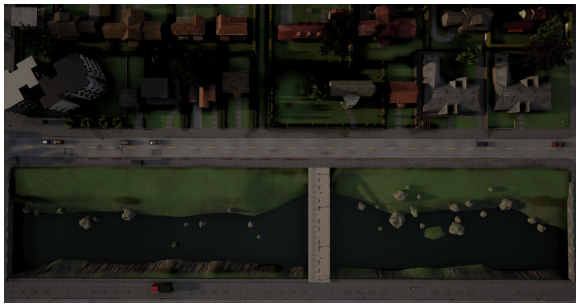
(a) View from behind in scenario 1: one lane without traffic.



(b) View from behind in scenario 2: one lane with traffic.



(c) View from top in scenario 1: one lane without traffic.



(d) View from top in scenario 2: one lane with traffic.

Figure 5.1: Screenshots from the CARLA scenarios in which the AV needs to learn how to drive. The red Tesla Model 3 is the AV.

shows a plot of these measurements of the longitudinal actions used in this thesis, which were listed in Section 3.5.2.

Based on the measurements that are shown in Figure 5.2, the acceleration of the vehicle for the current velocity and action can be obtained. The lines in the figure are created by plotting measurements. From the action that is chosen, the two closest measurements above and below the current velocity are taken. The highest acceleration from those two measurements is used. This ensures that the shields will only slightly overestimate accelerations, which is safer than underestimating accel-

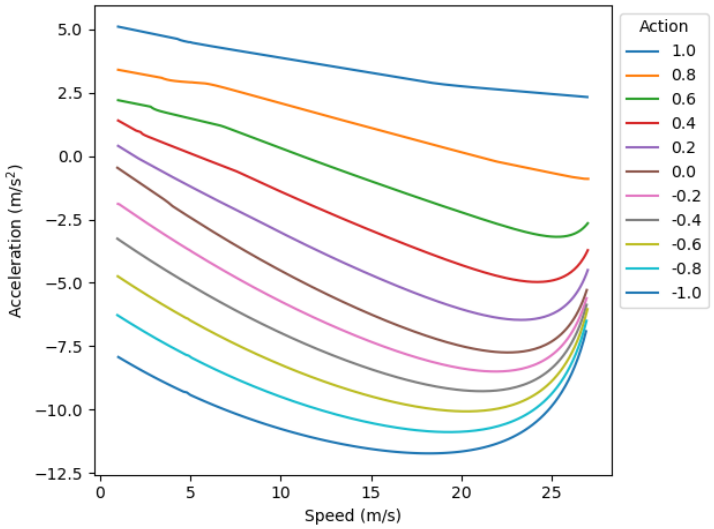



Figure 5.2: A mapping from velocities to accelerations for the selected range of throttle and brake actions in CARLA for the Tesla Model3 vehicle that is used in this research. The positive values are throttle actions, the negative values are brake actions.

erations, which could cause collisions. Every action consists of more than thousand measurements though, so these two measurements are always very close to each other.

## 5.2. Reinforcement Learning System

As stated in Section 3.5.4, the Double Deep Q-Network (DDQN) is used as the RL technique in this research. This section will describe the specific implementation of the DDQN and its training process. The DDQN is designed in such a way that it is able to learn a good policy on both scenarios, which were described in Section 3.4.

### 5.2.1. Deep Neural Network

A simple neural network is used as the Q-network. Its design is based on Keras' own Deep Q-learning implementation[2]. Keras [17] is the Python package that was used to implement the neural network. For simplicity and to prevent overfitting, the Deep Q-Network that is used in this thesis contains fewer layers than Keras' network. To be precise, it contains three dense layers. For a more complex task, it might be necessary to add an extra layer. The neural network is visualized in Figure 5.3.

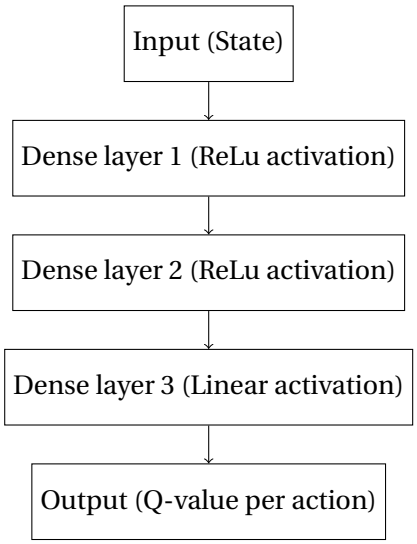



Figure 5.3: Schematic representation of the layers of the neural network used in this thesis.

The first dense layer takes the state as input, after which the second dense layer takes the output of the first layer as input. Both have a ReLu activation function. Finally, the last dense layer takes the output of the second dense layer as input and outputs estimated Q-values for each action. This layer has a linear activation function. For the network, the Adam optimizer [37] is used, which is one of the most popular optimizers. Mean squared error, as shown in Equation 5.1, is used as the loss function. It is calculated by taking the mean of the squared differences between the estimated values by the neural network $\hat{y}$ and the actual values $y$.

$$MSE(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{5.1}$$

The layers, activation functions, optimizer and loss function should result in a neural network that is able to effectively learn good behavior in the AV scenarios. Since it takes around ten hours to train and evaluate one setting of the network for scenario 2, it was infeasible to test many different network settings. Only a couple of experiments have been done to check the optimal number of layers. The number of layers did not strongly influence the performance.

[2] `https://keras.io/examples/rl/deep_q_network_breakout/`

### 5.2.2. Hyperparameters

Deep RL is known to have many hyperparameters: parameters that are used to control the learning process. They need to be set before the learning starts, in opposite to parameters that are learned during training. The values of these hyperparameters can have a huge impact on the performance of the learning algorithm [18]. Therefore it is necessary to tune these parameters to achieve a good performance.

Many techniques exist to tune hyperparameters, like random sampling, particle swarm optimization, genetic algorithms, bayesian models and others [18]. All of them need many objective function evaluations, some more than others. Unfortunately, it takes around ten hours to reliably train the system created in this thesis. Therefore it is infeasible to do a lot of tuning. This does not matter too much for the goal of this research though, since the aim is to check the influence that Safe RL techniques have on the RL process, not to create the best possible driver model using RL. It is still possible to show this influence when the RL algorithm does not perform optimally.

There has been some hyperparameter tuning though. For the most important hyperparameters, a range of values was tested one by one while keeping other hyperparameters constant to check the influence of different values for that hyperparameter. They were tested on scenario 2 by using the regular DDQN without any safety techniques. The hyperparameter settings that yielded the highest return during training were chosen. This tuning resulted in an RL system that is consistently able to learn its tasks. It is likely though that the performance of the RL system will be increased when it is possible to do more extensive hyperparameter tuning. The values of the most important hyperparameters that were used, as well as the ranges of values that were tested, are listed in Table 5.1.

| | |
|---|---|
| Minimal replay memory size | **200** |
| Replay memory size | **50000** |
| Minibatch size | [**16**, 32, 64, 128] |
| Episodes between target network updates | [**10**, 20, 50] |
| Learning rate | [0.000025, **0.00025**, 0.0025] |
| Discount factor | [**0.95**, 0.99, 0.999] |
| Size DQN layer 1 | [**32**, 64, 128] |
| Size DQN layer 2 | [32, **64**, 128] |

Table 5.1: The chosen values of the most important hyperparameters are shown in bold within the range of values that were tested.

### 5.2.3. Training

The training process consists of two parts: running simulations in an environment and training a Q-network based on the experiences gathered in that simulation. This is done in parallel on two different threads. On the first thread, the agent interacts with the environment, and on the second threat, the Q-network is trained. This goes on for 200 or 1000 episodes for scenario 1 and scenario 2 respectively. One episode lasts a maximum of 30 seconds, but it ends earlier if the agent has a collisions or reaches its destination. There are 1.5 seconds between each step in an episode. After all episodes, the trained Q-network is saved and can be used as a driver model to control an AV within the scenario it is trained on. For scenario 2, three models are saved: one after 800 episodes, one after 900 episodes and one after 1000 episodes. The model which performs best is used as the final driver model. Sometimes the Q-network receives a couple of bad minibatches in a row to train on, which lowers its performance. For example, it could receive a couple of batches without any experiences of collisions, which might make it forget how to avoid collisions. If only one model is saved during training, this could cause the final model to coincidentally not perform well. This problem is solved by saving multiple models during training.

The policy during training is an $\epsilon$-greedy policy, as described in Section 2.1.3. At the start, every

action taken is random. During the process, the chance of taking a random action linearly decreases until 65% of training is done. Then only greedy actions are taken. The value of $\epsilon$ for each episode is shown in Equation 5.2. In this equation, $K$ is the total number of episodes and $k$ is the current episode. At every step during an episode, a number is drawn randomly from a uniform distribution between 0 and 1. Whenever this number is below $\epsilon$, an exploration action is chosen. Otherwise, the exploitation action is executed. The experiences from these actions are saved to the replay buffer.

$$\epsilon = \begin{cases} 1 - \frac{k}{K \cdot 0.65} & \text{if } \frac{k}{K} < 0.65 \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

Meanwhile, the Deep Q-Network is trained on the experiences from the replay buffer. Every ten episodes, the target network that is used for training is updated. The training only stops when all episodes have been simulated. The whole training process of a Deep Q-Network was shown previously in Section 2.2.2, Algorithm 2.

## 5.3. Setup

One experiment consists of two parts: training and evaluating an RL agent. Training is done over multiple episodes. During these episodes, the RL agent trains on experiences gathered, parallel to executing actions. After training, the resulting policy is saved. More information about this process can be found in Chapter 2.

For scenario 1, a simulation takes 200 episodes. Scenario 2 is more complex and therefore 1000 episodes are simulated. One episode lasts for a maximum of thirty seconds, which is equal to twenty steps. If the agent has a collision or reaches its destination before this time, the episode is terminated earlier. After every episode, the evaluation criteria (return, *col* and *sp*, see Section 3.2) are collected from the environment. These are then plotted over the episodes, so that the development of the KPI's during training can be shown. For the shields, also the percentage of steps for which the shield overrules an action is plotted in the same manner.

To get reliable results, five simulations are run for one setting of the algorithm. The means and standard deviations of the KPI's during training are plotted. Furthermore, the five resulting policies are all simulated for 100 episodes on their scenario during which the KPI's *col* and *sp* are saved. In a tabular form, the means and standard deviations of the KPI's during the evaluation of the final policies are shown. Summarized, one experiment consists of training and evaluating five RL agents and plotting and showing the means and standard deviations of the evaluation criteria during training and execution.

Except for checking the evaluation criteria, it is also interesting to check how the final learned policies behave in their environments. Therefore the final policies that yield the highest return out of the five learned policies are once again simulated, just for one episode on a specific task, to check their behavior. This is a qualitative analysis, for which custom tasks are created, on which the behavior of the agent is evaluated by interpreting the resulting plots. For scenario 1, this task is equal to the original scenario. For scenario 2, three different tasks are created. In the first task, there is a vehicle standing still 140 meters ahead of the AV. In the second task, there is a vehicle which drives with a constant velocity of 6 m/s in front of the AV. In the third task, the vehicle in front of the AV starts the episode with a constant velocity of 12 m/s, but brakes after ten seconds. For all tasks in both scenarios, the velocity of the AV during the task is plotted. For scenario 2, the velocity of the other vehicle and the distance between the vehicles is plotted as well. This enables us to check the actual behavior of the AV after learning a policy, which is relevant when the AV needs to be able to drive in the real world.

All the experiments are carried out on the same machine for consistency. The specifications of the machine that is used and the versions of the installed software are shown in Table 5.2. The code

that is used for the experiments will be made available on GitHub[3].

| | |
|---|---|
| Operating system | Microsoft Windows Pro 10 |
| Central Processing Unit | Intel Core i9-7900X CPU @ 3.30GHz, 10 cores |
| RAM size | 32.0GB |
| Graphics Processing Unit | 2x NVIDIA GeForce GTX 1080 Ti |
| | |
| Python | 3.7.9 |
| NumPy | 1.18.3 |
| Keras | 2.2.4 |
| Scipy | 1.4.1 |
| Tensorflow | 1.15.4 |
| Pygame | 1.9.6 |
| CARLA | 0.9.9 |

Table 5.2: Specifications of the machine and software versions used for the experiments.

Now that the general setup is explained, the specific experiments that will be carried out can be described. To answer the fifth research question, the two shields need to be compared with the regular DDQN to be able to analyze their effects on the learning process. For the main experiments, the three systems - DDQN, SCS and SIPS - will all be trained and tested on both scenarios. For the SCS and the SIPS, fabricated experiences are used as the shield-based learning technique. As stated, five agents will be trained for each system and scenario.

Furthermore, an experiment will be carried out to compare the different types of shield-based learning, which were described in Section 4.3. This experiment is carried out on scenario 2, using the SCS. The two methods, fabricated experiences and the alternative loss function, are compared to not using a shield-based learning method.

[3] https://github.com/AlgTUDelft/Carla_Safe_RL

# 6

# Results

This chapter will first list the expectations about the results of the experiments that were described in Section 5.3. After that, the results of the experiments are presented, both for the comparison of the shields and the comparison of the shield-based learning methods. This chapter mainly aims to answer the fifth subquestion (Section 1.4), which relates to how the different shields compare to the baseline Double Deep Q-Network (DDQN) in terms of performance during training and execution.

## 6.1. Expectations

Chapter 3 showed that the main goal of this thesis is to build a shielding system which enables an RL agent to train safely, while still achieving a satisfying performance in terms of efficiency. This means that the main interest in the results is whether the number of collisions of the shielding systems is zero, and whether the average velocity of the final policies when using the shields is at least close to the average velocity of the final policy when using the regular DDQN. This section will discuss the expected results per evaluation criterion: the return during training, number of collisions per kilometer, average velocity in meters per second and percentage overruled actions by the shields.

The return is only used to check whether the agent is able to learn a task during training. The return should always increase over the training episodes, otherwise the agent learns bad behavior instead of good behavior. Since the DDQN and both the SCS and the SIPS should be able to learn efficient behavior, the return should increase during training for all systems for both scenarios. At the end of training, it should stop increasing, meaning that the agent has learned a good final policy. As mentioned earlier, the return is only measured during training and not during execution, because it does not directly say anything about the performance of a policy, only about whether the agent is able to learn during training.

The number of collisions during training, which indicates safety, should be zero for both the SCS and the SIPS on both scenarios during the whole training process and during execution of the final policy. If it is not zero, it indicates that the AV still has collisions, which means that the AV could not safely be trained in real life without causing dangerous situations. For the DDQN, it is expected that the number of collisions decreases during training. In scenario 1, the number of collisions should decrease to zero, because the task is simple enough to learn a completely safe policy and an increase of safety does not decrease efficiency in this task. Since scenario 2 is more complex, the number of collisions might not decrease to exactly zero, but the agent should learn to avoid almost all unsafe situations.

The average velocity indicates the efficiency. It is expected that all systems on both scenarios increase the average velocity during training. For scenario 1, all three systems should behave similarly in terms of efficiency. Once they learn to not leave the road, all that needs to be learned is to

fully use the throttle, which they should be able to learn. For scenario 2, it is expected that the SCS and the SIPS are a bit more conservative than the regular DDQN. The shields might overrule more high throttle actions than strictly necessary to drive safe, so the DDQN will probably have a higher efficiency both during training and during execution. The difference should not be too high though.

Finally, the percentage of overruled actions is checked during training. Clearly, this percentage is always zero for the regular DDQN, since that system does not contain a shield that overrules actions. For the SCS and the SIPS, it is expected that this percentage decreases during training in a similar manner as the number of collisions decreases during training when using the regular DDQN. For scenario 1, the percentage should decrease to zero, and for scenario 2 it should get at least close to zero.

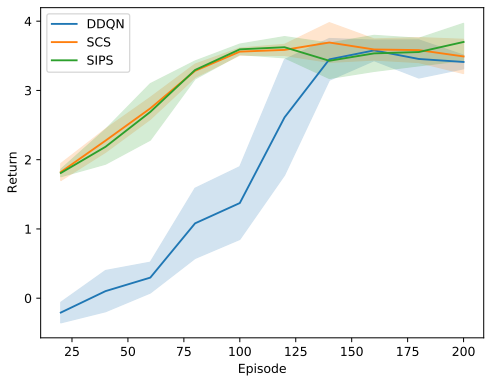
After the comparison, some experiments are shown that test the two shield-based learning techniques and compare them to a system where no shield-based learning is used. It is expected that both of the techniques result in zero collisions during execution, since they are designed to let the agent learn to avoid collisions. When no shield-based learning is used during training and no shield is employed during execution, there will be collisions. In terms of efficiency, it is expected that all techniques are on the same level. The experiments will show whether one of the techniques is better in terms of efficiency than the others.

## 6.2. Results

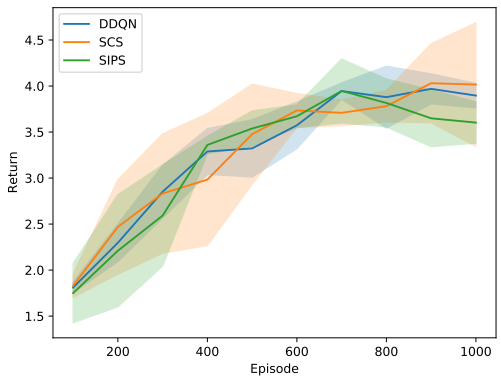
As stated before, two experiments were carried out: one that compares the two shields with the baseline DDQN algorithm and one that tests different shield-based learning methods. The two subsections within this section describe these two experiments. For all experiments, figures are presented that show the development of the evaluation criteria during training. Furthermore, the performance of the final learned policies is shown in tables. The results are analyzed by comparing them to the expectations from Section 6.1 and the goals from Section 3.3.

### 6.2.1. Comparison

This subsection shows the results of the comparison between the two shielding systems, the SCS and the SIPS, and the baseline RL system, the DDQN. The results of the evaluation criteria during training can be found in Figures 6.1, 6.2, 6.3 and 6.4. Both the means and standard deviations over five training sessions are shown.



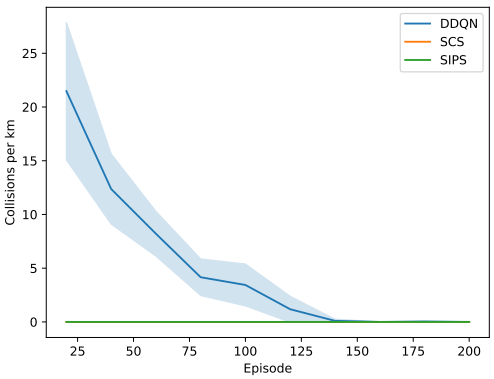

(a) The return (sum of rewards) over episodes when training an RL agent using the regular DDQN, the SCS and the SIPS on scenario 1.



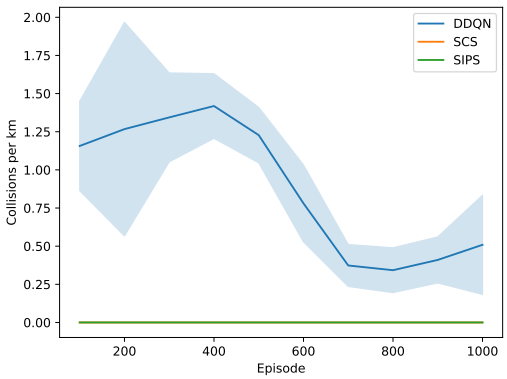

(b) The return (sum of rewards) over episodes when training an RL agent using the regular DDQN, the SCS and the SIPS on scenario 2.

Figure 6.1: Plots of the return during training of an RL agent using the regular DDQN, the SCS and the SIPS on scenario 1 and 2. Five experiments were run per scenario, of which the means and standard deviations of the return are plotted.

In Figures 6.1a and 6.1b, the return (sum of rewards) over the training episodes is shown for scenario 1 and scenario 2 respectively. The first thing that becomes clear is that all systems on both scenarios are able to learn a better policy during training, because the return increases over the episodes. For scenario 1, the SCS and the SIPS behave very similar. This was expected, since they overrule exactly the same actions. The only difference is the way how it is determined which actions are overruled. The regular DDQN starts much lower in terms of return than the SCS and the SIPS. This can be explained by the fact that it has a lot more collisions at the start, which yield a negative reward. When it learns to avoid these collisions and when it does no longer take random actions, it is shown that the return gets to the same level as the return of the SCS and the SIPS. In the end, the return for all three is on the same level. For scenario 2, all three systems behave very similar and based on the return figure, no clear differences can be seen. This means that employing a shield on scenario 2 does not clearly affect the development of the return during training. All are able to approximately double their return when comparing the policies during the end of training with the random policy at the start of training.




(a) The number of collisions per km over episodes when training an RL agent using the regular DDQN, the SCS and the SIPS on scenario 1.

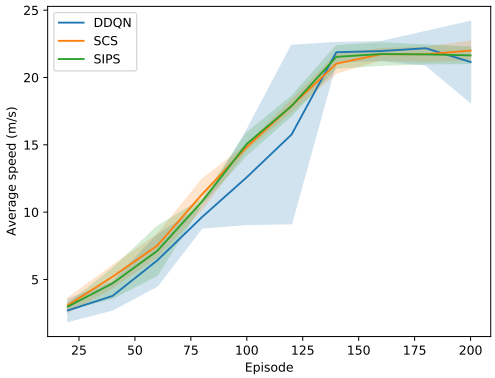



(b) The number of collisions per km over episodes when training an RL agent using the regular DDQN, the SCS and the SIPS on scenario 2.

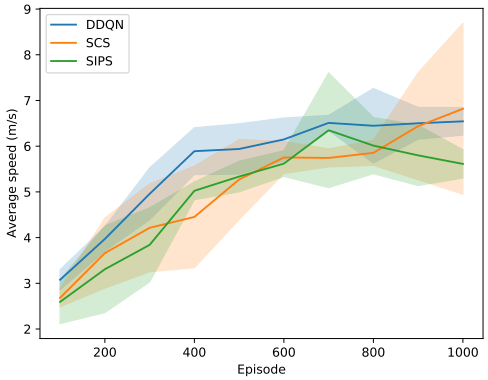
Figure 6.2: Plots of the collisions per km during training of an RL agent using the regular DDQN, the SCS and the SIPS on scenario 1 and 2. Five experiments were run per scenario, of which the means and standard deviations of the collisions per km are plotted.

Figures 6.2a and 6.2b show the collisions per kilometer driven during training for scenarios 1 and 2 respectively. Both the SCS and the SIPS have zero collisions in scenario 1 and 2. The shields prevent the AV from ever leaving the road or hitting another vehicle. The regular DDQN does have collisions. The number of collisions is very high at the start of training in scenario 1, but the agent learns to avoid collisions during the training process. At the end of training, it has learned to completely avoid any collisions. The regular DDQN is also able to decrease the number of collisions in scenario 2, but it does not quite learn to completely avoid any collisions in the end. Longer training or a different reward function could possible solve this issue, but the second options might cause more conservative behavior of the agent.

The average velocity in meters per second during training is shown in Figures 6.3a and 6.3b for scenarios 1 and 2 respectively. For scenario 1, there is no significant difference between the SCS and the SIPS, which both have very small standard deviations as well, indicating that the training process is stable. The DDQN seems to be a bit less stable. All three systems learn to increase their velocity during training, going from 4 m/s to over 20 m/s within 150 episodes. In scenario 2, the DDQN seems to have a higher velocity throughout the training process than the two shielding systems, which again behave very similar. This can be caused by the fact that the shielding systems drive



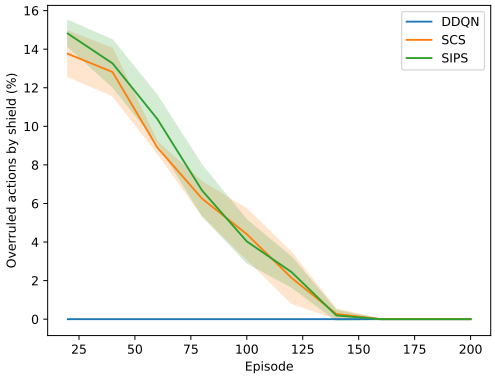

(a) The average velocity (m/s) over episodes when training an RL agent using the regular DDQN, the SCS and the SIPS on scenario 1.



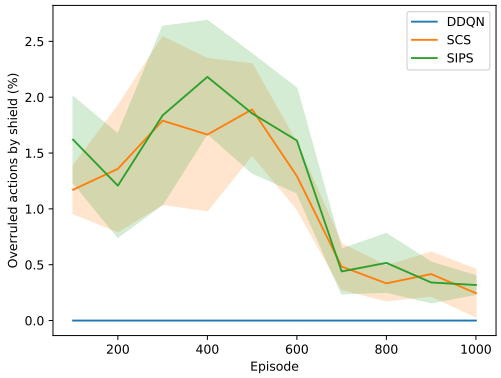

(b) The average velocity (m/s) over episodes when training an RL agent using the regular DDQN, the SCS and the SIPS on scenario 2.

Figure 6.3: Plots of the average velocity during training of an RL agent using the regular DDQN, the SCS and the SIPS on scenario 1 and 2. Five experiments were run per scenario, of which the means and standard deviations of the average velocity are plotted.

a bit more conservative, since actions that are a bit unsafe might already be overruled. This could decrease the average velocity. The difference is not that large though, the standard deviations of the systems overlap.




(a) The percentage of overruled actions over episodes when training an RL agent using the regular DDQN, the SCS and the SIPS on scenario 1.




(b) The percentage of overruled actions over episodes when training an RL agent using the regular DDQN, the SCS and the SIPS on scenario 2.

Figure 6.4: Plots of the percentage of overruled actions during training of an RL agent using the regular DDQN, the SCS and the SIPS on scenario 1 and 2. Five experiments were run per scenario, of which the means and standard deviations of the percentage of overruled actions are plotted.

Finally, the percentage of overruled actions during training is shown in Figures 6.4a and 6.4b for scenarios 1 and 2. It is evident that the number of overruled actions for the regular DDQN is always zero, since the regular DDQN does not contain a shield that overrules actions. For scenario 1, the SCS and the SIPS agents learn to never propose actions that would be overruled by the shield, resulting in no overruled actions at the end of training. In scenario 2, these agents are also able to decrease the number of overruled actions during training, but the shield is still needed at the end of training. Until approximately 400 out of the 1000 episodes, the percentage of overruled actions actually increases. This is likely caused by the fact that the agents learn to propose actions that result

in a higher velocity at the start of training since this yields a higher reward. As a consequence, more unsafe actions are proposed which means that the shields have to intervene more often. In the end, the agent learns to increase velocity while proposing less unsafe actions though, indicating that it is able to learn its task. Interestingly, the shapes of the overruled action plots of the SCS and the SIPS are very similar to the shapes of the collision plots of the DDQN. This makes sense: the underlying policies of the systems are equal. The only difference is that the shields overrule actions that would normally cause collisions. Whenever collisions are shown for the DDQN in the collision figures, it makes sense that at the same time the shields overrule actions for the SCS and the SIPS, which is shown in the overrule figures.

| | **Collisions per km** | **Average velocity (m/s)** |
|---|---|---|
| *Scenario 1* | | |
| **DDQN** | 0.0 ± 0.0 | 22.6 ± 0.8 |
| **SCS** | 0.0 ± 0.0 | 22.4 ± 1.1 |
| **SIPS** | 0.0 ± 0.0 | 22.9 ± 0.9 |
| SIP | 0.0 ± 0.0 | 10.2 ± 0.1 |
| *Scenario 2* | | |
| **DDQN** | 0.2 ± 0.2 | 5.5 ± 0.5 |
| **SCS** | 0.0 ± 0.0 | 6.0 ± 0.4 |
| **SIPS** | 0.0 ± 0.0 | 6.0 ± 0.1 |
| SIP | 0.0 ± 0.0 | 5.2 ± 0.1 |

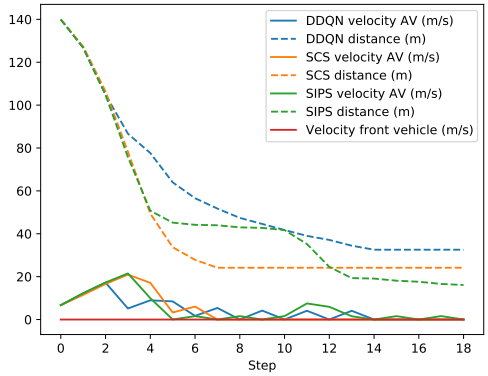
Table 6.1: The average *col* and *sp* and standard deviations of five final policies that were simulated for 100 episodes on the scenario that they were trained on.

The performances of the final policies after training for the regular DDQN, the SCS and the SIPS are shown in Table 6.1. The performances of the Safe Initial Policy (SIP), which is used by the SIPS, are also shown. For scenario 1, both KPI's are very similar for the policies following from all three systems. None of the executed policies results in any collisions and the average velocities are very close as well. It seems that all three systems produce a policy of the same quality when checking these KPI's. The SIPS is clearly able to improve on the SIP, more than doubling its velocity while both having zero collisions.

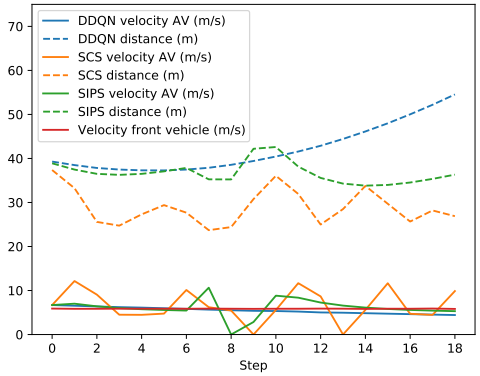
For scenario 2, there are some slight differences between the systems. The thing that stands out most is that the policies learned through regular RL can cause collisions, although this does not happen often. The policies that were created from systems that use shielding do not have any collisions during execution, although the shields are not employed anymore. This indicates that it is easier to learn to avoid collisions when shields are used than when the agent actually experiences collisions during training. This can be caused by the fact that the shields overrule more actions than only the actions that directly cause a collision. Because of this, the agent might learn to drive a bit safer than when it only gets punished for collisions. Moreover, it happens more often that the shield overrules an action when using the SCS or the SIPS than that the agent has a collision when using the regular DDQN. This means that there are more unsafe experiences for the agent to train on when using one of the shields. In terms of efficiency, the average velocity of the SCS and the SIPS is a bit higher than the average velocity of the regular DDQN, but the difference is not large enough to draw conclusions from this. The SIPS is again able to improve on the SIP that is used for it. It has a higher velocity, while having the same number of collisions.

The behavior of the final policies of the DDQN, SCS and SIPS on the three tasks within scenario 2 that were described in Section 5.3 is shown in Figure 6.5. The scenario 1 plot was omitted since it did not show anything substantial: all three systems use full throttle during execution, which is the optimal policy in every episode.

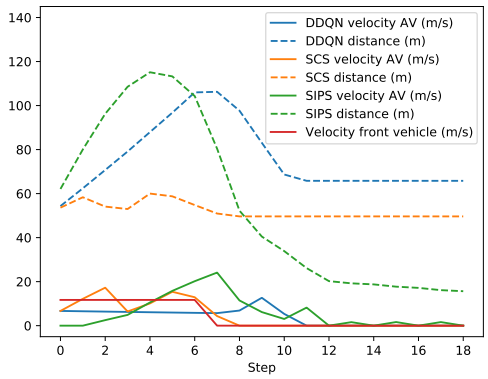
Figure 6.5a shows the behavior of the final policies on the first task, in which a vehicle stands




(a) One simulation of a policy learned with a regular DDQN, the SCS and the SIPS on a task in which one vehicle stands still 140 meters in front of the AV.




(b) One simulation of a policy learned with a regular DDQN, the SCS and the SIPS on a task in which one vehicle drives in front of the AV with a constant velocity of 6 m/s.




(c) One simulation of a policy learned with a regular DDQN, the SCS and the SIPS on a task in one vehicle drives in front of the AV with a velocity of 12 m/s and fully brakes after 10 seconds.

Figure 6.5: Plots of the behavior of the best performing policies of every system on different tasks within scenario 2.

still 140 meters in front of the AV at the start of the episode. The policy learned by the SCS seems to be the smoothest, using a lot of throttle until getting close to the AV, after which the brake is used. The behavior of the policy learned with the regular DDQN is less smooth. When the AV gets within 50 meters of the other vehicle, it switches between using the throttle and braking every step, getting close to the other vehicle very slowly. After the episode finished, it is still around 40 meters away from the leading vehicle. Finally, the policy learned with the SIPS is also not very smooth, but it does manage to get closer to the leading vehicle than the other policies in a safe manner. Figure 6.5b shows the behavior of the final policies on the second task, in which the leading vehicle drives with a constant velocity of 6 m/s throughout the episode. Here the regular DDQN behaves in the smoothest way, but the two shielding systems stay closer to the leading vehicle. Finally, Figure 6.5c shows the behavior of the final policies on the third task, in which the leading vehicle starts with a velocity of 12 m/s, but brakes after ten seconds. In this task, the SCS policy responds quickest and stays at a quite constant distance to the leading vehicle throughout the episode. The SIPS policy responds slower, but in the end gets closer to the leading vehicle. The DDQN performs worst, responding late and keeping a large gap to the leading vehicle at all times.

It must be noted that these figures all come from one simulation of one task, which makes it hard to draw general conclusions from the comparison between them. They mostly show how the driving behavior of the agents looks. Most of the final policies learned to keep a large gap to the

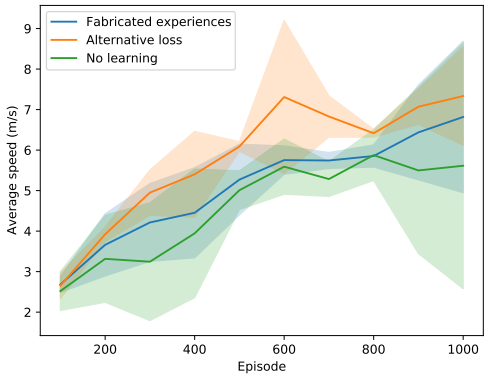
leading vehicle at all times, which guarantees safety but is not necessarily preferred. Some of the policies are a bit smoother than others, depending on the task. Most of the behavior would be quite odd if it would happen in real life. This could be solved by adding a smooth driving component to the reward function, for example by punishing the agent if it switches between actions too often. Having a continuous action space instead of a discrete one might also increase the smoothness of the policies. This research was not focused on smoothness though, but on safety and efficiency.

Overall, the results that were presented in this section match with the expectations that were stated in Section 6.1. All systems were able to increase their return during training on both scenarios, indicating that they were able to learn their tasks. The number of collisions of the shielding systems was zero, as expected. Both the number of collisions of the regular DDQN and the percentage of overruled actions for the shielding approaches decreased during training. In terms of efficiency, all three methods performed very similar. During training on scenario 2, the regular DDQN learned to be efficient a bit sooner, but the resulting policy did not differ much from the shielding systems.

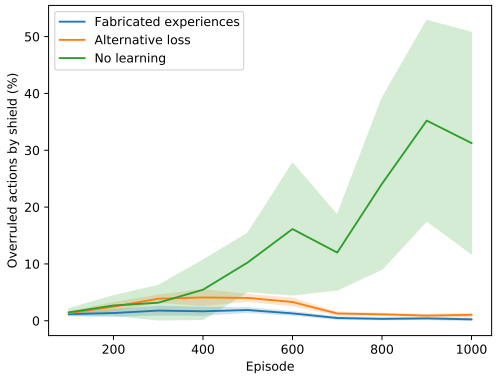
The goal of this thesis was to create a system that ensures safety while preserving efficiency when it is employed around a baseline RL algorithm. The results show that the systems that use a shield had no collisions during training and execution, while behaving very similar in terms of efficiency and return to the baseline DDQN. This shows that shields enable the agent to train safely when using RL. When the regular DDQN would be used to train an AV in real life, it would have had a collision within a few hundred meters at best on both scenarios. If one of the shields would be employed though, the AV would have had zero collisions as far as the simulations show.

### 6.2.2. Shield-based learning

Training an RL agent with a shield causes a problem. When every dangerous action is overruled by a shield, the agent is not able to learn to avoid those dangerous actions since no experiences containing these actions are generated. As introduced in Section 4.3, two methods are proposed to solve this problem: creating fabricated experiences and using an alternative loss function. This section will show the results of experiments that compare these methods. As stated previously, these experiments are run with the SCS on scenario 2. Next to the two methods that are tested, the RL agent is also trained without a shield-based learning technique to check the importance of the agent being able to learn from dangerous experiences. For this setting, evaluation is done by both keeping the SCS employed and removing the SCS.




(a) The average velocity (m/s) over episodes when training an RL agent using the SCS on scenario 2 for multiple shield-based learning techniques.




(b) The percentage overruled actions over episodes when training an RL agent using the SCS on scenario 2 for multiple shield-based learning techniques.

Figure 6.6: Plots of the average velocity and percentage of overruled actions during training of an RL agent on scenario 2 with the SCS employed, using fabricated experiences, an alternative loss function and no shield-based learning. Five experiments were run per setting, of which the means and standard deviations of the evaluation criteria are plotted.

Figure 6.6 shows two evaluation criteria during training for the two shield-based learning methods and the setting where no shield-based learning is used. Since the number of collisions is always zero for every setting, its plot is omitted. The reward function is based on the velocity and the number of collisions. Since there are no collisions, the return figure is very similar to the average velocity figure and therefore it is omitted as well. For the alternative loss function, one of the simulations was replaced since the agent did not learn anything in that simulation because of an NaN loss. This shows that even if the alternative loss function performs well in most experiments, it has a risk of not working. If the failed simulation would not be replaced, it would be harder to tell something about the performance of the alternative loss function when it does converge.

The average velocity of all systems is shown in Figure 6.6a. Although the standard deviations overlap a bit, it is shown that using the alternative loss function yields a higher average velocity during the whole training process than when fabricated experiences are used. It is able to get a higher velocity quicker and keep this lead during the whole process. The difference is not very large though. The standard deviations indicate that both methods are quite consistent. This is different for the setting in which no shield-based learning is used. This setting is much less stable, and yields a lower average velocity throughout the training process. Not being able to learn to detect dangerous situations has a negative effect on the efficiency of the agent, even though the SCS overrules unsafe actions.

Figure 6.6b shows the percentage of overruled actions during training. Here we see a large difference between the systems. Both the fabricated experience system and the alternative loss function system have a slight increase in overruled actions at the beginning of training, after which the percentage of overruled actions is decreased again. It was explained previously that this is likely caused by the fact that the agent first learns to increase its velocity before it learns to drive safe. The standard deviations show that both shield-based learning systems are quite stable. The fabricated experiences system needs less interference from the SCS throughout the training process. When no shield-based learning technique is used, it is shown that the SCS needs to intervene much more. This is likely caused by the fact that the agent learns to increase its velocity, but it does not learn to avoid danger. Therefore it will only speed up and the SCS has full responsibility to overrule dangerous actions. At the end, around one in three actions needs to be overruled. The system without shield-based learning is also much less consistent. Overall, it seems to not have gathered any intelligence in terms of avoiding collisions, as expected.

| | **Collisions per km** | **Average velocity (m/s)** |
|---|---|---|
| **Fabricated experiences** | 0.0 ± 0.0 | 6.0 ± 0.4 |
| **Alternative loss function** | 0.0 ± 0.0 | 7.0 ± 0.2 |
| **No learning** | 1.1 ± 1.8 | 5.2 ± 3.6 |
| **No learning (with SCS during execution)** | 0.0 ± 0.0 | 6.5 ± 0.7 |

Table 6.2: The average *col* and *sp* and standard deviations of five final policies that were simulated for 100 episodes on scenario 2 for each shield-based learning method. The first three rows in the table were evaluated without the SCS employed, the last is evaluated with the SCS employed.

After the training process, the final policies were evaluated during 100 episodes. The fabricated experiences system and the alternative loss function system were both evaluated without the SCS being employed. The system without shield-based learning was evaluated both with and without the SCS being employed. The evaluation results are shown in Table 6.2.

Both of the shield-based learning techniques result in zero collisions during execution. This means that the learning techniques were effective to teach the RL agent to stay safe. When no learning technique is used, the AV does have collisions during execution. As expected, the agent that was not able to learn to avoid unsafe situations is still completely safe if the SCS is employed during execution.

In terms of efficiency (the average velocity), there is not a huge difference between the systems. The fabricated experiences system and the alternative loss function system are both quite stable, with the alternative loss function system being a bit more efficient than the fabricated experiences system. The no learning system with the SCS employed during execution is between the two in terms of efficiency. Its standard deviation overlaps with both of them. All three perform very similar, having zero collisions and a similar velocity. The system that does not use any shield-based learning technique and has no shield employed during execution performs much worse, both for safety and efficiency. It is also much less consistent. This makes sense, since it does not have any experience in dealing with unsafe situations, and therefore the Q-values belonging to potentially unsafe actions are quite random. In some cases, the agent might always pick unsafe actions, while it might never pick unsafe actions in other cases. This can cause the instability.

Overall, the alternative loss function seems to yield the best results. As stated before though, one of the simulations did not work, which makes the method is bit less stable. If the issue of the NaN loss would be fixed though, the alternative loss function system would be the best performing system. The system without shield-based learning does perform well when the shield is employed during execution, but it does not understand the environment in terms of avoiding collisions, which makes sense since it did not get the opportunity to learn to detect dangerous actions. Safety is guaranteed during execution though, since the SCS is employed during execution. The fabricated experiences system is able to consistently learn its task and the fabricated experiences allow the agent to learn how the avoid collisions. That is the reason that this technique was chosen to use for the comparison experiments. If it would be possible to improve stability for the alternative loss function technique, it could outperform the fabricated experiences technique. Both techniques show that they can effectively enable the agent to learn to avoid unsafe actions.

# 7

# Conclusion and Future Work

The previous chapters introduced the subject of this thesis, described background information about (Safe) Reinforcement Learning (RL), formalized the problem, described the Safety Checking Shield (SCS) and the Safe Initial Policy Shield (SIPS), and showed experiments with these shields. The goal of this thesis was to create a system which ensures that an Autonomous Vehicle (AV) is able to safely train in the real world on certain scenarios using RL. This chapter will draw conclusions about whether that problem was solved based on the experiments that were carried out. Furthermore, some potential future work will be listed.

## 7.1. Conclusion

The goal of this thesis was create a system that ensures safety when training an RL agent to drive an AV, as formalized in the research questions in Section 1.4. One promising technique of creating driver models for AVs is RL, but it is not possible to safely train AVs in real life when using regular RL. Therefore a system needed to be created to ensure safety during training and execution. Two shields were tested which aim to solve the problem of unsafety. They are built around a regular RL algorithm and compared to it to show the effects of employing them.

First, an RL algorithm was implemented to be able to show the issue of unsafety, to build the shields around and to compare the performance of the shields against. The chosen algorithm is a Double Deep Q-Network (DDQN). Chapter 6 showed that this DDQN was able to learn a fully safe and efficient policy on scenario 1 and a safer and more efficient policy than a random policy on scenario 2. It also immediately showed the issue that training this DDQN causes a lot of collisions on both scenarios.

The two shields based on two different models that should solve this issue were described in Chapter 4: the SCS and the SIPS. Both let the agent create a sorted list of actions based on preference, based on the height of the Q-values, and then check the actions one by one until a safe action is found. That action is then executed. These shields should ensure safety of any policy that they are built upon. Furthermore, Chapter 4 described two methods that enable the agent to learn which actions not to propose based on the overruled actions by the shields. These learning methods should ensure that the agent learns to only propose safe actions when the shield is not employed anymore.

Chapter 6 showed a comparison between the baseline DDQN and the two shields to check whether the shields have the desired influence on the RL process. For both the SCS and the SIPS, it turned out that employing them on the scenarios resulted in no collisions during training. Furthermore, the efficiency of the final policy was very similar to the efficiency of the regular DDQN. It can therefore be concluded that the shields improve safety while not decreasing efficiency on the two scenarios that were tested. This means that both shields are suitable techniques to solve the problem of unsafety that was investigated in this thesis.

Chapter 6 also showed a comparison between two shield-based learning techniques, the fabricated experiences and the alternative loss function, and a system where no shield-based learning is used. It was shown that the shield-based learning techniques both enable the agent to effectively learn to avoid dangerous situations. The alternative loss function was a bit more efficient, but less stable. Both techniques are promising and could be tuned further to check whether they can work even better.

Overall, one new type of shielding was approached, the SIPS, which is based on a Safe Initial Policy (SIP). It was compared with a more standard type of shielding, the SCS. Both work well on the autonomous driving scenarios that were tested in this research. They ensure safety during training while not sacrificing performance of the final policy. This confirms that shielding is a promising Safe RL technique which can be used in the field of AVs. It should be possible to train AVs in the real world with RL when using such a technique. RL could very well be able to create better driver models than the current techniques that are used for this. Creating driver models using RL could be a step towards a future in which AVs are part of our daily life. A future in which there are way less traffic incidents, where cities become less crowded, where congestion and emissions are reduced, where people experience less stress during their daily commute and have an improved productivity and where transportation is available for more people.

## 7.2. Future Work

This thesis investigated two shields on two autonomous driving scenarios. Although the research of this thesis is concluded, there are still many opportunities for future research based on this thesis. This section describes possible directions for future research.

One of the main issues when running experiments for this research was the lack of processing power. The CARLA simulator is quite heavy and slow and only runs one simulation in real time on one machine. Since only one machine was available, this resulted in needing almost a week for the simulations of one setting to finish. If the experiments would have taken less time, which might be possible if more machines were used, it would have been possible to do more hyperparameter tuning, and to train the agent longer than at most 1000 episodes. This would likely increase the performance of the regular RL algorithm as well as the shields and the shield-based learning methods. Therefore it is recommended to make sure to have enough processing power when a similar system is built and tested, or when new experiments are conducted with the current system.

In Section 2.4, it was shown that all Safe RL research in the field of AVs only focuses on one or two simple scenarios. The shielding systems created in this thesis are also only tested on two scenarios. The goal of Safe RL for AVs is to create systems that can be used for real life AVs, which is why they should be able to deal with all traffic scenarios that can be encountered. It would be interesting to check whether the shields created in this thesis, as well as existing Safe RL techniques for AVs, would generalize well to other traffic scenarios. For this research, a third scenario was created as well, which consists of a multiple lanes road with traffic on which the AV needs to learn to drive safe and efficient. Unfortunately, due to high simulation times and a lack of research time, this scenario was omitted. It would be a start to test the systems created in this research on this more complex scenario.

In this thesis, a new shielding technique called the SIPS, which is based on a SIP, was proposed. It was only tested on the two scenarios that were described in this research, on which it showed to work quite well. Creating a model that can propose one safe action for every state might sometimes be easier to build than a model which can judge action safety for every state, which means that the SIPS might be easier to build for some other applications than the SCS. It would be interesting to test whether the SIPS also works on different kinds of problem or in other fields. Other than testing the SIPS on a different type of problem or within a different field, it would be interesting to try different settings of the SIPS with different types of Safe Initial Policies to check how that affects

the performance of the SIPS. For example, it would be good to test what happens when the SIP is not perfectly safe, and to check how much the SIPS can improve on the SIP. The version tested in this thesis is just a basic version without much tuning, because of the long simulation times. It is very well possible that the SIPS can perform much better when more tuning is done. It might for example be able to safely learn a good policy with a poor SIP.

Another idea is to check whether the SIPS can be used to calibrate existing rule-based driver models. An existing driver model could be used as the SIP that is used by the SIPS. The RL system using the SIPS is likely able to produce an even better policy than the SIP if a better policy exists. Data from simulations of this new policy can be saved and analyzed to check how it behaves in comparison to the initial driver model. Based on its behavior, the parameters of the initial driver model can be calibrated to perform better than before. This results in an improved driver model that can still be rule-based, instead of it being a black box Deep Q-Network.

In this research, it was assumed that all information about the environment was available to the agent at all times. In the real world, this is not the case. Techniques like camera's, radar and LIDAR need to be used to build a representation of the environment that the agent can use. This representation may not always be correct. It could be useful to check how the performances of the shields are influenced when the representation about the environment is wrong.

Two shield-based learning techniques were tested in this thesis: creating fabricated experiences and using an alternative loss function. There are more possible techniques that could be tested, which might perform even better. One example of an idea is to add a binary layer as the last layer of the Q-network. This binary layer should multiply all Q-values with 1 if they are assessed safe by the shield, and with 0 if they are assessed not to be safe, assuming that all Q-values are positive. This would result in only safe actions having a positive Q-value and therefore they would always be chosen.

One technique that was briefly tested, but was not adopted for this thesis is using a variable punishment for the fabricated experiences, which is one of the shield-based learning techniques. When the proposed action by the agent is very unsafe, for example when it is far away from the safe action range when using the SIPS, the punishment could be higher than for slightly unsafe actions. This enables the agent to learn when actions are very unsafe and when they are only slightly unsafe, which could result in a more efficient and flexible final policy. Using these variable punishments did not make a difference on the scenarios tested in this research, but it might very well have an impact on other problems.

Finally, all of the papers that propose Safe RL techniques for AVs do this to be able to train AVs in real life, while none of these methods are actually tested on real life AVs. It is not very realistic, but it would be really interesting to see what happens when the systems created in this thesis, as well as other Safe RL systems, are employed on a real life AV in a controlled environment. In such research, the assumption that no irregular events happen can not be made anymore. The shields should be adapted in such a way that they can deal with such events. Only if tests like these are done successfully, it can be concluded that Safe RL techniques work well enough to help an AV to learn a good driving policy in the real world by using RL while staying safe.

# Bibliography


[1] Road safety. URL https://www.who.int/data/gho/data/themes/road-safety.

[2] Global road safety, Dec 2020. URL https://www.cdc.gov/injury/features/global-road-safety/index.html.

[3] Sae j3016 automated-driving graphic, May 2020. URL https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic.

[4] Pieter Abbeel, Adam Coates, and Andrew Y Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[5] Saleh Albeaik, Alexandre Bayen, Maria Teresa Chiri, Xiaoqian Gong, Amaury Hayat, Nicolas Kardous, Alexander Keimer, Sean T McQuade, Benedetto Piccoli, and Yiling You. Limitations and improvements of the intelligent driver model. *arXiv preprint arXiv:2104.02583*, 2021.

[6] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[7] James M Anderson, Kalra Nidhi, Karlyn D Stanley, Paul Sorensen, Constantine Samaras, and Oluwatobi A Oluwatola. *Autonomous vehicle technology: A guide for policymakers*. Rand Corporation, 2014.

[8] James Arbib and Tony Seba. Rethinking transportation 2020-2030. *RethinkX, May*, 143:144, 2017.

[9] Edmond Awad, Sohan Dsouza, Richard Kim, Jonathan Schulz, Joseph Henrich, Azim Shariff, Jean-François Bonnefon, and Iyad Rahwan. The moral machine experiment. *Nature*, 563 (7729):59–64, 2018.

[10] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[11] Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence Jackel, and Urs Muller. Explaining how a deep neural network trained with end-to-end learning steers a car. *arXiv preprint arXiv:1704.07911*, 2017.

[12] Maxime Bouton, Alireza Nakhaei, Kikuo Fujimura, and Mykel J Kochenderfer. Safe reinforcement learning with scene decomposition for navigating complex urban environments. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 1469–1476. IEEE, 2019.

[13] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[14] Dong Chen, Longsheng Jiang, Yue Wang, and Zhaojian Li. Autonomous driving using safe reinforcement learning by incorporating a regret-based human lane-changing decision model. In *2020 American Control Conference (ACC)*, pages 4355–4361. IEEE, 2020.

[15] Simiao Chen, Michael Kuhn, Klaus Prettner, and David E Bloom. The global macroeconomic burden of road injuries: Estimates and projections for 166 countries. *The Lancet Planetary Health*, 3(9):e390–e398, 2019.

[16] Richard Cheng, Gábor Orosz, Richard M Murray, and Joel W Burdick. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In *AAAI Conference on Artificial Intelligence*, volume 33, pages 3387–3395, 2019.

[17] François Chollet et al. Keras. `https://keras.io`, 2015.

[18] Marc Claesen and Bart De Moor. Hyperparameter search in machine learning. *MIC 2015: The XI Metaheuristics International Conference*, 2015.

[19] Jeffery A Clouse and Paul E Utgoff. A teaching method for reinforcement learning. In *Machine Learning Proceedings*, pages 92–101. Elsevier, 1992.

[20] Jeffery Allen Clouse. *On integrating apprentice learning and reinforcement learning*. University of Massachusetts Amherst, 1996.

[21] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

[22] Erik Dovgan. Multiobjective discovery of driving strategies using the scaner studio. *Dep. Intell. Syst*, 2017.

[23] Kurt Driessens and Sašo Džeroski. Integrating guidance into relational reinforcement learning. *Machine Learning*, 57(3):271–304, 2004.

[24] Marius Dupuis, Martin Strobl, and Hans Grezlikowski. Opendrive 2010 and beyond–status and future of the de facto standard for the description of road networks. In *Driving Simulation Conference Europe*, pages 231–242, 2010.

[25] Daniel J Fagnant and Kara Kockelman. Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. *Transportation Research Part A: Policy and Practice*, 77:167–181, 2015.

[26] Nathan Fulton and André Platzer. Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In *AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[27] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[28] Laura García Cuenca, Enrique Puertas, Javier Fernandez Andres, and Nourdine Aliane. Autonomous driving in roundabout maneuvers using reinforcement learning with q-learning. *Electronics*, 8(12):1536, 2019.

[29] Bernd Gassmann, Fabian Oboril, Cornelius Buerkle, Shuang Liu, Shoumeng Yan, Maria Soledad Elli, Ignacio Alvarez, Naveen Aerrabotu, Suhel Jaber, Peter van Beek, et al. Towards standardization of av safety: C++ library for responsibility sensitive safety. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 2265–2271. IEEE, 2019.

[30] Clement Gehring and Doina Precup. Smart exploration in reinforcement learning using absolute temporal difference errors. In *International conference on Autonomous agents and multi-agent systems*, pages 1037–1044, 2013.

[31] Matthias Heger. Consideration of risk in reinforcement learning. In *Machine Learning Proceedings*, pages 105–111. Elsevier, 1994.

[32] Ronald A Howard and James E Matheson. Risk-sensitive markov decision processes. *Management science*, 18(7):356–369, 1972.

[33] David Isele, Alireza Nakhaei, and Kikuo Fujimura. Safe reinforcement learning on autonomous vehicles. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–6. IEEE, 2018.

[34] Sergey Ivanov and Alexander D'yakonov. Modern deep reinforcement learning algorithms. *arXiv preprint arXiv:1906.10025*, 2019.

[35] Yoshinobu Kadota, Masami Kurano, and Masami Yasuda. Discounted markov decision processes with utility constraints. *Computers & Mathematics with Applications*, 51(2):279–284, 2006.

[36] Cameron F. Kerry and Jack Karsten. Gauging investment in self-driving cars, Oct 2017. URL `https://www.brookings.edu/research/gauging-investment-in-self-driving-cars/`.

[37] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[38] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[39] Hanna Krasowski, Xiao Wang, and Matthias Althoff. Safe reinforcement learning for autonomous lane changing using set-based prediction. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–7. IEEE, 2020.

[40] Sampo Kuutti, Richard Bowden, Harita Joshi, Robert de Temple, and Saber Fallah. End-to-end reinforcement learning for autonomous longitudinal control using advantage actor critic with temporal context. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2456–2462. IEEE, 2019.

[41] Edith LM Law, Melanie Coggan, Doina Precup, and Bohdana Ratitch. Risk-directed exploration in reinforcement learning. *Planning and Learning in A Priori Unknown or Dynamic Domains*, 97, 2005.

[42] Edouard Leurent. An environment for autonomous driving decision-making. `https://github.com/eleurent/highway-env`, 2018.

[43] Shuo Li and Osbert Bastani. Robust model predictive shielding for safe reinforcement learning with stochastic dynamics. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7166–7172. IEEE, 2020.

[44] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[45] Todd Litman. Autonomous vehicle implementation predictions: Implications for transport planning. 2020.

[46] Björn Lütjens, Michael Everett, and Jonathan P How. Safe reinforcement learning with model uncertainty estimates. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8662–8668. IEEE, 2019.

[47] Rafael Math, Angela Mahr, Mohammad M Moniri, and Christian Müller. Opends: A new open-source driving simulator for research. *GMM-Fachbericht-AmE 2013*, 2, 2013.

[48] Helmut Mausser. Beyond var: From measuring risk to managing risk. *ALGO research quarterly*, 1(2):5–20, 1998.

[49] Oliver Mihatsch and Ralph Neuneier. Risk-sensitive reinforcement learning. *Machine learning*, 49(2):267–290, 2002.

[50] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[51] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[52] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[53] Teodor Mihai Moldovan and Pieter Abbeel. Safe exploration in markov decision processes. *arXiv preprint arXiv:1205.4810*, 2012.

[54] Subramanya Nageshrao, H Eric Tseng, and Dimitar Filev. Autonomous highway driving using deep reinforcement learning. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 2326–2331. IEEE, 2019.

[55] Arnab Nilim and Laurent El Ghaoui. Robust control of markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005.

[56] Jingyu Niu, Yu Hu, Beibei Jin, Yinhe Han, and Xiaowei Li. Two-stage safe reinforcement learning for high-speed autonomous racing. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3934–3941. IEEE, 2020.

[57] Martin L Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[58] Max Peter Ronecker and Yuan Zhu. Deep q-network based decision making for autonomous driving. In *2019 3rd International Conference on Robotics and Automation Sciences (ICRAS)*, pages 154–160. IEEE, 2019.

[59] Andrew Sanders. *An introduction to unreal engine 4*. CRC Press, 2016.

[60] Makoto Sato, Hajime Kimura, and Shibenobu Kobayashi. Td algorithm for the variance of return and mean-variance reinforcement learning. *Transactions of the Japanese Society for Artificial Intelligence*, 16(3):353–362, 2001.

[61] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.

[62] Donald Shoup. *The high cost of free parking: Updated edition*. Routledge, 2017.

[63] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[64] Cambridge Systematics. Traffic congestion and reliability: Linking solutions to problems. Technical report, United States. Federal Highway Administration, 2004.

[65] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[66] Philip Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High confidence policy improvement. In *International Conference on Machine Learning*, pages 2380–2388. PMLR, 2015.

[67] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[68] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical review E*, 62(2):1805, 2000.

[69] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[70] Martijn Van Otterlo and Marco Wiering. Reinforcement learning and markov decision processes. In *Reinforcement learning*, pages 3–42. Springer, 2012.

[71] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[72] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[73] Lu Wen, Jingliang Duan, Shengbo Eben Li, Shaobing Xu, and Huei Peng. Safe reinforcement learning for autonomous vehicles through parallel constrained policy optimization. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–7. IEEE, 2020.

[74] Yi Zhang, Ping Sun, Yuhan Yin, Lin Lin, and Xuesong Wang. Human-like autonomous vehicle speed control by deep reinforcement learning with double q-learning. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1251–1256. IEEE, 2018.

[75] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744. IEEE, 2020.

[76] Zeyu Zhu and Huijing Zhao. A survey of deep rl and il for autonomous driving policy learning. *arXiv preprint arXiv:2101.01993*, 2021.