Practical implementation of reinforcement learning algorithms for giving personalised speed advice to cyclists approaching intersections using function approximation and Dyna

Systems & Control Master thesis

Midas Becker

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Practical implementation of reinforcement learning algorithms for giving personalised speed advice to cyclists approaching intersections using function approximation and Dyna

Systems & Control Master thesis

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Midas Becker

January 13, 2021

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

Being a safe and healthy alternative for polluting and space-inefficient motorised vehicles, cycling can strongly improve living conditions in urban areas. Idling in front of traffic lights is seen as one of the major inconveniences of commuting by bicycle. By giving personalised speed advice, the probability of catching a green light can be increased whilst taking the cyclist preferences into account. Due to its adaptive properties, Reinforcement learning (RL) is a suited algorithm for developing optimal speed advice policies when dealing with a dynamic traffic environment and unique cyclist preferences. Generally, a large amount of training samples is required to successfully train a RL algorithm. This poses a problem for this specific application since training samples must be generated by humans and are therefore scarce. Moreover, exploration of the environment is challenging since humans will not comply with irrational speed advice. These factors currently restrain the practical implementation of RL algorithms for giving speed advice.

This thesis aims to overcome these problems whilst maintaining a competitive performance compared to conventional RL algorithms. This is done by using function approximators and a combined planning and learning method called Dyna. During a case study, three different function approximators are compared to reduce the amount of required training samples, namely polynomial functions, radial basis functions, and artificial neural networks. Secondly, the effectiveness of Dyna to improve the quality of the speed advice in an unknown environment is assessed. Finally, these methods are applied in a framework focused on the practical implementation of RL for giving speed advice.

It was concluded that function approximation method can significantly reduce the amount of required training samples to train a RL algorithm. Dyna can increase user retention by providing cyclists with a high quality speed advice algorithm during the early learning phase of the algorithm. Therefore, it can be concluded that this RL approach for giving personalised speed advice to cyclist approaching intersections is practically implementable and can even outperform benchmark algorithms in terms of travel time, energy consumption, and safety.

Table of Contents

Preface

In some way, writing this thesis was like learning to ride a bike. Falling off and getting back up was a part of 2020. Especially in the beginning of the learning process, some things were learned the hard way thanks to a high exploration rate. Noor and my parents could always keep me motivated to ensure a high learning rate.

I was very lucky to have some optimal speed advice from Azita and Bart to guide me towards the green light. I learned very much from all your hours of guidance, Azita. I wish you the best of luck with your research on making traffic safer and more sustainable.

People cheering on the side of the road kept me up to speed during this pandemic. I want to thank my roommates Jurjen, Job, Frank, Koen and Paul for exploring the unknown environment of Delfshaven with me. Joost, Pieter and Emiel helped me to smoothly accelerate towards that green light at the end of this bumpy trajectory. Who needs an ANN-Dyna RL algorithm, when you have such great friends?

Chapter 1

Introduction

Due to a growing urban population, cities are getting crowded with motorised vehicles. This increases safety risks, air pollution, and travel time. Therefore, many countries are now heavily promoting the use of bicycles as a healthy and clean alternative for polluting motorised vehicles. The Netherlands is an illustrative example, where high-quality bicycle infrastructure and abundant bicycle parking facilities encourage more people to commute by bicycle [49]. It is even stated that on average, the life expectancy of Dutch people is increased by half a year due to the nation's cycling habit [49].

A proven method to encourage cycling is by reducing idling time caused by traffic lights since this is experienced as a major inconvenience by cyclists [28]. Idling is not only time-consuming, but energy inefficient as well. The example given in [19] states that cycling at 5.6 m/s on a road with a stop sign at every 100 meters requires five times as much energy as on a road without stop signs. Besides these inconveniences, it has been established that red-light running is a major contributor to traffic incidents [37, 46]. On average, 55% of the cyclist fatalities in the Netherlands occurred at intersections [28]. Therefore, a better user experience with traffic lights is not only encouraging people to commute by bicycle, but could decrease traffic accidents at intersections as well.

Assisting cyclists to catch green is not a simple task. So-called "green waves", guaranteeing green lights for cyclists cycling at a certain speed, were created in cities in the Netherlands, Denmark, and Germany [41]. However, there is still room for improvement in this concept since each cyclist has a different comfortable speed [17]. Deploying actuated traffic lights that give cyclists higher priority for catching a green light can cause long delays for other road users and be quite costly in maintenance [18]. However, technological advances in cooperative intelligent traffic systems have opened up new possibilities in assisting users to catch green lights. By sharing traffic light information with incoming vehicles, drivers can adjust their speed to increase the chance of catching green without stopping [29, 30, 31, 32, 33, 48, 50]. Since these intelligent speed advisory systems (ISAS) have proven to be energy efficient and capable of reducing idling time for motorised vehicles, ISAS were applied to cyclists as well [13, 14, 15]. By adapting the speed of the cyclist to the current traffic light phase, no additional idling time is imposed upon surrounding traffic on the intersection. Furthermore,

speed advice can be adapted to individual cyclist's preferences. Based on the shared traffic light messages, algorithms can be used to compute speed advice that will yield the largest probability of catching a green light whilst cycling an energy-efficient trajectory. For the remainder of this thesis, generating this personalised speed advice will be referred to as "the speed advice problem".

Reinforcement learning (RL) methods have been gaining attention as an alternative to solve decision-making problems. This can be explained by the fact that RL has the ability to learn innovative strategies for complex problems that humans cannot come up with easily. This makes RL a suitable algorithm for giving optimal speed advice in a dynamic and nonstationary traffic environment. Nonetheless, little research on using RL for giving optimal speed advice can be found [6]. In [15], an effective RL algorithm is presented. However, a large amount of training samples is required to successfully train the RL algorithm. In the speed advice problem, training samples are human-generated, making these samples scarce by nature. Moreover, RL algorithms often take suboptimal actions to explore unknown environments to find high-yielding policies. Humans are not likely to comply with irrational speed advice in the early stages of exploration. These factors currently restrain the practical implementation of RL algorithms for giving speed advice.

1-1 Problem statement

This thesis aims to overcome the problems that are currently restraining RL-generated speed advice from practical implementation. This is done by using function approximators and a combined planning and learning method, called Dyna. Therefore, the main research goal of the thesis is:

> *"Practical implementation of reinforcement learning algorithms for giving personalised speed advice to cyclists approaching intersections using function approximation and Dyna."*

The RL algorithm is considered practically implementable when the following requirements are met:

1. *The speed advice technology is accessible and user friendly.*
   Invasive technological enhancement of the vehicle is undesired. The cyclist's senses must be supported by the speed advice application, not distracted.

2. *The speed advice significantly improves the user's cycling experience compared to cycling without speed advice.*
   Without a significant benefit gained from the speed advice, user retention can deteriorate.

3. *The agent must learn to give high-quality speed advice within a reasonable number of traffic light encounters.*
   When the required training experience for the RL algorithm is too large, cyclists will lose patience and incentive for using the speed advice application.

4. *The speed advice may never negatively impact cycling experience.*
   When not executed properly, speed advice algorithms can increase energy, travel time, and safety risks.

Conventional RL algorithms do not meet the last two requirements due to their need of exploration. Yet, these requirements are of vital importance. In this thesis an RL approach is developed that aims to meet each of the above requirements whilst remaining competitive in terms of performance compared to conventional RL algorithms by overcoming the following two challenges:

- *Scarcity of training samples by proper choice of function approximation methods.*
- *Restricted environment exploration by integrating planning and learning using Dyna.*

The practical implementation will be assessed by computer simulations during a case study. In this case study, three different function approximators are compared, namely polynomial functions, radial basis functions, and artificial neural networks. Secondly, the effectiveness of multiple variations of Dyna is compared. Finally, these methods are applied in a framework focused on the practical implementation of RL for giving speed advice.

1-2 Thesis contribution

Promoting cycling can strongly improve human living conditions. One way to do this is by giving speed advice to cyclists approaching an intersection to increase safety, reduce energy consumption, and reduce travel time. Due to its adaptive properties, RL has presented itself as a suitable algorithm for generating personalised speed advice. Training RL algorithms with humans-in-the-loop impose two challenges. First, training samples are scarce since they must be human-generated. Secondly, exploration of the environment is challenging since humans will not comply with irrational speed advice. This thesis aims to overcome these problems, making RL-generated speed advice practically implementable.

As growing computer power opens up new possibilities in human-machine interaction, algorithms must be able to learn from humans. The speed advice problem is a representative example that reveals the problems imposed by human-machine interaction. Therefore, this thesis is both relevant for giving speed advice to traffic participants, and for algorithms interacting with human behaviour in general.

1-3 Thesis outline

The outline of the remainder of this thesis is as follows. Chapter 2 discusses the state of the art of the technological developments in traffic communication that can be utilized for giving speed advice to traffic participants. Chapter 3 introduces the reader to the concept of RL, the application of function approximation, and Dyna. In Chapter 4, the models used to simulate the traffic environment and the cyclist behaviour are defined. The case studies researching the practical implementation of RL algorithms for giving speed advice are discussed in Chapter 5. Finally, the conclusions of this thesis and future research are presented in Chapter 6.

Chapter 2

Intelligent speed advisory systems

Technological advancements in sensing, communication, and computation have opened up more possibilities in developing intelligent speed advisory systems (ISAS). The aim of developing an ISAS may be increasing the safety of users, minimizing travel time, minimizing energy consumption, minimizing fuel emissions, or a combination of them. Acknowledging the need for reducing the abundant use of motorised vehicles has put a spotlight on alternative modes of transportation such as cycling. One related field of research, that is the focus of this chapter as well, is designing an ISAS for guiding cyclists to pass through signalised intersections. In this chapter, several technological requirements that must be incorporated in this specific type of ISAS are discussed. At the end of this chapter, concept choices for each technological requirement that is discussed are motivated, thereby justifying the ISAS used in the case study.

2-1 Future traffic light phase information

To give speed advice to guide a vehicle or cyclist in passing a signalised intersection, it is important to know how the traffic light phase will evolve. When future traffic light phases are difficult to predict, this can result in fluctuating speed advises, thereby influencing the level of compliance of the user and therefore deteriorating the performance of the speed advice [8, 28, 62, 63]. Actuated traffic lights adapt their phase based on incoming traffic. This flexibility in traffic light phase improves traffic flow, but makes the future traffic light phase more difficult to predict. Not only actuated traffic light controllers are difficult to predict but even fixed-time traffic signals drift significantly due to variations in the electric grid frequency [27]. Furthermore, some fixed-time traffic lights have different programs that are selected based on the day of week [70]. To provide the user with an accurate prediction of the future state of the traffic light, the current state of the traffic light and the transition probability of going to a next state are required [13, 30].

2-1-1 Current state of the traffic light

Recent developments on sensing and communication technologies have been of great use in improving the accuracy of traffic light signal prediction. An example of such a system is dedicated short-range communication where, within a local perimeter, messages are transmitted between vehicles and infrastructure such as traffic lights [8, 10, 70]. This method is used for communicating the current phase of the traffic light, alert vehicles for incoming collisions, traffic congestion avoidance or platooning [2, 32]. Dedicated short-range communication shows promising results but also has its drawbacks since simulations often overestimate the communication performance due to signal attenuation caused by the surroundings and message processing delays [55]. Moreover, all vehicles and infrastructure must be equipped with dedicated short-range communication technology.

In [5, 66, 67], the use of a centralized server is suggested that collects and distributes the traffic data of vehicles and infrastructure. In addition, phones can now be used to connect to this data server and use the real-time traffic information. In [26, 36], mobile phones are used to capture images and track GPS data to synchronize with historical fixed-time traffic light data or to predict the next traffic light state in case of actuated traffic lights.

2-1-2 Traffic light state prediction

Knowing the current state of the traffic light is only a part of the information that is required to know the future state of the traffic light. The probability of a traffic light transitioning to its next phase is usually obtained by historical data [13, 14, 15, 30]. Since the average phase times of these traffic lights are not constant, this procedure is repeated for different times of a day and days of a week. In [26], support vector regression is used on a historical training data set shared by mobile phones images to predict the next state of actuated traffic lights. The model was tested in real-life experiments and predicted from one to even four phases ahead with a rather small error. This model has to be retrained every four to eight months for every intersection. Also, other information such as operating logic of signalised intersections, infrastructure sensor data, and crowdsource information is used to improve this prediction [9, 30, 40].

2-2 Surrounding traffic

The current traffic light signal prediction methods only predict the future colour of the light. However, when the vehicle approaches the vicinity of the intersection, there may be a queue waiting in front of the traffic light, limiting the number of users that can pass the intersection. As can be seen in Figure 2-1, the vehicle or cyclist may not be able to follow the trajectory generated by the speed advice because of the presence of a queue at point A. A new trajectory must be generated where the vehicle should cross the intersection at point B to maintain a smooth speed transition [22]. Research has been done on the modelling of cyclists approaching and queuing in front of a traffic light [20, 45].

Surrounding vehicles are another issue that has an effect on the speed of vehicles or cyclists on the road. As can be seen in Figure 2-2, it can be the case that a preceding vehicle is

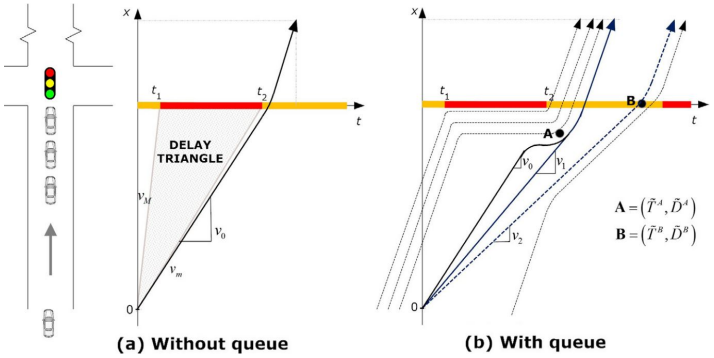

Figure 2-1: Speed trajectories generated to avoid idling whilst minimizing acceleration and braking. Figure (a) shows the optimal trajectory without consideration of queues, whereas Figure (b) takes the queue into consideration [22].

heading for a different traffic light, thereby cruising at a lower speed, and blocking other cars that require a higher cruising speed to reach their traffic light of interest in time.

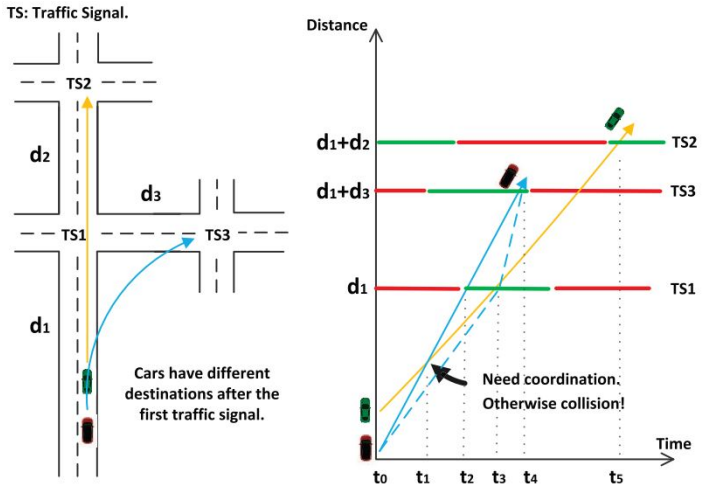

Figure 2-2: Example scenario of two conflicting driving trajectories. When not taking surrounding traffic into consideration, a collision can occur [68].

2-3 Human-machine interface

Various forms of human-machine interaction have already been tested for cyclists. One way to communicate the speed advice to the cyclist is to use a roadside sign that displays the information. An example can be seen in Figure 2-3 (left) where the speed advice is schematically displayed by figures. Roadside signs have the advantage that not all cyclists have to be equipped with sensors and communication devices [28]. However, this also means that speed advice cannot be communicated frequently over a trajectory and can not be personalised [15]. Moreover, a roadside unit is only helpful when it is in the range of sight of the cyclist.

An on-board unit can communicate with the user more frequently. For cyclists to have information on the current state of the traffic light, a communication device is required.

According to [5], one way to resolve this issue is by using a centralized data server connection with a smartphone, as is displayed in Figure 2-3 (centre). Communication between the user and the system can be through a screen or by using an audio earpiece and smart motor support when riding an e-bike [3].

A method of communicating speed advice that combines more frequent communication opportunities and the use of infrastructure can be seen in Figure 2-3 (right). Small lights are placed on the ground, turning green when a cyclist is cycling at the speed to cross the traffic light when green.

Figure 2-3: Speed advice communication methods: roadside sign (left), smartphone (center), embedded lights (right).

2-4 Conclusions

Developments in cooperative traffic communication technology have opened up new possibilities for ISAS. Messages can be sent and received through different types of networks and processed in a distributed way or by collection on a centralized server. Information on the current phase of the traffic light and historical data on the duration of the traffic light's phases can be used to predict the future states of actuated traffic lights. Since it is our goal to develop ISAS that is personalised for each individual cyclist, the combination of a smartphone communicating with a centralized server provides a suitable setting. The centralized server allows the algorithm to collect samples over the entire trajectory and post-process the data offline. Communicating through a smartphone allows frequent speed advice without invasive technological changes to the bicycle. When using this type of human-machine interaction, the first requirement in the problem statement can be considered to be fulfilled. Surrounding traffic and queue formation can constrain the possible trajectories that the user can follow. However, this topic is deferred to future research.

Chapter 3

Reinforcement learning

Reinforcement learning (RL) methods have been gaining more and more attention as a method to solve complex decision-making problems. This can be explained by the fact that RL has the ability to learn innovative strategies for complex problems that humans cannot come up with easily. RL has been used to beat the world champion of the complex game Go [51], learning a robot how to walk [4], and optimizing chemical reactions [69]. The speed advice problem can also be defined as a decision-making problem where at each time step an action, in the form of speed advice, has to be chosen to control the cyclist speed in a way that e.g. reduces travel time, reduces energy consumption, ensures safety, and maximizes the probability of catching green. This makes RL a suited algorithm for solving the speed advice problem. The start of this chapter introduces the concept of RL. After this, the use of function approximation will be presented. Finally, planning and learning algorithms will be discussed.

3-1 Introduction to reinforcement learning

As can be seen in Figure 3-1, the RL framework consists of an agent learning by interacting with an environment over a sequence of discrete-time steps to achieve a goal [43, 53]. The agent interacts with the environment during an episode. This means that the agent starts in an initial state and proceeds to take interact until resources run out or a terminal state is reached, then the episode ends. An episode can also be infinitely long in a continuous process. Much of the RL algorithms used are restricted to a finite Markov decision process (MDP). A finite MDP is a sequential decision-making process where the estimate of the future state only depends on the current state and action taken in a finite set of states and actions [7]. MDP suffers from the curse of modelling and the curse of dimensionality [21]. The curse of modelling means that in complex, stochastic systems, it is difficult, if not infeasible to model the transition probabilities from one state to another. The curse of dimensionality means that the computation time grows exponentially with the number of states.

In RL, the agent takes actions which brings the agent from one state to another state in the environment. Based on the chosen action in each state, the agent receives rewards from the

environment. The function of this cumulative reward is called the return. It is the agent's objective to maximize the return over time. This can be done by choosing actions yielding high rewards according to the objective function. A policy is a stochastic rule by which the agent selects these actions based on the current state [43]. The expected return that can be achieved in each state when following the current policy is called the value of that state. The values of all states form the value function and can be stored in a table. One can also store a value for each individual action at each state, called the action-value. Due to the curse of dimensionality, storing action-values of each state-action pair in a table becomes infeasible when increasing the state-action space. Action-value function approximation can be a remedy for this. However, we defer the discussion on function approximators until Section 3-2.

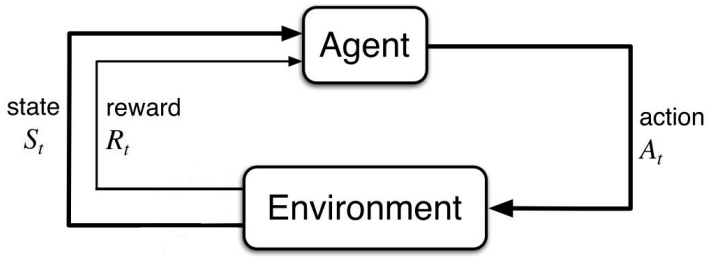

Figure 3-1: The agent-environment interaction [43].

To find the optimal policy, there is a trade-off between spending efforts exploring the environment for new state-action pairs resulting in higher rewards or exploiting those that are known to yield high rewards. This is called the explore-exploit dilemma. When an agent follows the current optimal target policy, these are called on-policy methods. Exploration can be done by sometimes diverting from that policy using for example an ϵ-greedy policy. Where, at each time step, with a probability of ϵ, the agent takes a random action. The agent can also choose not to follow the current target policy and use the information found while exploring another, so-called behaviour policy, to improve the target policy. These are called off-policy methods.

Another distinction between RL algorithms can be made based on their action-value function update mechanism. Some RL algorithms update all the visited state-action pairs at the end of the episodes, whereas others update the value function after each visited state-action pair. There also exist multi-step methods that provide a balance between these two extremes. When action-value pairs are updated before terminating the episode, bootstrapping is used, where the return of the current state value is recursively updated by means of the estimate of action-values of future state-action pairs obtained from the previous experiences.

3-1-1 Q-learning

Q-learning is one of the most well-known algorithms in RL and is used for the remainder of this thesis. Q-learning has been a popular RL algorithm because of some desirable properties that will be explained next [65]. Q-learning is an off-policy algorithm that updates the value of each state-action pair by

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)). \tag{3-1}$$

where $Q_t(s_t, a_t)$ is the action-value for the current state-action pair (s_t, a_t), also called the Q-value, α is the learning rate, γ is the discount factor to weigh closer state-action pairs more heavily than state-action pairs further away from the current state-action pair, r is the reward of the state-action pair, and $\max_a Q_t(s_{t+1}, a)$ is the maximum Q-value of the next state-action pair (s_{t+1}, a).

Like other RL algorithms, Q-learning can learn from experience without a model of the environment. Q-learning uses bootstrapping to update its action-value function after each state-action pair visited. The use of bootstrapping can cause bias in the action-value function. However, by using bootstrapping, Q-learning has faster converging properties than algorithms that only update their action-value function at the end of an episode. This also has memory benefits since the algorithm does not have to store all the visited state-action-pair values. Furthermore, updating each action-value at once at the end of an episode can cause high variance.

As stated in [61], a common pitfall of using off-policy RL methods such as Q-learning is overestimation of the value function when using function approximators which is further explained in 3-2-4. This phenomenon of a positive bias can strongly deteriorate the performance and lead to failure of the algorithm.

3-2 Function approximation

In [44] it is stated that oversimplification of a realistic traffic environment can deteriorate the performance of the speed advice algorithm. Since it is our goal to design a high-performance speed advice algorithm, a large state-space is inevitable. In this section, a framework is presented on how RL algorithms from the previous section can be extended to problems with arbitrarily large state spaces by using function approximation. There is a large diversity of function approximators, yet it has been decided to focus on three examples, each defined by different properties, namely polynomial functions (PF), radial basis functions (RBF), and artificial neural networks (ANN).

3-2-1 Introduction to function approximation

Due to the curse of dimensionality of MDP's, the number of action-values that has to be stored in a table grows exponentially with the number of states and actions. This exponential growth translates into an increase in the memory capacities to store these tables. Moreover, the required time to visit each state-action pair enough to find an accurate action-value function quickly becomes infeasible.

Approximation methods use a weight vector $\mathbf{w}$ and a feature vector $\phi(s, a)$ consisting of combinations of the state-action pair values of interest to approximate the action-value function. This weight vector $\mathbf{w}$ can for example be the weights of a PF or the weights of an ANN that approximate the action-value function. Using the weight vector $\mathbf{w}$, the approximate action-value function can be written as $\hat{Q}(s, a, \mathbf{w}) \approx Q(s, a)$. Since only the variables of the approximation function would have to be updated, the number of variables that would have to be stored is significantly less than the number of states in the state space. This solves the problem of memory capacity of the table. When an action-value is updated, the generalization

property of the function approximator causes other, nearby state-action pairs to be updated as well. This is a remedy for the problem of having to visit each individual state-action pair.

3-2-2 Linear function approximation

Linear function approximation methods are simple, well-understood, and computationally inexpensive compared to nonlinear approximation methods. This subsection will discuss the application of linear approximations methods and will introduce the two methods that are used in the case studies in Chapter 5, namely PFs and RBFs.

Approximating action-value functions In the case of linear approximation methods, $\hat{Q}(s, a, \mathbf{w})$ is approximated by a linear function of weight vector $\mathbf{w}$ and feature vector $\phi(s, a)$. For each state-action pair, a feature vector $\phi(s, a) \doteq [\phi_1(s, a), \phi_2(s, a), \ldots, \phi_\mathrm{d}(s, a)]^\top$ with the same number of components as $\mathbf{w}$ is defined. For the speed advice problem, feature vectors can be constructed using the states and actions described in Chapter 4. Linear methods approximate the action-value $\hat{Q}(s, a, \mathbf{w})$ by the inner product of $\mathbf{w}$ and $\phi(s, a)$ as

$$\hat{Q}(s, a, \mathbf{w}) \doteq \mathbf{w}^\top \phi(s, a) \doteq \sum_{i=1}^{d} w_\mathrm{i} \phi_\mathrm{i}(s, a). \tag{3-2}$$

Linear methods are linear in the weights but the features in the feature vector $\phi(s, a)$ can be designed to capture the nonlinearities of the function. Capturing these nonlinearities can be achieved by mapping action-values using, for example, periodic functions, multiplying state-values with other state-values, or using one-hot encoded feature vectors. Since these features are constructed manually, linear function approximation methods are well-understood and supply information on which features contribute significantly to the estimation. A disadvantage of linear methods compared to nonlinear methods is that features cannot interact with each other since they are coded separately.

Updating weight vectors In order for the agent to learn from the environment, the weight vector $\mathbf{w}$ of the approximated action-value function $\hat{Q}(s, a, \mathbf{w})$ has to be updated. Examples of such an update strategy are gradient-descent methods, linear least squares prediction, or evolutionary algorithms. Gradient-based methods for updating the weight vector of the function approximator are popular due to their fast convergence and simplicity. Since stochastic gradient descent (SGD)-methods train on single data points instead of batches of data points, this method is often used for updating action-value functions when a new state-action pair is encountered. However, since the gradient is only taken over one data point, the gradient tends to be noisy over time which can cause a longer time to converge and overshooting at local minima [47]. A remedy for this can be methods that adjust the step size over time such as Adam [25] and Momentum [42]. In some problems, gradient-based methods are not desired since the gradient is difficult or impossible to compute. Evolutionary algorithms converge slowly and require much computational power but do converge to a global optimum without computing a gradient. Linear least-squares prediction can be used for fast learning but also comes at the cost of computational and memory requirements that scale quadratically with

the size of the state space. Because of their mathematical simplicity and good convergence properties, SGD will be used throughout this thesis to update the weight vector $\mathbf{w}$ as

$$\begin{aligned}\Delta\mathbf{w} &\doteq -\frac{1}{2}\alpha\nabla\left(Q\left(s_t,a_t\right)-\hat{Q}\left(s_t,a_t,\mathbf{w}\right)\right)^2\\ &= \alpha\left(Q\left(s_t,a_t\right)-\hat{Q}\left(s_t,a_t,\mathbf{w}\right)\right)\nabla\hat{Q}\left(s_t,a_t,\mathbf{w}\right).\end{aligned} \tag{3-3}$$

Polynomial functions A simple linear function approximation method is polynomial function (PF), where the linear representations of states form the feature vector $\phi(s,a)$. The weight vector $\mathbf{w}$ and the feature vector $\phi(s,a)$, together form a polynomial that computes $\hat{Q}(s,a,\mathbf{w})$ according to (3-2) [43]. After each encountered state-action pair, the weights of $\mathbf{w}$ are updated according to an update strategy such as SGD in (3-3).

There are design choices to be made when constructing a polynomial approximation. Imagine a system with two states (s_1,s_2). Different feature vectors of $\phi(s,a)$ can be defined, giving the approximation method different characteristics. A simple vector could be $\phi(s,a)=(s_1,s_2)^\top$, which provides low computational costs but also low adaptive properties to accurately fit a more complex action-value function. A more complex, higher-dimensional feature vector such as $\phi(s,a)=\left(1,s_1,s_2,s_1s_2,s_1^2,s_2^2,s_1s_2^2,s_1^2s_2,s_1^2s_2^2\right)^\top$ can provide more adaptive properties to better fit an action-value function, though with the expense of higher computational costs.

Overall, PF's are seen as a simple and fast approximation method. Similar to linear regression methods, the magnitude of each weight expresses the significance of each feature, which can provide valuable information on which features have a significant contribution to the approximation. There are hardly any parameters to be tuned and only a weight vector $\mathbf{w}$ has to be stored. However, this does come at the cost of flexibility of the function that is required to fit complex action-value functions.

Radial basis functions To describe an action-value function using RBFs, weights are assigned to Gaussian-shaped kernels located in the state-space. These kernels can be modified by adjusting the location of the center c_i or the width σ_i of the kernel. The sum of these overlapping Gaussians approximate the action-value function. The values in the feature vector are represented by the distance of the state-action pair of interest to the center- c_i of each kernel and width σ_i of the kernel:

$$\phi_{\mathrm{i}}(s,a)\doteq\exp\left(-\frac{\|s-c_{\mathrm{i}}\|^2}{2\sigma_{\mathrm{i}}^2}\right) \tag{3-4}$$

These parameters can be tuned to fit the desired properties of the problem. An advantage of RBFs is that they produce a smooth approximate function that is differentiable. A disadvantage of RBFs is the computational complexity that increases when the dimension of states grows. Furthermore, manual tuning of the kernel locations and kernel width may be required for learning to be robust and efficient [43]. However, this increase in tuning parameters also increases the flexibility of the function. It can be concluded that RBFs are most suited for lower dimensional systems that require smooth, differentiable functions.

3-2-3 Nonlinear function approximation

Linear approximation methods have proven to be effective for mapping inputs to the desired outputs with little computational complexity. However, nonlinear approximation methods provide more flexibility for fitting complex functions. ANNs are a very popular nonlinear function approximation method based on the way human brains work. Finding suitable features can be a very difficult task for linear methods since prior knowledge of the system is required. An ANN does not require predefined features but can generate these features by itself by combining state information, allowing fitting on raw data [60]. Therefore, ANNs are the third type of function approximator that is used in the case study.

Artificial neural networks There exist several types of ANNs. Convolutional ANNs are most used on high-dimensional problems such as image recognition, whereas recurrent ANN are better at finding relationships between data sequences. Since approximating action-value functions is a regression problem, we will focus on using the more common feedforward ANN.

A feedforward ANN is a network that takes in input values and maps it to output values. It consists of an input layer, an output layer, and one or more "hidden layers". Furthermore, no loops between layers are present such that the output can influence the input. Each layer consists of one or multiple so-called perceptrons, as shown in Figure 3-2. Perceptrons take in a sum of their input values $\phi_{\text{i}}(s, a)$ weighted by a matrix $\mathbf{w}$ to compute an output z [60]:

$$z = \sum_{i=1}^{l} w_i \phi_i + w_0 = \mathbf{w}^T \phi, \tag{3-5}$$

where w_0 is the bias and l denotes the number of inputs of the perceptron. The output z can then be fed to a (non)linear function called an activation function to compute an output y which can be fed to the next layer of perceptrons. The topology of these layers of perceptrons can be seen in Figure 3-3.

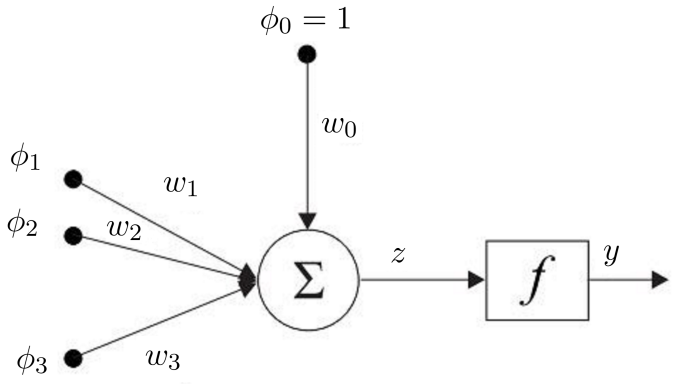

Figure 3-2: The perceptron [60]. A weighted sum of feature values and a bias term is fed to an activation function. The output of the perceptron is fed to other perceptrons in the neural network.

Activation functions map input values z to output values y between 0 and 1 to indicate to what degree a data point belongs to a certain feature. The output values are then multiplied by a certain weight and fed to the next layer. There is a large variety of activation functions with different purposes. However, since we prefer using gradient-based update methods for our case study, differentiable activation functions such as sigmoid functions, logistic functions, or rectified linear unit functions are preferred.

By adjusting the weights of the inputs for each perceptron, features that contribute more to the action-value function approximation can be weighed more heavily, thereby improving the overall approximating performance of the ANN. Like other algorithms, the weights are adjusted in the direction that maximizes an objective function using, for example, SGD. In most cases, the objective function is an error function derived from the performance of the system on a labelled set of training data. A popular training technique is backpropagation, where forward passes compute the activation values from input values and backward passes adjust the weights between the perceptrons by computing the partial derivative of the objective function with respect to the weights. Proper initialization of the weights can strongly increase the efficiency of the backpropagation algorithm.

The number of perceptrons per layer and the number of layers determine the adaptive properties of the ANN. However, as stated in [12], an ANN with one hidden layer containing a large enough number of perceptrons can approximate any continuous function to any degree of accuracy. Nevertheless, this does not mean that every ANN can learn any continuous function. In practice, multiple hidden layers are required to train the network since there is often a lack of prior knowledge to design a network as described above. By increasing the number of layers and number of perceptrons in a layer, features appropriately representing the problem can be created without relying on manually selected features. However, adding too much complexity to the ANN can cause generalization issues such as overfitting, as is explained in Section 3-2-4.

Having that said, ANN suffers from some complexities since multiple tuning parameters have to be adjusted, overfitting issues can occur and training the ANN can take a long time. Another disadvantage of using ANN is that it is a so-called black-box method. This means it is unclear why it is updating some weights more than others, whereas, such information can be extracted more easily from other feature-based methods.

Experience replay and double-Q learning As further discussed in Subsection 3-2-4, instability of the ANN can occur due to the noisy and correlated update sequence of the ANN imposed by the nature of MDP's. In [34], a deep Q-network algorithm is presented, where the action-value function of a Q-learning algorithm is approximated by an ANN. Two adjustments were made to deal with the instability caused by correlation problems.

The first adjustment is using experience replay, where state-action-reward pairs are stored in a replay memory. When training the ANN, experience replay is used to fit the action-value function not only on the new state-action-reward pair but also on a batch of samples that are uniform randomly drawn from the replay memory. This decorrelates the trajectories of the algorithm and stabilizes the updates.

The second adjustment is using double-Q learning, where the algorithm incorporates two approximated action-value functions [64]. The weight vector $\mathbf{w}^{\mathrm{l}}$ of one approximated action-value function is updated every time step, called the local action-value function $Q^{\mathrm{l}}(s,a)$. The

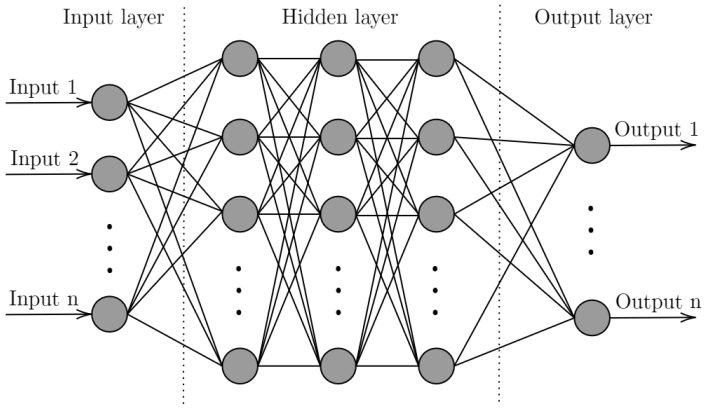

Figure 3-3: Schematic overview of an ANN including input layers, hidden layers, and output layers.

weight vector $\mathbf{w}^{\mathrm{t}}$ of the other action-value function, called the target action-value function $Q^t(s,a)$, is set equal to the local weight vector $\mathbf{w}^{\mathrm{l}}$ after each certain number of time steps. Another option is to update the target Q-function each time step with a small factor τ of the local value function, called soft target update as

$$\mathbf{w}^{\mathrm{t}} = \mathbf{w}^{\mathrm{l}}\tau + \mathbf{w}^{\mathrm{t}}(1-\tau). \tag{3-6}$$

Only the target action-value function $Q^{\mathrm{t}}(s,a)$ is used for estimating future values, removing the correlation between sequential states since samples are taken from the replay memory randomly and not sequentially. Introducing this second action-value function is like introducing a "fixed" target for bootstrapping to follow instead of following a moving target.

Deep Q-networks have proven to be very successful on a wide range of problems. In [39, 23], the deep Q-network presented in [34] is enhanced with a Dyna architecture, allowing for more data efficiency, making the algorithm suitable for problems with a human in the loop. This algorithm is extended even further in [56], where the quality of the simulated experience is enhanced by discriminating between real-world samples and simulated samples.

3-2-4 Problems with action-value function approximation

As mentioned previously, problems can arise when applying function approximation methods to RL. Two of them will be discussed in this section.

Instability Some off-policy RL methods using function approximators can show unstable behaviour where the function approximator diverges away from the true action-value function when using bootstrapping [61]. The combination of off-policy methods, function approximators, and bootstrapping causes instability when several states are represented by the same element in the column vector $\mathbf{w}$ [43]. In general, when updating the action-value of the current state-action pair with the estimates of the future state-action pairs (bootstrapping), the corresponding element of $\mathbf{w}$ is updated. However, this also influences the approximated

action-values of the surrounding states represented by this element of $\mathbf{w}$. When using off-policy RL methods, this can cause the target policy to divert from the behaviour policy and cause instability in the action-value function [43].

Several approaches have been suggested in the literature to overcome the convergence difficulties that arise when integrating function approximation, bootstrapping, and off-policy learning [43]. An example is a gradient TD-learning method that estimates the expected update vector of the TD(0) algorithm and performs stochastic gradient descent on its TD-error [59]. In [58], off-policy, Dyna-style architecture is successfully extended by linear function approximators using a novel way of prioritized sweeping. In Subsection 3-2-3, experience replay and double Q-learning were introduced as a successful remedy for instability occurring when using ANN as a function approximator.

Overfitting and underfitting Another problem that occurs when using function approximation, especially ANN, is over- and underfitting. The essence of overfitting is that the function uses more features than is justified by the data. In this case, the approximation method has used an extensive amount of complexity to fit every data point, including outliers and noise of the training dataset. When the function approximator is validated on a test set, the approximating performance decreases since the function approximator is attempting to represent the noise and outliers of the training set. In other words, using too many features, the approximation method may try very hard to fit the training data, but may fail to generalize to new data. Underfitting is the opposite of this phenomenon when the underlying structure of a dataset is not adequately captured due to missing features. Examples of overfitting and underfitting can be seen in Figure 3-4. So-called regularization methods such as early stopping of training, cross-validation of a training set with a test set, or discouraging using too many datapoints by including a penalty in the objective function can prevent overfitting [1]. For ANN in particular, the dropout method is effective, where multiple ANN are trained, each missing randomly selected features [54].

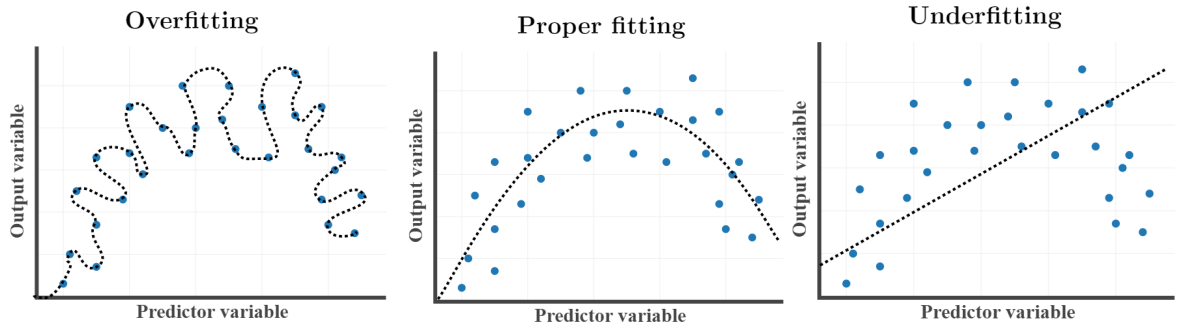

Figure 3-4: Example of overfitting, proper fitting, and underfitting on a simple data set.

3-3 Planning and learning: Dyna

Considering the speed advice problem, the experience required to train the RL algorithm on is human-generated. Consequently, this experience is scarce and must be used efficiently. Integrated planning and learning algorithms are known for being sample efficient. Learning

algorithms refer to algorithms that interact with the environment by experimentation to collect samples to improve the approximation of the optimal action-value function. Planning algorithms use a model of the environment to generate simulated experience to either improve the action-value function, which is called background planning, or plan the best policy at that time step maximizing the return, called decision-time planning. Planning- and learning algorithms both estimate the action-value function at every time step. Planning, acting, and learning methods can therefore be combined to interact circularly as in Figure 3-5 [43]. The agent acts, observes the new state and reward, and updates the action-value for that state-action pair (direct RL). It also uses this newly acquired environment information to improve a model of the environment. After this, simulated experience is generated using this model of the environment to update the action-value function or plan the next optimal move based on the return of the simulated experience (indirect RL). This process is repeated until a terminal state is reached. Moreover, new state-action pair samples from real-world experience are used to improve the model accuracy.

The states that are simulated in the planning phase are not always chosen at random since it can be favourable to focus on states near high-yielding states or states which values have changed most significantly during the last update. This is called prioritized sweeping. It can also be desirable to focus on updating states that are likely to be encountered in the current policy, called trajectory sampling. By introducing indirect RL, more use is made out of the limited experimental experience than when using only direct RL. However, direct RL methods are simpler and require less computational power than indirect RL. Two popular integrated planning and learning methods are Monte Carlo tree search and Dyna [43]. Monte Carlo tree search is a decision-time planning and learning method simulating many trajectories from the current state to the terminal state, selecting high-yielding states in a search tree [11, 43]. Due to its long online computation time, this algorithm is more suited for games such as chess and Go where longer online computation times are available than for a traffic environment. Due to its faster convergence properties, we will focus on Dyna in this thesis [24, 35, 57].

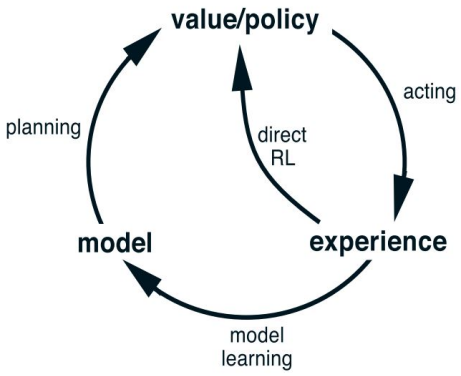

Figure 3-5: Planning, acting and model learning steps circularly applied [43].

Dyna-Q Dyna-Q is an integrated planning and learning algorithm that combines direct and indirect RL. As can be seen in Figure 3-6, the agent receives information on the current state and its reward from the environment. It then uses a direct RL algorithm such as Q-learning to determine the next best action [43]. The computation time left in the time step is then

used for the model learning process. During this process, a random previously visited state is sampled and a randomly chosen previously performed action at this state is taken. Then, the action-values of the visited states in the simulation are updated in the same way as they would be updated in real experience with the environment using Q-learning. This process is repeated until the simulation time is exceeded. During the next real time step, the direct RL process can use an action-value function that has been improved by simulated experience resulting in faster convergence to an optimal policy. In [24], prioritized sweeping is used to select which states to update during the planning phase, increasing convergence speed.

Dyna-Q is an effective addition to direct RL since it exploits the remaining simulation time to improve the action-value function. However, this is under the assumption an accurate model of the environment is known. When a model is unknown or inaccurate the agent bases its policy on an environment that is not equal to the real environment, thus deteriorating performance as opposed to applying direct RL only. Moreover, Dyna-Q is also prone to time-varying environments. For example, in the speed advice problem, the cyclist preferences might change. Direct RL would adapt to this change, whereas, Dyna-Q might still be committed to the model of the old cyclist preferences. Therefore, Dyna-Q is not considered in the remainder of this thesis.

Dyna-2 When it is not desired to let the action-value function learn from simulated experience, because the environment model is inaccurate or it changes over time, Dyna-Q may not work. If one still wants to benefit from simulated experience, a specific implementation of Dyna can be used, called Dyna-2 [52]. Dyna-2 uses two action-value functions, a permanent-, and a transient action-value function. The permanent action-value function only learns from real experience, whereas the transient action-value function uses simulated experience to provide an improved local approximation of the action-value function. The transient action-value function is reset to the permanent value function after a desired number of time steps.

Dyna-2 has proven to be an effective algorithm since it enhances the agent's policy by providing additional local information. This even allows the algorithm to achieve high performance in problems where there are not enough resources to approximate an accurate action-value function. An example of this is the game Go where the dimensions of the state-action space are too large to search for an optimal policy. In [52], Dyna-2 is still able to achieve a high performance in the game of Go by providing a local action-value function generated by simulated experience. Depending on the number of required simulated episodes after each real time step, Dyna-2 can get computationally demanding. In Chapter 5, a case study will investigate if Dyna-2 can be applied to the speed advice problem.

3-4 Conclusions

In this chapter, the concept of RL was introduced. RL is a powerful tool for solving MDP problems such as the speed advice problem. The field of research of RL knows many different algorithms. However, only those that are relevant for the case study were presented in this chapter. Being a popular RL algorithm, Q-learning provides a solid base for more advanced RL algorithms when adding function approximation or combined planning- and learning methods.

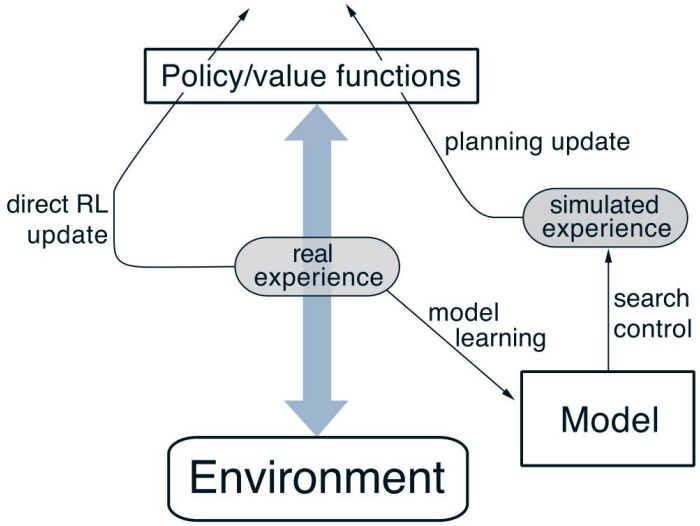

Figure 3-6: The general Dyna Architecture. Real experience, passing back and forth between the environment and the policy, affects policy and action-value functions in much the same way as does simulated experience generated by the model of the environment [43].

The large state-space of the speed advice problem requires the use of function approximation to overcome the curse of dimensionality. PFs, provide a mathematically simple way of action-value function approximation making the method easy to understand and computationally inexpensive. RBFs have more expressive properties compared to PFs, coming paired with higher computational demand, and more complex tuning parameters. ANNs are a nonlinear function approximation method that possesses different properties than the other selected approximation methods. ANNs provide very expressive properties enabling an accurate fit to complex value functions. Unlike linear function approximators, ANNs are able to combine state information to create significant predictive features. Nevertheless, ANNs are computationally demanding, difficult to analyse due to their black-box nature, and have more tuning parameters than the other selected function approximators.

Planning and learning algorithms provide a solution to dealing with the scarce training data imposed by the speed advice problem. Dyna aims to improve the action-value function by simulating experience using a model that is obtained from real experience. Since the speed advice problem concerns learning from humans in real traffic scenarios it is more difficult to obtain an accurate model of this environment than a clearly predefined set of rules such as a game of chess. If the action-value function were to be trained on an inaccurate model, this could deteriorate the performance. However, for variations of Dyna, such as Dyna-2, a reasonably accurate model can still be useful. Dyna-2 provides the agent with an enhanced local approximation of the action-value function instead of affecting the action-value function permanently.

Chapter 4

Traffic environment modeling

To carry out experiments with speed advice RL algorithms, a model of the environment is required to train and test the algorithms on. It is our goal to let the model resemble a realistic traffic scenario based on Dutch standards. First, a model of the traffic light situation will be presented. Secondly, the cyclist model is discussed. Finally, the reward functions will be defined.

4-1 Traffic light model

There is a wide range of intersection configurations, characterised by the legal traffic directions of the intersection, the number of lanes, and the number of traffic lights present. Multiple traffic light phases B^{i} can be defined, where i indicates the corresponding traffic light phase number. Each phase B^{i} has a different configuration of the traffic light colours of the traffic lights at the intersection. Therefore, each traffic light phase B^{i} grants passage for one or more traffic directions and blocking others for a certain phase time t^{p}. The majority of traffic lights in the Netherlands is actuated, meaning that their phase time t^{p} is influenced by incoming traffic. Since this incoming traffic is stochastic, the phases of the traffic light are stochastic as well. Nevertheless, historical data of the traffic lights can provide average phase durations which can be used as additional information. In our model of the traffic light, phase termination probability $P^{\mathrm{t}}(B^{\mathrm{i}}|t^{\mathrm{p}})$ and phase switching probability $P^{\mathrm{s}}(B^{\mathrm{j}}|B^{\mathrm{i}})$ of an intersection define the traffic flow dynamics of the intersection. A more detailed overview of the traffic light model is presented in Section 5-1. Overall, the state-space of the traffic light model S_k^{l} at discrete time step k can be defined as

$$S_k^{\mathrm{l}} = (B^{\mathrm{i}}, t^{\mathrm{p}}). \tag{4-1}$$

4-2 Cyclist model

This section will discuss the model of the cyclist. Starting with the kinematics of the longitudinal movement of the cyclist cycling towards the traffic light. After this, the cyclist preferences, and how the cyclist's compliance to the speed advice is modelled will be discussed.

4-2-1 Cyclist kinematics

To simplify the model of the cyclist, we are considering a one-dimensional longitudinal movement of the cyclist towards the traffic light. A schematic overview of the trajectory of the cyclist moving towards the intersection is described in Figure 4-1. In this figure $x^{\max}$ is the distance between the starting point and the endpoint of the simulation trajectory, x^{tl} indicates the position of the traffic light, and x_k describes the current position of the cyclist.

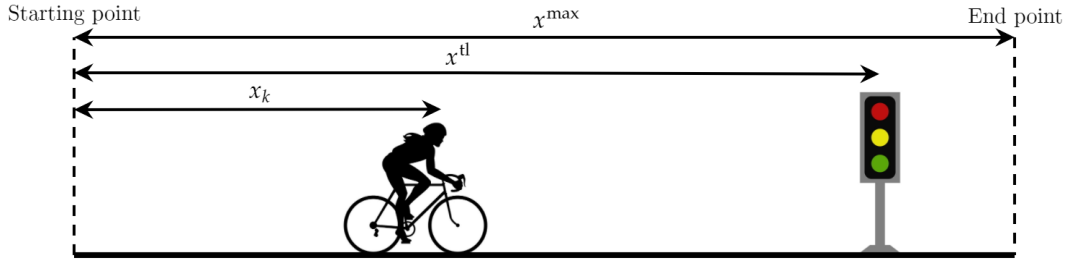

Figure 4-1: Schematic overview of the longitudinal trajectory of the cyclist moving towards an intersection.

The kinematics of the cyclist are described by the following discretized set of deterministic equations:

$$\begin{aligned} t &= k\Delta t, \\ x_{k+1} &= x_k + v_k\Delta t + \frac{1}{2}u_k\Delta t^2, \\ v_{k+1} &= v_k + u_k\Delta t, \end{aligned} \tag{4-2}$$

where time t(s) is described by discretization time step Δt(s) and time step counter k. The position of the cyclist with respect to the traffic light x_k(m) is affected by the speed of the cyclist v_k(m/s) and its acceleration u_k(m/s^2). Due to physical properties of the system of the cyclist the following constraints are imposed:

$$\begin{aligned} 0 \leq x_k &\leq x^{\max}, \\ 0 \leq v_k &\leq v^{\max}, \\ -u^{\max} \leq u_k &\leq u^{\max}, \end{aligned} \tag{4-3}$$

Where $x^{\max}$ is the distance between the initial starting position and the terminal position of the cyclist in the simulation. Speed of the cyclist v_k is constrained by the maximum speed of the cyclist $v^{\max}$ and zero since we do not consider scenarios that the cyclist changes in longitudinal direction. $u^{\max}$ the maximum acceleration of the cyclist during a normal situation.

Table 4-1: The action space $\mathcal{A}$ of the agent in the speed advice problem is defined by five different speed advises.

	Symbol	$\tilde{u}_{k+1}$
Aggressive acceleration	aa	$u^{\max}$
Normal acceleration	na	$\frac{1}{2}u^{\max}$
Maintain current speed	mcs	0
Normal deceleration	nd	$-\frac{1}{2}u^{\max}$
Aggressive deceleration	ad	$-u^{\max}$

However, the constraints of u_k are not enforced during an emergency brake situation. This will be further discussed in Subsection 4-2-3. The state-space of the cyclist model at discrete time step k can be defined as

$$S_k^{\mathrm{c}} = (v_k, x_k) \tag{4-4}$$

It is the agent's goal to control the cyclist's acceleration by giving speed advice. This can be done by suggesting a reference speed for the cyclist to follow. However, even when discretizing the set reference speeds, this still leaves a large action space for the agent to choose from. To decrease the action space size, the agent has been given a set of five actions that can be used to control the rate of acceleration with respect to the current speed by suggesting an acceleration $\tilde{u}_{k+1}$. Therefore, the agent's action space $\mathcal{A}$ is defined as in Table 4-1. It must be noted that the suggested acceleration $\tilde{u}_{k+1}$ may not be equal to the cyclist's real acceleration u_{k+1}. The compliance rate of the cyclist is discussed further in Subsection 4-2-3.

4-2-2 Cyclist preferences and reward function

Each cyclist is different and has specific preferences. A great advantage of RL as opposed to other algorithms is that it can learn from its user and adapt its policy. To include cyclist preferences in the model, several performance criteria have been selected that influence the cyclist behaviour. This subsection will describe which performance criteria are used and how these are represented in the reward function. The performance criteria presented in [16] were used as a starting point.

1. **Safety**: The main contribution of the speed advice is increasing the probabilities of catching green traffic lights. This increases traffic flow, reduces travel time, and reduces energy consumption. At all costs, red-light crossings should be avoided. If a cyclist crosses a red light the agent receives a negative reward $-R^{\mathrm{s}}$. However, in practice a cyclist will always overrule speed advice leading towards crossing a red light by initiating an emergency brake. When a green light is crossed without prior use of an emergency brake, the agent receives a positive reward R^{s}. There are also situations that the green traffic light phase suddenly terminates and the cyclist is unable to come to a full stop before the traffic light. To reduce the state-space in our traffic light model, no yellow lights were used. However, in reality, these situations would offer cyclists a safety margin by presenting a yellow light. Yellow light situations receive a small positive reward $\frac{1}{2}R^{\mathrm{s}}$. This can be described in a function $F^{\mathrm{s}}(s, s', a)$ as

$$F^{\mathrm{s}}\left(s, s', a\right) = \begin{cases} -R^{\mathrm{s}} & \text{if } (s, s') \in \mathcal{S}^{r} \\ R^{\mathrm{s}} & \text{if } (s, s') \in \mathcal{S}^{g} \\ \frac{1}{2}R^{\mathrm{s}} & \text{if } (s, s') \in \mathcal{S}^{y} \\ 0 & \text{else} \end{cases} \tag{4-5}$$

where $\mathcal{S}^{\mathrm{r}}$, $\mathcal{S}^{\mathrm{g}}$, and $\mathcal{S}^{\mathrm{y}}$ are the subspaces of the state-space corresponding to states where the cyclist crosses a red, green or yellow traffic light without prior use of an emergency brake. As will be described in Subsection 4-2-3, red lights are never crossed since a cyclist will overrule the speed advice.

2. **Minimizing energy consumption**: A similar energy model as in [16] is used to describe the energy consumption of the cyclist. The total energy consumption E_{cyc} can be described by the energy associated with the acceleration E_{ac}, rolling resistance E_{rr}, aerodynamic drag E_{ad}, and road slope E_{rs} by

$$E_{\mathrm{cyc}}(v, u) = \underbrace{\left(m_{\mathrm{c}} + m_{\mathrm{w}}\right) uv}_{E_{\mathrm{ac}}} + \underbrace{C_{\mathrm{rr}} mgv}_{E_{\mathrm{rr}}} + \underbrace{0.5\rho v \left(v + v^{\mathrm{w}}\right)^2 C_{\mathrm{ad}} A_{\mathrm{f}}}_{E_{\mathrm{ad}}} + \underbrace{mgve}_{E_{\mathrm{rs}}}, \tag{4-6}$$

where m_c, m_w, and m (kg) are the masses of the cyclist, the rotational mass of the wheels, and the combined mass of the bicycle and cyclist respectively, g (kg/m^2) is the gravitational acceleration, C_{rr} and C_{ad} are coefficients representing rolling resistance of the tires and the aerodynamic drag of the cyclist, ρ (kg/m^3) is the air density, v^{w} (m/s) is the speed of the headwind, A_{f} (m^2) is the frontal surface of the cyclist, and e is the road slope. In the reward function, energy consumption factors can be described by

$$F^{\mathrm{e}}(s, a) = \Delta t \frac{E_{\mathrm{cyc}}(v, u)}{E_{\max}(v^{\max}, u^{\max})} \tag{4-7}$$

where $E_{\max}$ is used for normalisation of the function as

$$\begin{aligned} E_{\max} = &\left(m_{\mathrm{c}} + m_{\mathrm{w}}\right) u^{\max} v^{\max} + C_{\mathrm{rr}} mgv^{\max} \\ &+ 0.5\rho v^{\max} \left(v^{\max} + v^{\mathrm{w}}\right)^2 C_{\mathrm{ad}} A_{\mathrm{f}} + mgv^{\max} e \end{aligned} \tag{4-8}$$

3. **Minimizing travel time**: Unnecessarily long trajectories toward a traffic light should be avoided in general. However, some cyclist value this criterion more than others and are willing to reduce travel time at the cost of other preferences such as maintaining a desired speed. Therefore, the agent receives a small negative reward $-R^{\mathrm{t}}$ at every time step spent on the trajectory towards the traffic light:

$$F^{\mathrm{t}}(s) = -R^{\mathrm{t}}. \tag{4-9}$$

The agent also receives a negative reward $-R^{\mathrm{i}}$ for idling in front of a red light:

$$F^{\mathrm{i}}(s) = \begin{cases} -R^{\mathrm{i}} & \text{if } s \in \mathcal{S}^{\mathrm{i}} \\ 0 & \text{else} \end{cases} \tag{4-10}$$

where $\mathcal{S}^{\mathrm{i}}$ is the subspace of the state-space where the cyclist is idling in front of a red light.

4. **Cycling at a desired speed**: Each cyclist has a desired speed that provides the most comfortable cycling experience. Diverting from this speed feels unnatural, can cause frustration, and requires extra physical exertion and should therefore be avoided when possible. The agent receives a negative reward that increases quadratically when diverging further away from the desired speed v^{ds} as

$$F^{\mathrm{ds}}(s,u) = -\frac{\left(v + u\Delta t - v^{\mathrm{ds}}\right)^2}{\beta}, \tag{4-11}$$

where β is used for normalisation purposes as

$$\beta = \max\left(\left(v^{\mathrm{ds}}\right)^2, \left(v^{\max} - v^{\mathrm{ds}}\right)^2\right). \tag{4-12}$$

5. **Avoiding instability**: Low speeds can cause instability of the bicycle. For some, the extra effort of balancing the bicycle is undesired. For others, especially, older cyclists, bicycle instability can be dangerous as it can cause accidents. Therefore, the agent receives a negative reward $-R^{\mathrm{l}}$ when cycling speeds lower than a certain threshold v^{l} as

$$F^{\mathrm{ls}}(s) = \begin{cases} -R^{\mathrm{l}} & \text{if } v_k \leq v^{\mathrm{l}} \\ 0 & \text{else} \end{cases} \tag{4-13}$$

High speeds are undesired as well as dangerous situation can occur with surrounding traffic, and cyclists can lose control over their steering. Similar to low speeds, high speeds are penalised when cycling faster than a threshold speed v^h by

$$F^{\mathrm{h}}(s) = \begin{cases} -R^{\mathrm{h}} & \text{if } v_k \leq v^{\mathrm{h}} \\ 0 & \text{else} \end{cases} \tag{4-14}$$

6. **Avoiding fluctuating speed advice**: As described in [28], fluctuating speed advice is not perceived as trustworthy. This can cause deterioration of the cyclist's compliance to the speed advice. Therefore, the amount of speed advice messages containing a deceleration or acceleration action should be minimized. Consequently, the agent receives a negative reward $-R^a$ when giving normal acceleration (na)- and deceleration (nd) speed advises. The agent receives a double negative reward $-2R^a$ when giving aggressive acceleration (aa)- and deceleration advises (ad) as

$$F^{\mathrm{a}}(a) = \begin{cases} -2R^{\mathrm{a}} & \text{if a = aa} \\ -R^{\mathrm{a}} & \text{if a = na} \\ -R^{\mathrm{a}} & \text{if a = nd} \\ -2R^{\mathrm{a}} & \text{if a = ad} \\ 0 & \text{else .} \end{cases} \tag{4-15}$$

To express the cyclist preferences in the RL algorithm, the rewards collected during the interaction with the environment are multiplied by a unique set of weights W. The sum of these weighted rewards results in the reward function as

$$R = F^{\mathrm{s}}W^{\mathrm{s}} - F^{\mathrm{e}}W^{\mathrm{e}} - F^{\mathrm{t}}W^{\mathrm{t}} - F^{\mathrm{i}}W^{\mathrm{i}} - F^{\mathrm{ds}}W^{\mathrm{ds}} - F^{\mathrm{l}}W^{\mathrm{l}} - F^{\mathrm{h}}W^{\mathrm{h}} - F^{\mathrm{a}}W^{\mathrm{a}} \tag{4-16}$$

The weights of reward function R are predefined by the user and can be adapted manually. These weights could even be learned by interaction, yet this is deferred to future studies.

4-2-3 Compliance

As mentioned in Subsection 4-2-1, the agent aims to control the user's acceleration by sending speed advice messages. Nevertheless, the acceleration of the cyclist is affected by a stochastic process by the cyclist's personal preferences and physical limitations. This variable compliance rate means that the actual cyclist's acceleration is not always equal to the speed advice given. One can imagine that a cyclist is more likely to adhere to an aggressive acceleration advice when cycling below its desired speed than when the cyclist is already cycling at a very high speed. As can be seen in Figure 4-2, our model of the environment captures this cyclist behaviour by introducing five zones Z with different compliance rates based on the speed of the cyclist. Each zone has a set of specific compliance rules. As an illustrative example, let us assume a cyclist is currently cycling above its desired speed and is therefore in zone Z_+. The agent takes an action suggesting an aggressive acceleration $\tilde{u}_{k+1} = u^{\max}$. In that case, the compliance rate of the cyclist is modeled by making a random choice between three accelerations with the following probability distribution:

$$\begin{aligned} &P(u_{k+1} = u^{\max} | Z = Z_+ \cup \text{action} = \text{aa}) = 0.7 \\ &P(u_{k+1} = 0.75u^{\max} | Z = Z_+ \cup action = aa) = 0.2 \\ &P(u_{k+1} = 0.5u^{\max} | Z = Z_+ \cup action = aa) = 0.1 \end{aligned} \tag{4-17}$$

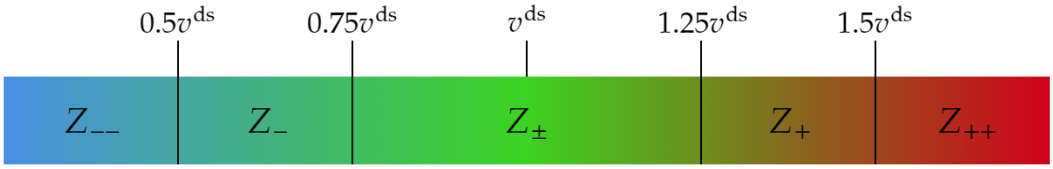

Figure 4-2: Depending on the speed of the cyclist with respect to its desired speed v^{ds}, the cyclist can be categorised in five different speed zones, all causing different compliance rates.

These probability distributions are defined for each zone-speed combination possible and can be found in Table 4-2.

Besides cyclists not fully complying with the speed advice due to imposed discomfort, the cyclist's behaviour can also be influenced by safety reasons. This occurs when the cyclist receives speed advice that, in the cyclist's observation, leads to crossing an occupied intersection with a red light. A cyclist can then choose to initiate an emergency brake. To model this behaviour, the estimated time of arrival t_k^{eta} is computed by

$$t_k^{\mathrm{eta}} = \frac{x^{\mathrm{tl}} - x_k}{v_k}. \tag{4-18}$$

Table 4-2: Cyclist's compliance rate expressed in the probability of the future cyclist acceleration influenced by the current speed zone Z and action a taken by the agent.

	a=aa	a=na	a=mcs	a=nd	a=ad
Z_{--}	$P(u_{k+1} = u^{\max}) = 1$	$P(u_{k+1} = 0.5u^{\max}) = 1$	$P(u_{k+1} = 0) = 0.7$ $P(u_{k+1} = 0.1u^{\max}) = 0.2$ $P(u_{k+1} = 0.25u^{\max}) = 0.1$	$P(u_{k+1} = -0.5u^{\max}) = 0.5$ $P(u_{k+1} = -0.3u^{\max}) = 0.3$ $P(u_{k+1} = -0.1u^{\max}) = 0.2$	$P(u_{k+1} = -1u^{\max}) = 0.5$ $P(u_{k+1} = -0.75u^{\max}) = 0.3$ $P(u_{k+1} = -0.5u^{\max}) = 0.2$
Z_{-}	$P(u_{k+1} = u^{\max}) = 1$	$P(u_{k+1} = 0.5u^{\max}) = 1$	$P(u_{k+1} = 0) = 0.85$ $P(u_{k+1} = 0.1u^{\max}) = 0.1$ $P(u_{k+1} = 0.25u^{\max}) = 0.05$	$P(u_{k+1} = -0.5u^{\max}) = 0.7$ $P(u_{k+1} = -0.3u^{\max}) = 0.2$ $P(u_{k+1} = -0.1u^{\max}) = 0.1$	$P(u_{k+1} = -1u^{\max}) = 0.7$ $P(u_{k+1} = -0.75u^{\max}) = 0.2$ $P(u_{k+1} = -0.5u^{\max}) = 0.1$
$Z_{\pm}$	$P(u_{k+1} = u^{\max}) = 1$	$P(u_{k+1} = 0.5u^{\max}) = 1$	$P(u_{k+1} = 0) = 1$	$P(u_{k+1} = -0.5u^{\max}) = 1$	$P(u_{k+1} = -u^{\max}) = 1$
Z_{+}	$P(u_{k+1} = 0.5u^{\max}) = 0.1$ $P(u_{k+1} = 0.75u^{\max}) = 0.2$ $P(u_{k+1} = u^{\max}) = 0.7$	$P(u_{k+1} = 0.1u^{\max}) = 0.1$ $P(u_{k+1} = 0.3u^{\max}) = 0.2$ $P(u_{k+1} = 0.5u^{\max}) = 0.7$	$P(u_{k+1} = 0) = 0.05$ $P(u_{k+1} = -0.1u^{\max}) = 0.1$ $P(u_{k+1} = -0.25u^{\max}) = 0.85$	$P(u_{k+1} = -0.5u^{\max}) = 1$	$P(u_{k+1} = -u^{\max}) = 1$
Z_{++}	$P(u_{k+1} = 0.5u^{\max}) = 0.2$ $P(u_{k+1} = 0.75u^{\max}) = 0.3$ $P(u_{k+1} = u^{\max}) = 0.5$	$P(u_{k+1} = 0.1u^{\max}) = 0.2$ $P(u_{k+1} = 0.3u^{\max}) = 0.3$ $P(u_{k+1} = 0.5u^{\max}) = 0.5$	$P(u_{k+1} = 0) = 0.1$ $P(u_{k+1} = -0.1u^{\max}) = 0.2$ $P(u_{k+1} = -0.25u^{\max}) = 0.7$	$P(u_{k+1} = -0.5u^{\max}) = 1$	$P(u_{k+1} = -u^{\max}) = 1$

Secondly, the required emergency braking time t_k^{eb} for the cyclist to come to a full stop if it complies to the agent's suggested action is computed by

$$t_k^{\text{eb}} = \frac{v_k}{u^{\text{eb}}}, \tag{4-19}$$

where u^{eb} is the cyclist's maximum deceleration during an emergency brake. The cyclist initiates an emergency brake when the following three conditions hold. First, the current traffic light phase block B^{i} must impose a red light for the cyclist. Secondly, at the next time step, the estimated time of arrival t_{k+1}^{eta} is smaller than the required braking time t_{k+1}^{eb} if the cyclist were to follow the agent's speed advice, meaning $t_{k+1}^{\text{eta}} \leq t_{k+1}^{\text{eb}}$. Finally, the cyclist must be able to come to a full stop when using an emergency brake at its current speed, meaning $t_k^{\text{eta}} \leq t_k^{\text{eb}}$. When these conditions, hold the cyclist's acceleration is described as

$$u_{k+1} = \max(u^{\text{eb}}, -v_k). \tag{4-20}$$

As will be further discussed in Chapter 3, RL is a sample-based algorithm, allowing for it to automatically take the uncertainty imposed by the compliance of the cyclist into account by learning which actions are profitable and which are not. However, this does mean that it is difficult to create a model of the cyclist's response behaviour. This is vital for the performance of an algorithm using integrated planning and learning. This will be further discussed in Subsection 5-3-1.

4-3 Focus feature

As will be further described in Section 3-2-2, it can be beneficial to have an extra feature that integrates information of different states to assess combinations of states instead of only assessing individual state-values. The focus feature t^{f} is an extra feature that aims to combine the information of existing states. This is achieved by combining cyclist information with traffic light information to determine if the current cyclist's trajectory is focused on a green traffic light phase. If t^{f} is negative, the cyclist should accelerate. A positive t^{f} would indicate that the cyclist arrives at the traffic light before it turns green and should decelerate.

To calculate t^{f}, the estimated time of arrival t_k^{eta} is computed by (4-18), thereby combining the position and the speed of the cyclist. Historical traffic light data can be exploited to

obtain average durations of each traffic light phase block B^i. This information can be used to construct a vector $\mathbf{t}^{\text{ttg}}$ containing the average times until next green traffic light phases. From $\mathbf{t}^{\text{ttg}}$, the next green traffic light phase that is closest to the estimated time of arrival t_k^{eta} can be found by

$$t^{\text{ttg}} = \arg\min_{t^{\text{ttg}}}(\mathbf{t}^{\text{ttg}} - t^p - t^{\text{eta}}). \tag{4-21}$$

Using the traffic light phase time t^{p} and the time to green t^{ttg} that is closest to t^{eta}, the focus time t^{f} can be determined by

$$t_k^{\text{f}} = t^{\text{ttg}} - t^p - t^{\text{eta}}. \tag{4-22}$$

4-4 Conclusions

In this chapter, a model was defined to train and test RL algorithms to solve the speed advice problem. The traffic light model incorporates the stochasticity imposed by the actuated traffic lights. The cyclist model uses a simple kinematic model to describe the longitudinal movement of the cyclist. However, more complexity is added to this model by taking cyclist preferences and compliance into account. An additional feature is added to the state-space to combine state-space information and exploit historical traffic light data. Together, the following state-action space at discrete time step k can be defined as

$$\begin{aligned} S_k &= (B_k^{\text{i}}, t_k^{\text{p}}, v_k, x_k, t_k^{\text{f}}), \\ A_k &= (\text{aa, na, mcs, nd, ad}). \end{aligned} \tag{4-23}$$

Chapter 5

Case study

To test the performance of the different RL algorithms, simulations will be performed in the environment as described in Chapter 4. Python will be used for the modelling and simulation of these experiments. Three different case studies will be conducted, each assessing different RL techniques. In case study I, experiments are conducted aimed at investigating the effectiveness of different function approximators. Case study II focuses on the application of using Dyna. Case study III will combine features from previous case studies and aims to present an RL algorithm that is practically implementable. In Section 5-1, some general environment parameters and assumptions are defined. Sections 5-2 to 5-4 present the experiments and results of the case studies. Finally, conclusions are drawn in Section 5-5.

5-1 Set-up

The method of modeling the traffic environment and the cyclist including its preferences has been addressed in Chapter 4. In this section, the parameters of these models are defined, performance measures are presented, and benchmarks are introduced.

5-1-1 Parameter definition

In this subsection, the parameters that are used to define the model of the traffic environment and the cyclist are defined.

Physical parameters To use the mathematical models of the environment discussed in Chapter 4, several parameters must be defined. Some general physical properties of the traffic environment can be found in Table 5-1. The values of these parameters have been taken from [38].

Table 5-1: Value of physical parameters of the traffic environment.

x^{tl} (m)	$x^{\max}$ (m)	ρ (kg/m^3)	C_{ad}	A_{f} (m^2)	C_{rr}	e (rad)	v^{w} (m/s)	Δt (s)	g (m/s^2)
180	200	1.226	1.2	0.616	0.008	0	0	1	9.81

Traffic light parameters In Figure 5-1 (left), the intersection that will be used in the case study is illustrated. It has been chosen to use a common intersection with six different traffic movements. The phase blocks B^{i} in the flowchart in Figure 5-1 (right) describe which combination of movements have a green phase at the same time. This figure also illustrates the possible transition directions between the traffic light phase blocks. The probability of the transitions between these blocks $P_t(B^j|B^{\mathrm{i}})$ can be found in Table 5-3.

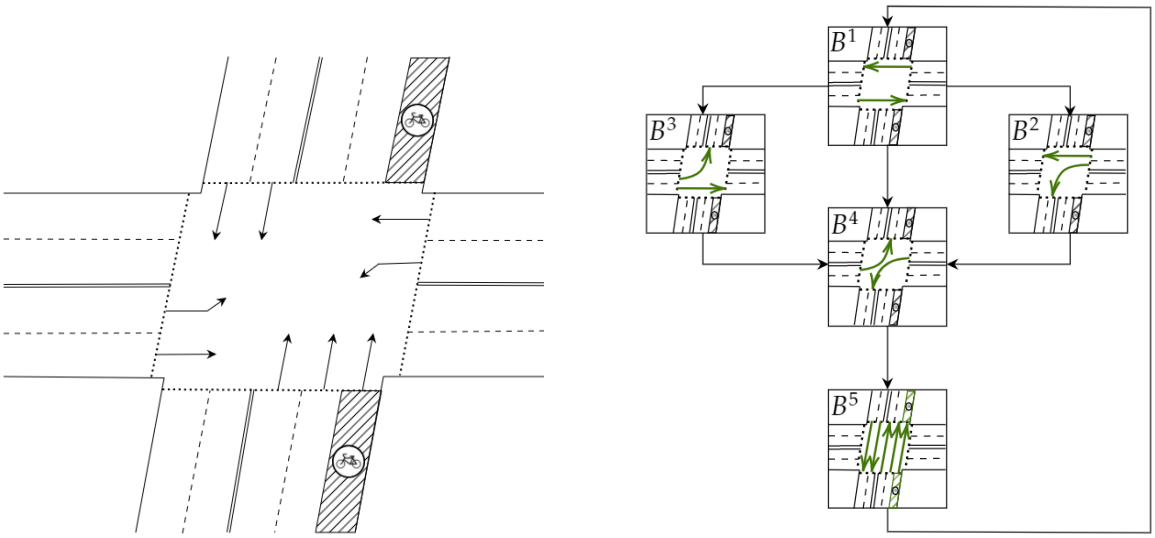

Figure 5-1: Schematic overview of traffic flows on an intersection (left). Green phase flowchart of the intersection (right).

Table 5-2, describes the phase termination probability of a traffic flow block $P_t(B^{\mathrm{i}}|t^{\mathrm{p}})$, where t^{p} indicates the time this block has been in a green phase. During the case study, the cyclist of interest will always be coming from the same direction and only receives a green light for traffic light phase block B^5.

Table 5-2: Time dependent traffic light phase termination probabilites of each traffic flow block.

	$t^{\mathrm{p}}=1$	$t^{\mathrm{p}}=2$	$t^{\mathrm{p}}=3$	$t^{\mathrm{p}}=4$	$t^{\mathrm{p}}=5$	$t^{\mathrm{p}}=6$	$t^{\mathrm{p}}=7$	$t^{\mathrm{p}}=8$	$t^{\mathrm{p}}=9$	$t^{\mathrm{p}}=10$	$t^{\mathrm{p}}=11$	$t^{\mathrm{p}}=12$
$P^{\mathrm{t}}(B^1\|t^{\mathrm{p}})$	0	0	0	0	0.3	0.7	1	1	1	1	1	1
$P^{\mathrm{t}}(B^2\|t^{\mathrm{p}})$	0	0	0	0.5	1	1	1	1	1	1	1	1
$P^{\mathrm{t}}(B^3\|t^{\mathrm{p}})$	0	0	0	0.5	1	1	1	1	1	1	1	1
$P^{\mathrm{t}}(B^4\|t^{\mathrm{p}})$	0	0	0	0	0	0.3	0.7	1	1	1	1	1
$P^{\mathrm{t}}(B^5\|t^{\mathrm{p}})$	0	0	0	0	0	0	0	0.2	0.4	0.6	0.8	1

Cyclist model parameters As each cyclist has different preferences, the parameters of the cyclist kinematics and the weights of the reward function are different for each cyclist. Therefore, three cyclist types are defined: an average cyclist, an athlete cyclist, and an old cyclist. The physical properties of the three different cyclist types are reflected in Table 5-4. These

Table 5-3: Traffic light phase transition probabilities $P^s(B^{\mathrm{i}}|B^j)$.

	B^1	B^2	B^3	B^4	B^5
$P^s(B^{\mathrm{i}}\|B^1)$	0	0.25	0.25	0.5	0
$P^s(B^{\mathrm{i}}\|B^2)$	0	0	0	1	0
$P^s(B^{\mathrm{i}}\|B^3)$	0	0	0	1	0
$P^s(B^{\mathrm{i}}\|B^4)$	0	0	0	0	1
$P^s(B^{\mathrm{i}}\|B^5)$	1	0	0	0	0

physical properties are based on information from [38]. The average cyclist represents a person of average fitness, that uses the speed advice to increase its probability to catch green lights whilst maintaining a comfortable cycling experience. The athlete cyclist is not afraid of cycling at high speeds and is willing to invest more energy than average cyclists to save travel time by catching green lights. The old cyclist uses the speed advice to reduce full stops at traffic lights that force the old cyclist to mount his or her bicycle. The preferences of the different cyclist types are reflected in the weights of the reward function as described in Subsection 4-2-2. A higher weight suggest more emphasis on the corresponding feature. These weights can be found in Table 5-5.

Table 5-4: The physical properties of three different cyclist types.

	m^{c} (kg)	m^{b} (kg)	v^{ds} (m/s)	v^{l} (m/s)	v^{h} (m/s)	$v^{\max}$ (m/s)	$u^{\max}$ (m/s^2)	u^{eb} (m/s^2)
Average cyclist	75	20	4.0	1.0	8.0	10.0	0.7	-3.0
Athlete cyclist	85	10	6.0	1.0	12.0	15.0	1.0	-4.0
Old cyclist	65	20	3.0	1.0	6.0	8.0	0.5	-3.0

Table 5-5: The preferences of three different cyclist types reflected in weights of different properties.

	W^{s}	W^{e}	W^{t}	W^{i}	W^{ds}	W^{h}	W^{l}	W^{a}
Average cyclist	40	10	0	1	3	1	1	1
Athlete cyclist	30	5	0.2	2	1	0.5	0.5	0
Old cyclist	50	20	0	3	4	2	2	1

Episode initialization To generate realistic traffic scenarios and guarantee exploration of the environment, the environment settings of the simulation are initiated randomly. For the traffic light model, episodes are initialized by uniform randomly choosing a traffic flow block B^{i} to be green and for how long this block has already been green t^{p}. The cyclist's initial speed v_0 is uniform randomly selected on an interval around the cyclist's desired speed $[v^{\mathrm{ds}}-2, v^{\mathrm{ds}}+2]$.

Episode termination An episode terminates when the cyclist crosses the endpoint of the road section, so $x^{\max} \leq x_k$. However, the agent also stops giving speed advice after an emergency brake is initiated since the speed advice is then overruled by the cyclist. The speed advice is also overruled after the emergency brake, when the cyclist is idling in front of the traffic light, and when it is accelerating to its desired speed when a green light is given. Since the algorithm's speed advice is not obeyed until after the traffic light, there is no reason to continue the episode. Therefore, the state-action pair leading to the initiation of the emergency brake is updated as a Markov reward process. For the simulation, this means

that the cyclist finishes the episode without speed advice and rewards are given based on the cyclist's visited states and actions. The sum of the rewards R^{eb} between the emergency brake state s^{eb} at time step k^{eb} and the terminal state s^{t} at time step k^{t} are used to update the action-value of the state-action pair that led to the emergency brake using (3-1), where the reward R is defined as

$$R^{\text{eb}} = \sum_{l=k^{\text{eb}}}^{l=k^{\text{t}}} \gamma^{l-k^{\text{eb}}} R_{\text{l}}. \tag{5-1}$$

Algorithm parameter selection To guarantee an efficient learning process, the RL algorithm's parameters concerning learning and exploration must be selected carefully. During the case study, a selection process was used for each specific experiment to fulfil the needs of each unique RL algorithm. First, the initial learning rate α_0, alpha decay factor d^{α}, and minimum alpha $\alpha^{\min}$ are selected. These learning rates are chosen by finding the highest learning rate α that did not cause stability issues over the entire simulation. Once a learning rate α is properly chosen, an estimate of the amount of required training episodes can be deduced by observing where the learning curve of the algorithm converges. Once this amount of required training episodes is known, the exploration rate ϵ and the epsilon decay factor d^{ϵ} are chosen to ensure a smooth decay from initial epsilon ϵ_0 to the minimum epsilon $\epsilon^{\min}$ over the entire set of training episodes. The discount factor γ can be used to value local state-action pairs more heavily than state-action pairs that are further away. It was found that 0.98 was a suitable discount factor γ for each case study.

5-1-2 Test episode

Since the traffic environment parameters are randomly generated at each episode, one cannot compare the agent's performance between episodes. To compare the learning rate between different algorithms, every C episodes, a test episode is performed. During the test episode, the traffic parameters are always the same. The traffic light transition probability $P^{\text{t}}(B^{\text{i}}, t^{\text{p}})$ and the switching traffic light phase probability $P^{s}(B^{\text{i}}, t^{\text{p}})$ have been made deterministic for the test episodes and can be found in Appendix B. To remove the effect of random actions on the performance, the agent only takes greedy actions during the test episode. Moreover, the episode is always initiated with the same parameter values. Namely, the initial velocity v_0 is 5 m/s, the initial traffic light block B_0^{i} is B^5, and initial phase time t_0^{p} is 3 seconds.

5-1-3 Performance measures

The performance of the different RL algorithms are assessed based on the behaviour of the agent in the environment and on the requirements of the algorithm. Once a RL algorithm has been fully trained, 100 test episodes are carried out to measure its performance. These episodes are all test episodes as described in Subsection 5-1-1 to allow comparison between other RL algorithms. The behaviour of the cyclist is observed using the following five performance measures:

1. Average reward per episode

2. Green light ratio
3. Average travel time per episode
4. Average cycling speed
5. Average energy consumption per episode

The average reward and the average energy consumption per episode are computed using (4-6) and (4-16), respectively. The average travel time and average cycling speed per episode can be obtained by the mean of the corresponding state variables. The green light ratio is computed after the experiment by the percentage of episodes where a green light was caught without using an emergency brake.

Two performance measures are used for the assessment of the requirements of the algorithm:

1. Required amount of training episodes
2. Response time

First, the sample efficiency of the algorithms is tested by observing the amount of training episodes required before the learning curve of the algorithm converges to a steady-state. Secondly, the computational demand of the algorithm is tested by measuring the response time of the algorithm. The response time is defined as the time it takes to observe a new state-action pair, determine the next action to take, and update the action-value function with the new state-action pair. This is a vital performance measure since the dynamic traffic environment requires fast response times. Therefore, we will set the response time constraint to one second.

5-1-4 Benchmarks

Besides comparing the developed RL algorithms with each other, it is important to investigate how the algorithms compare with the current state of the art. Therefore, two benchmarks are introduced.

No speed advice The first benchmark assesses simulations of episodes of a cyclist that does not receive speed advice. By comparing the performance of the cyclist without speed advice with the novel methods of giving speed advice, observations can be made to what degree performance is increased. Cyclists without speed advice miss the benefits of the speed advice such as increasing the probability of catching green lights. Nevertheless, these cyclists can cruise at their desired speeds and are not bothered by energy-consuming speed advice commands. The performance of this benchmark was evaluated by running 100 test episodes of an average cyclist without speed advice and taking the averages of the performance measures found. In Table 5-6, the values of the performance measures of the cyclist without speed advice can be found.

Table 5-6: Performance of a cyclist without speed advice.

Avg. response time (s)	Avg. reward	Green ratio (%)	Avg. TTS (s)	Avg. speed (m/s)	Avg. energy (J)
-	-72.28	34.1%	52.51	3.99	2707

RL using a Q-table The second benchmark is used to determine if the developed RL algorithms are an improvement compared to the state of the art. The state of the art is in this case an RL algorithm using a Q-table to store the action-value function. The entire algorithm description is found in Appendix A. The RL algorithms that are tested in the case study aim to overcome the scarcity of training samples and the restrictions on the exploration of the environment compared to the RL algorithm using a Q-table whilst adhering to the response time constraint of one second. By comparing the developed algorithms with this benchmark algorithm it can be concluded if these algorithms are still competitive in terms of performance.

The Q-table consists of six state indices, namely: traffic phase block B^{i}, traffic light phase time t^{p}, cyclist position x_k, cyclist speed v_k, focus feature t^{f}, and action a. For each unique combination of indices, an action-value must be stored. The performance of the RL algorithm using a Q-table to store the action-value functions is strongly influenced by the discretization steps used in the Q-table. This is because the Q-table grows exponentially when decreasing the discretization step size of the six Q-table indices. Table 5-7 presents the three different discretization step size settings that are compared.

Table 5-7: Different discretization step sizes of Q-table indices. Traffic light phase block B^{i} and actions a are already discrete parameters.

	$\Delta t^{\mathrm{p}}(s)$	$\Delta x_k(m)$	$\Delta v_k(m/s)$	$\Delta t^{\mathrm{f}}(s)$
Q-table1	2	20	2	2
Q-table2	1	10	1	1
Q-table3	0.5	5	0.5	0.5

In Table 5-8 the number of simulation episodes, discount factor γ, the initial learning rate α_0, alpha decay factor d^{α}, minimum alpha $\alpha^{\min}$, initial epsilon ϵ_0, the epsilon decay factor d^{ϵ}, and the minimum epsilon $\epsilon^{\min}$ are defined using the process as described in 5-1-1. This combination of simulation parameters produced the best results for RL algorithms using a Q-table.

Table 5-8: Simulation parameters used for RL algorithms using a Q-table.

Episodes	γ	α_0	d^{α}	$\alpha^{\min}$	ϵ_0	d^{ϵ}	$\epsilon^{\min}$
50,000	0.98	0.2	1	0.2	0.5	0.9995	$1.00 \cdot 10^{-2}$

As can be seen in Figure 5-2, a strong correlation can be seen between the size of the Q-table and the convergence speed of the RL algorithms. However, it can also be concluded that over time, the RL algorithms using a smaller Q-table achieve lower returns. This can be explained by the accuracy of the action-value function, which is confirmed by the larger variation around Q-table1. Q-table3 has the smallest discretization step sizes, but also lower returns and larger uncertainty bounds than Q-table2. This could be explained by the fact that this RL algorithm requires more than 50,000 training episodes to converge. From Table 5-9, it can be concluded that Q-table2 achieves superior performance to Q-table3. This could change when even more training episodes are run. However, the current amount of training

episodes is already more than a human cyclist can ever encounter.

Table 5-9: Performance of a cyclist with speed advice generated by RL algorithms using Q-table with different discretization step sizes.

	Avg. response time (s)	Avg. reward	Green ratio (%)	Avg. TTS (s)	Avg. speed (m/s)	Avg. energy (J)
Q-table1	$1.23 \cdot 10^{-3}$	-19.17	82.0	50.83	4.03	3087
Q-table2	$1.24 \cdot 10^{-3}$	-7.50	92.5	50.67	4.07	2933
Q-table3	$1.30 \cdot 10^{-3}$	-24.33	84.3	53.67	3.88	3117

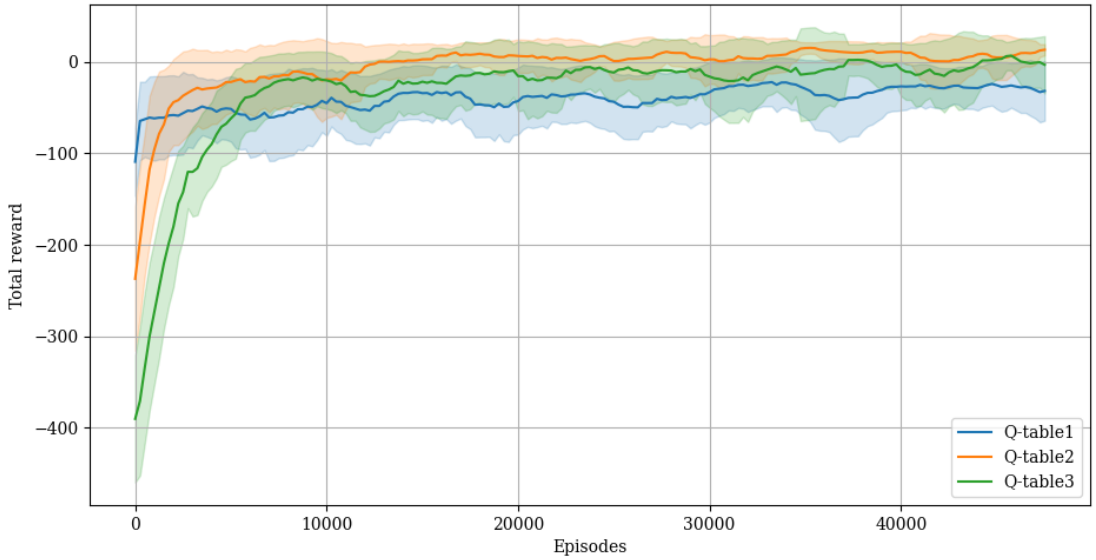

Figure 5-2: Learning curve of three RL algorithms with different Q-tables to store its action-value function.

The cyclist's position- and speed trajectories, presented in Figure 5-3, show desired behaviour of each algorithm on a test scenario. Each trajectory decelerates toward its desired speed smoothly and they all cross a green traffic light phase without an emergency brake. The trajectories corresponding to Q-table1 and Q-table3 show some fluctuations around the traffic light in order to catch a green light.

Overall, the RL algorithm using Q-table2 outperformed the other RL algorithms. Therefore, the performance of this RL algorithm will be used as a benchmark. The RL algorithm required 50,000 episodes to learn and represent an accurate action-value function. The following case studies aim to reduce this number of episodes without excessive exploration, whilst being competitive in terms of performance and meet the response time constraint.

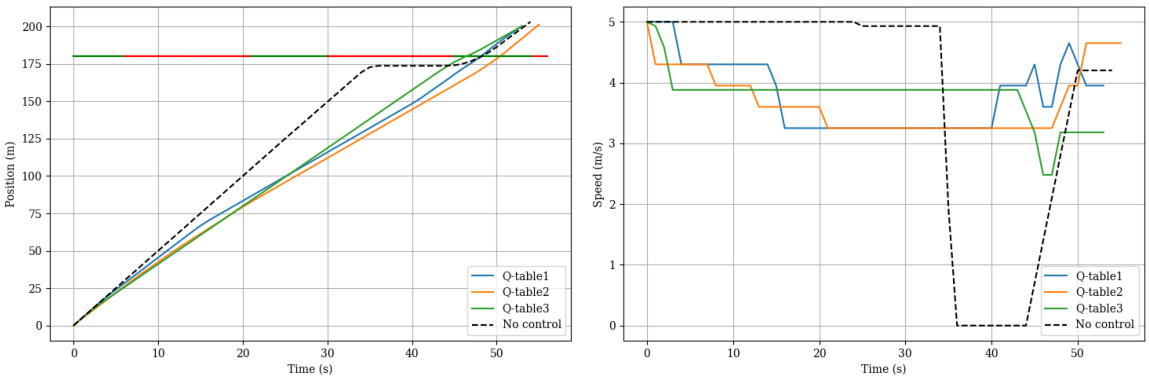

Figure 5-3: Cyclist's trajectories with speed advice generated by three different RL algorithms using a Q-table to store its action-value function, expressed in position (left) and speed (right). This simulation was generated using the average cyclist type settings and the test episode setting as described in Subsection 5-1.

5-2 Case study I: Function approximators

As motivated in previous chapters, the extension of RL algorithms with function approximators can improve the practical implementation of speed advice algorithms. They achieve this by approximating the entire state-action space using samples from a subset of the state-action space, thereby decreasing the amount of required training samples compared to tabular RL algorithms. In some cases, function approximation can decrease the performance of the algorithm and increase computational demand. Therefore, experiments with three different function approximators are carried out and their results are compared. Subsections 5-2-1, 5-2-2 and 5-2-3 present the experiments and results of using PFs, RBFs, and ANNs as function approximators respectively. Finally, Subsection 5-2-4 concludes case study I.

5-2-1 Polynomial functions

The algorithm description for RL with linear function approximators such as PFs and RBFs can be found in Appendix A. To find proper parameter settings of the PFs, several experiments are carried out. First, the number of simulation episodes, discount factor γ, the initial learning rate α_0, alpha decay factor d^{α}, minimum alpha $\alpha^{\min}$, initial epsilon ϵ_0, the epsilon decay factor d^{ϵ}, and the minimum epsilon $\epsilon^{\min}$ are properly chosen using the method described in 5-1-1. The values of these parameters can be found in Table 5-10.

The agent's actions a and the traffic light phase block B^{i} cannot be described as a continuous function. The features of each possible combination of actions and traffic light phase blocks are defined separately. These features are stored in one feature vector $\phi(s,a)$ that, together with weight vector $\mathbf{w}$, can be used to approximate Q-values according to (3-2). The features in feature vector $\phi(s,a)$ can be found in Table 5-11.

To find the proper settings for the polynomial function, three feature vectors $\phi(s,a)$ are compared. These feature vectors vary in the composition of features. These features are defined in Table 5-11. These features were selected by experimentation. Using more features than PF3 or less features than PF1 resulted in worse results.

Table 5-10: Simulation parameters used for RL algorithms using PFs as function approximator.

Episodes	γ	α_0	d^α	$\alpha^{\min}$	ϵ_0	d^ϵ	$\epsilon^{\min}$
10,000	0.98	$5.00 \cdot 10^{-3}$	0.99995	$2.50 \cdot 10^{-3}$	0.5	0.9995	$1.00 \cdot 10^{-2}$

Table 5-11: Features used in three different feature vectors $\phi(s, a)$.

	Length $\phi(s,a)$	Features
PF1	100	$[1, t_k^{\mathrm{p}}, x_k, v_k]^\top$
PF2	125	$[1, t_k^{\mathrm{p}}, x_k, v_k, t_k^{\mathrm{f}}]^\top$
PF3	200	$[1, t_k^{\mathrm{p}}, x_k, v_k, t_k^{\mathrm{f}}, v_k x_k, v_k t_k^{\mathrm{p}}, x_k t_k^{\mathrm{p}}]^\top$

As can be seen in Figure 5-4, the three RL algorithms using PFs as function approximators have a high variance during the exploration part of the learning process. The learning curves of the RL algorithms using a Q-table depicted in Figure 5-2 steadily increase their overall performance, whereas the learning curves of the RL algorithms with PFs fluctuate and have difficulty with converging to an optimum. It can be concluded that more features cause a more stable learning curve.

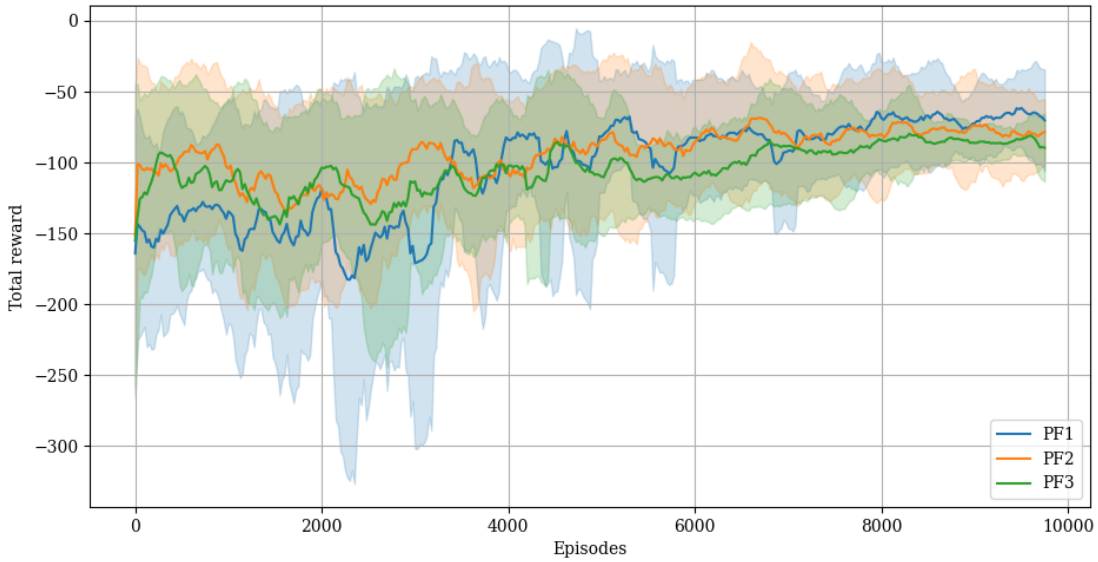

Figure 5-4: Learning curve of three RL algorithms with different PFs as functions approximator.

In Table 5-12, the overall performance of the three algorithms is presented. It can be concluded that all three algorithms slightly increase the probability of catching a green light with respect to the benchmark scenario where no speed advice is given. Moreover, the simplicity of this type of function approximator allows for a short response time. However, the average reward strongly decreases and the cyclist is forced to exert more energy whilst spending more time than the cyclist's without speed advice.

This behaviour is also reflected in Figure 5-5, where it can be seen that the cyclist rapidly decelerates towards a speed that is lower than its desired speed. Once decelerated, the cyclist does not cruise at one speed but spends energy by fluctuating between speeds. This indicates that the algorithm is unable to find a policy that can maximize different return components.

Table 5-12: Performance of a cyclist with speed advice generated by RL algorithms using PFs as function approximator.

	Avg. response time (s)	Avg. reward	Green ratio (%)	Avg. TTS (s)	Avg. speed (m/s)	Avg. energy (J)
PF1	$2.83 \cdot 10^{-4}$	-70.5	47.9	57.67	3.72	3150
PF2	$2.96 \cdot 10^{-4}$	-67.15	47.2	58.11	3.63	3265
PF3	$4.17 \cdot 10^{-4}$	-67.17	51.7	57.17	3.82	3460

This can be seen since the cyclist only focuses on one task to maximize at the current state that it is in instead of finding a strategy that is a balanced trade-off maximizing the overall return. Examples of these local strategies are deceleration imposed by current negative returns due to high energy consumption or acceleration during a green traffic light phase near the traffic light imposed by positive returns since this increases the chance of catching green.

This local optimization can be explained by the fact that PFs do not possess flexible fitting properties and only fundamental correlations can be found. Examples of these correlations are reducing speed to minimize energy consumption or accelerating when a green light is observed. However, many profound nonlinear correlations are not observed. For example, the effect of negative rewards of low speeds are cancelled out by the effect of negative rewards from high speeds. Therefore, the agent struggles to find optimal speeds around the cyclist's desired speed. Even the focus time feature t_k^{f}, designed for combining state information does not benefit the algorithm's performance since there is no linear correlation to be found in the focus time. A visualization of the correlations found by the different RL algorithms can be found in Appendix C.

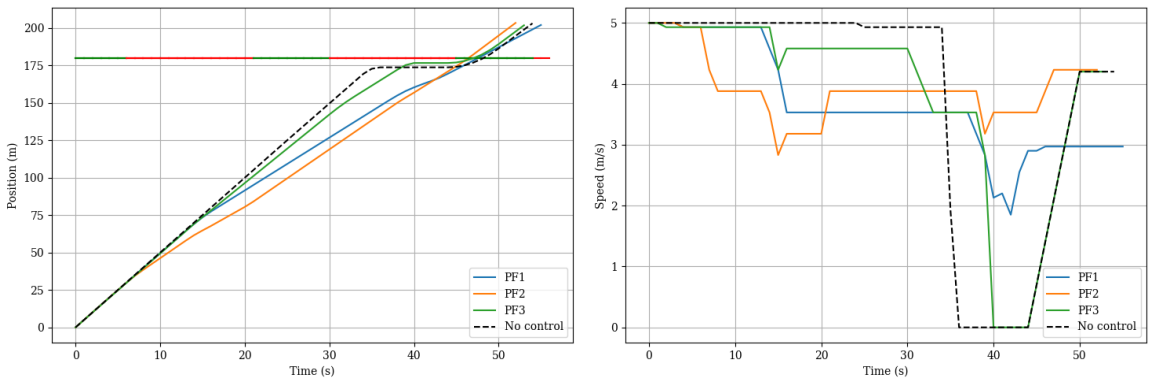

Figure 5-5: Cyclist's trajectories with speed advice generated by three different RL algorithms using PFs as function approximators, expressed in position (left) and speed (right). The colour of the line at x^{tl}=180 corresponds to the colour of the traffic light. This simulation was generated using the average cyclist type settings and the test episode setting.

5-2-2 Radial basis functions

Since RBFs have more flexible properties than PFs, experiments are carried out with RL algorithms using RBFs as function approximators. Similar to the previous experiment, proper parameter settings are chosen according to the method described in 5-1-1. The values of these parameters can be found in Table 5-13.

Table 5-13: Simulation parameters used for RL algorithms using RBFs as a function approximation method.

Episodes	γ	α_0	d^α	$\alpha^{\min}$	ϵ_0	d^ϵ	$\epsilon^{\min}$
10,000	0.98	$5.00 \cdot 10^{-3}$	0.99995	$3.50 \cdot 10^{-3}$	0.5	0.9995	$1.00 \cdot 10^{-2}$

Similar to PFs, the agent's actions a and the traffic light phase block B^{i} cannot be described as a continuous function. Therefore, a RBF is constructed for each unique action-block combination. The parameters specific for these RBFs are selected by carrying out experiments with varying values for the number of kernels n^{k} and the width of each kernel σ. According to (3-4), these parameters define the shape of the RBF. As described in Table 5-14, three different types of RBFs are selected, ranging from RBFs with few kernels and wide Gaussian shapes to RBFs with many kernels and narrow Gaussian shapes. The range of number of kernels and kernel width was determined by experimentation. Beyond this range, worse results were found.

Table 5-14: Parameters used in three types of RBFs.

	n^{k}	σ
RBF1	6	0.6
RBF2	24	0.2
RBF3	60	0.1

In Figure 5-6, it can be seen that, apart from the RL algorithm using RBF1, the algorithms using RBFs achieve higher returns than the maximum average reward of -67.15 that was obtained by the RL algorithms using PFs. It could be that RBF1 lacks the flexibility to fit the action-value function. From the learning curves of RBF2 and RBF3, it can be concluded that a lower RBF complexity results in faster learning. However, increasing a RBFs number of kernels n^{k} and decreasing kernel size σ, results in higher returns.

Table 5-15 compares the performance of the three RL algorithms. The average response time increases with the complexity of the RBF. The observation of the inferior learning curve of RBF1 is confirmed in the performance results in Table 5-15 as RBF1 scores lower results than RBF2 and RBF3 on each performance measure. There is no significant difference in performance between RBF2 and RBF3. However, the response time of RBF3 is more than double that of RBF2.

Table 5-15: Performance of a cyclist with speed advice generated by three different RL algorithms using RBFs as function approximator.

	Avg. response time (s)	Avg. reward	Green ratio (%)	Avg. TTS (s)	Avg. speed (m/s)	Avg. energy (J)
RBF1	$6.56 \cdot 10^{-4}$	-54.04	79.6	58.67	3.63	4194
RBF2	$1.44 \cdot 10^{-3}$	-18.30	90.3	50.80	4.12	3680
RBF3	$3.30 \cdot 10^{-3}$	-13.83	91.5	49.67	4.08	3482

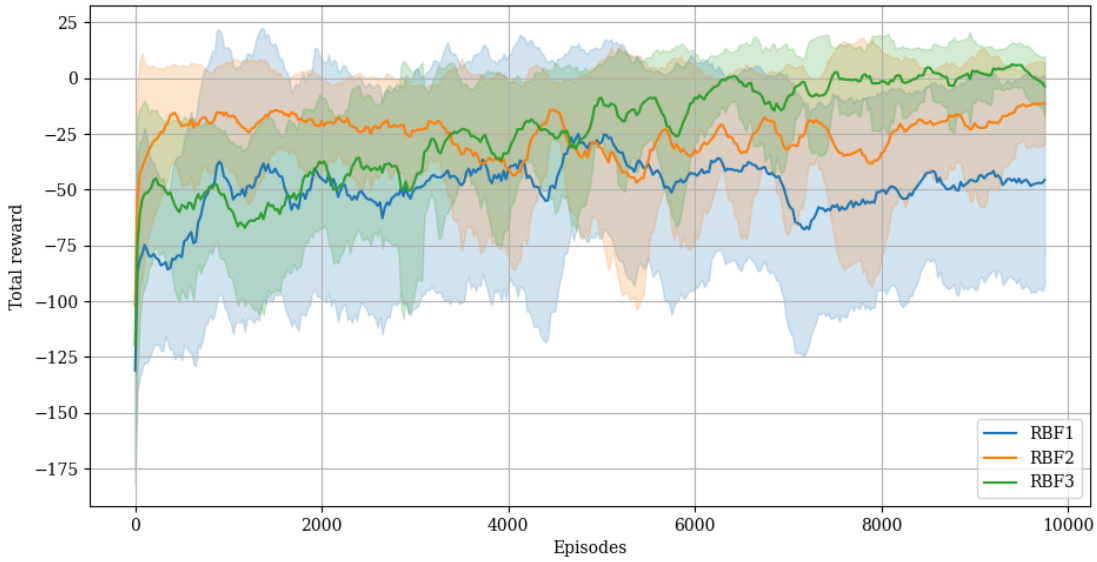

Figure 5-6: Learning curve of three RL algorithms with different RBFs as functions approximator.

In Figure 5-7 the trajectories on the test episode of the cyclists using the three RL algorithms with RBF function approximators can be seen. Around some particular reference speeds, oscillations are observed. It was found that these oscillations occur less frequently at RL algorithms using more complex RBFs. This phenomenon could be caused by states that are located in between two kernels, each favouring different actions. More kernels could provide a smooth transition between kernels. A visualization of the action-value functions represented by the RBFs in this experiment can be found in Appendix C. RBF2 and RBF3 both improve performance with respect to the benchmark where no speed advice is given. However, it can be concluded that the performance of the RBF RL algorithms is competitive compared to the benchmark Q-table algorithm, but not superior.

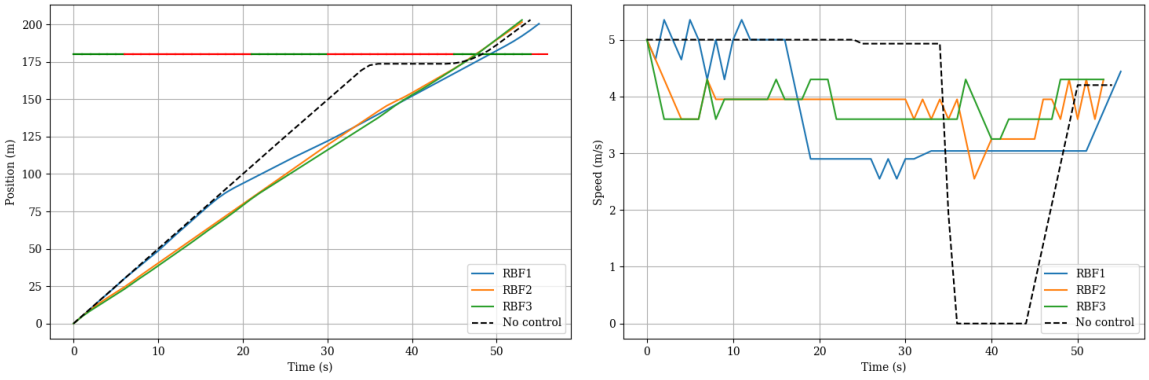

Figure 5-7: Cyclist's trajectories with speed advice generated by three different RL algorithms using RBFs as function approximators, expressed in position (left) and speed (right). The colour of the line at x^{tl}=180 corresponds to the colour of the traffic light. This simulation was generated using the average cyclist type settings and the test episode setting.

5-2-3 Artificial neural networks

As opposed to the previous experiments, in this experiment ANNs are used as a function approximator to find nonlinear correlations in the action-value function. It was noted that experience replay and double-Q learning with soft target updates do not only improve performance of the ANN, but were vital for convergence. Therefore, these techniques are applied to all ANNs in this experiment. The algorithm description of RL algorithms using ANNs as function approximators can be found in Appendix A. Some extra parameters are introduced, namely the soft target update factor τ, experience replay batch size, the maximum experience replay buffer size, the optimization method, the loss function used for the backpropagation algorithm, the activation function used in all the perceptrons, and the amount of input- and output dimensions. The dimensions of the input vector are a combination of all the scaled physical parameters as described in Table 5-1 and a one-hot encoded vector representing the traffic light phase block B^{i} and the action a. The output vector of the ANN is a vector of Q-values for each possible action. It is worth noting that due to the sample efficiency of ANN, only 300 episodes were required to achieve convergence of the RL algorithm.

Table 5-16: Simulation parameters used for RL algorithms using ANNs as function approximator.

Episodes	γ	α_0	d^{α}	$\alpha^{\min}$	ϵ_0	d^{ϵ}	$\epsilon^{\min}$
300	0.98	$1.00 \cdot 10^{-3}$	1	$1.00 \cdot 10^{-3}$	0.5	0.99	$1.00 \cdot 10^{-2}$

τ	batch size	Max. buffer size	Optimizer	Loss function	Activation function	Input dims.	Output dims.
$1.00 \cdot 10^{-2}$	64	$1.00 \cdot 10^{5}$	Adam	Mean squared error	Rectified linear unit	9	5

The difference between the ANN function approximators will lie in the topology of the network. As can be seen in Table 5-17, three different ANNs ranging from small networks, with few perceptrons per hidden layer, to large networks with many perceptrons per hidden layer. It was found that ANNs with smaller networks were not able to correctly fit the action-value function and ANNs with larger networks were too computationally demanding without adding extra performance.

Table 5-17: Number of perceptrons per hidden layer used in three types of ANNs.

	No. perceptrons hidden layer 1	No. perceptrons hidden layer 2
ANN1	32	32
ANN2	128	128
ANN3	256	256

Figure 5-8 presents a steep, upward learning curve for all three RL algorithms. It can be concluded that the more complex ANNs can find significant correlations in the action-value function faster than the less complex ANNs. The learning curves are also stable and have small variations compared to the previous experiments. This could be explained by the fact that the ANNs train on batches of samples instead of single sample points. It can be seen that the more complex ANNs are most stable and have the smallest variation.

Table 5-18 illustrates that the RL algorithms using ANNs as function approximators show high performance on all performance measures. This type of RL algorithms does not only beat the benchmark where no speed advice is given, but it also outperforms the benchmark using a Q-table in terms of performance. However, this does come at the cost of an increase

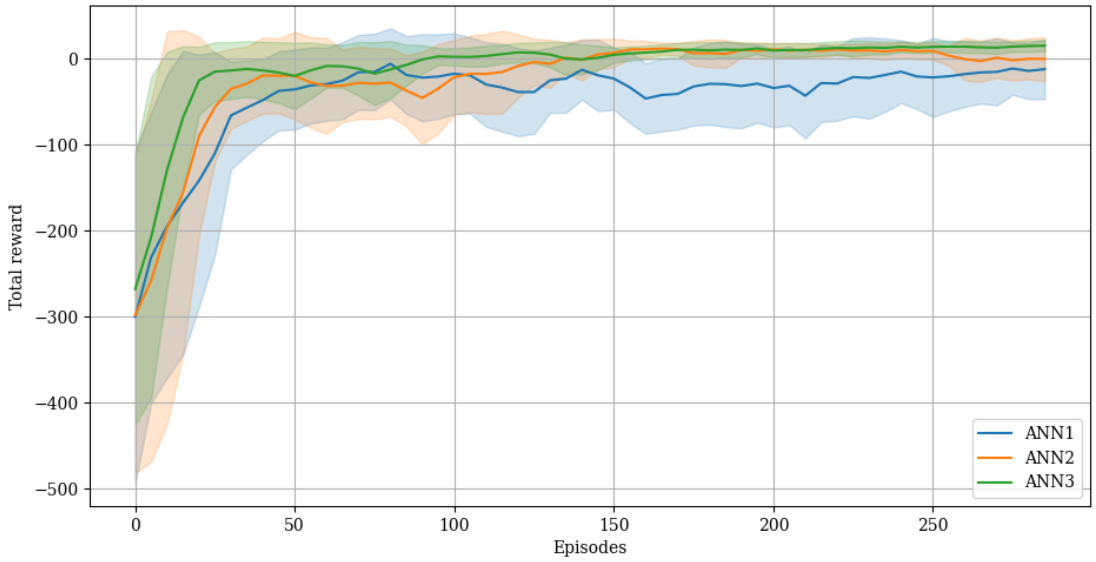

Figure 5-8: Learning curve of three RL algorithms with different ANNs as functions approximator.

of the average response time. It must be noted that this average response time is still well below the response time constraint of one second.

Table 5-18: Performance of a cyclist with speed advice generated by RL algorithms using ANNs as function approximator.

	Avg. response time (s)	Avg. reward	Green ratio (%)	Avg. TTS (s)	Avg. speed (m/s)	Avg. energy (J)
ANN1	$3.83 \cdot 10^{-2}$	-29.13	80.2	53.52	3.94	3225
ANN2	$3.97 \cdot 10^{-2}$	-9.33	93.7	51.41	4.01	2798
ANN3	$4.15 \cdot 10^{-2}$	-1.27	98.5	50.04	4.05	2643

The high performance of the three algorithms is also reflected in the behaviour that can be observed in Figure 5-9. All three algorithms are able to follow multiple strategies at different locations on the simulated trajectory, allowing for a smooth and energy-efficient experience.

Using ANNs as function approximators for RL algorithms to give speed advice to cyclist has proven to work very effectively. Moreover, these methods show high potential to reduce the amount of required training episodes since only 300 episodes were used in this experiment. Between the three RL algorithms there is a significant difference in performance. ANN3 clearly outperforms the remaining RL algorithms.

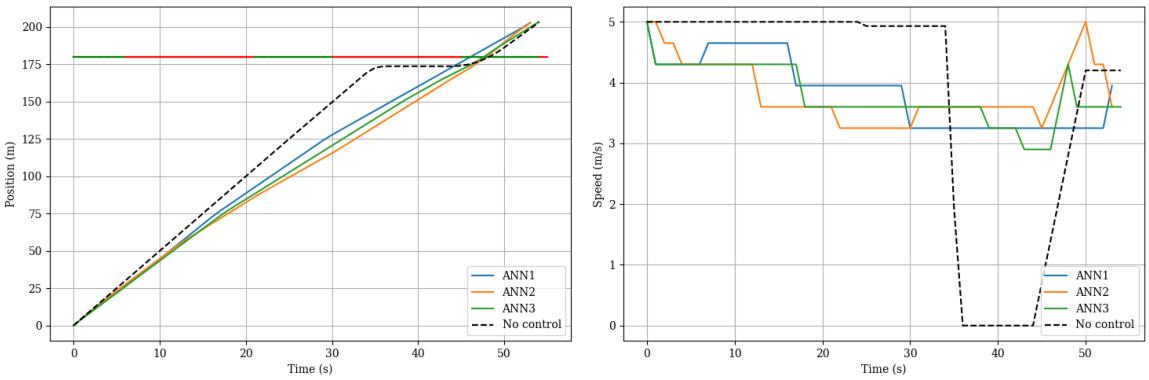

Figure 5-9: Cyclist trajectories with speed advice generated by three different RL algorithms using ANNs as function approximators, expressed in position (left) and speed (right). The colour of the line at x^{tl}=180 corresponds to the colour of the traffic light. This simulation was generated using the average cyclist type settings and the test episode setting.

5-2-4 Conclusions

In case study I, the effectiveness of three RL algorithms with different function approximators were applied to the speed advice problem and compared with each other and the predefined benchmarks. It can be concluded that each function approximator has its own characteristics and that some are more suited for the speed advice problem than others.

PFs have proven themselves as a simple, understandable, and computationally cheap method of function approximation. However, the performance of this method on the speed advice problem was low on all performance measures and is not seen as a viable option for real-world application. This behaviour can be assigned to the method being constrained to finding only linear correlations in the action-value function.

RBFs can provide more flexibility than PFs. This method of function approximation outperformed both benchmarks on some performance measures, but there were still some anomalies detectable. For example, oscillations occur around some reference speeds and the algorithms experience difficulty converging to an optimal policy.

The previous methods were linear approximation methods that are constraint by a limited flexibility in their feature selection. ANNs are nonlinear function approximators that enable the construction of complex features that combines different state information. By applying double-Q learning and experience replay, the algorithms present a steep, yet stable learning curve. The performance of the speed advice is not only superior to the other function approximation methods but also the two benchmark algorithms.

As can be seen in Figure 5-10, the RL algorithms using ANNs and RBFs as function approximators are competitive with the Q-table benchmark. From Figure 5-11, it can be concluded that the RL algorithm using ANN as a function approximator provides smooth speed advice that is desired by a human user. Table 5-19, confirms these findings and shows that ANNs outperform all other function approximation methods and also the benchmarks. Moreover, this method only requires 300 training episodes, thereby setting a large step to the practical implementation of RL algorithms for giving speed advice to cyclist.

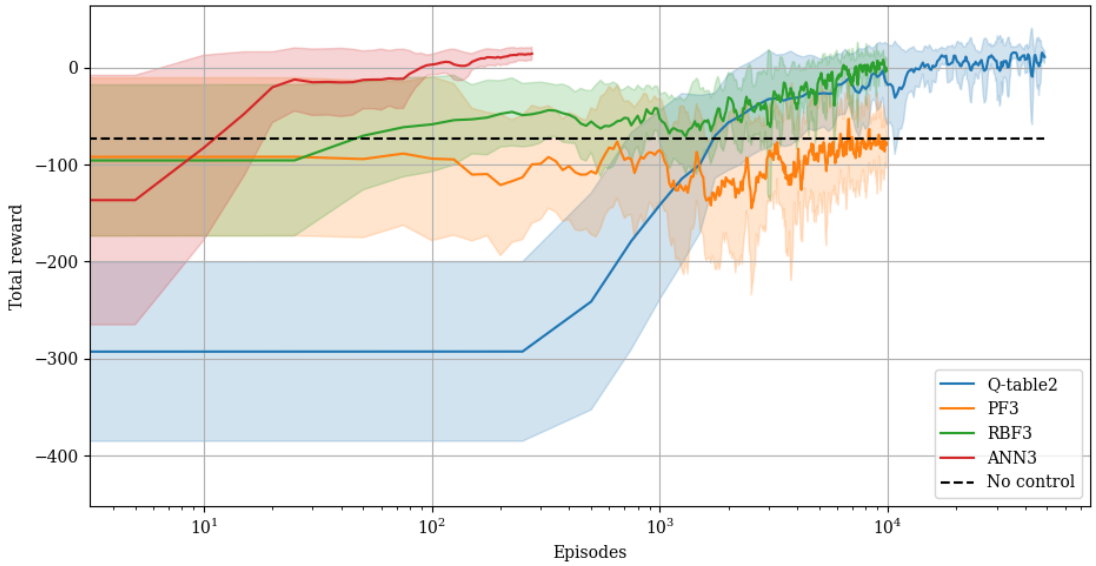

Figure 5-10: Learning curve of the three different RL algorithms extended with functions approximators and the two benchmarks.

Table 5-19: Performance of a cyclist with speed advice generated by RL algorithms using different function approximators.

	Training episodes	Avg. response time (s)	Avg. reward	Green ratio (%)	Avg. TTS (s)	Avg. speed (m/s)	Avg. energy (J)
No control	-	-	-72.28	34.1	52.51	3.99	2707
Q-table2	50,000	$1.24 \cdot 10^{-3}$	-7.50	92.5	50.67	4.07	2933
PF3	10,000	$4.17 \cdot 10^{-4}$	-67.17	51.7	57.17	3.82	3460
RBF3	10,000	$3.30 \cdot 10^{-3}$	-13.83	91.5	49.67	4.08	3482
ANN3	300	$4.15 \cdot 10^{-2}$	-1.27	98.5	50.04	4.05	2643

5-3 Case study II: Dyna

Case study I demonstrated the effectiveness of the application of function approximation on the practical implementation of RL algorithms for the speed advice problem. The second case study aims to investigate the effects of Dyna on the performance of the RL algorithm. The additional simulated experience can further reduce the amount of required training episodes. Moreover, Dyna can increase the performance of the speed advice in unknown environments by exploiting a model of the cyclist's response. Before analyzing the results of case study II, the cyclist response model that is used in the Dyna algorithm is discussed.

5-3-1 Cyclist response model

The RL algorithms that will be used in case study II require a model of how the cyclist responds to the speed advice given by the agent, called the cyclist response model (CRM). The response of the cyclist is a stochastic process which is different for each cyclist. Therefore, no CRM is known beforehand and hence must be learned. There are multiple methods of mapping the current state of the cyclist and the speed advice provided by the agent to a prediction of the acceleration of the cyclist. A sample-based method that stores cyclist response samples for each state-action pair visited in a table provided satisfying results. The

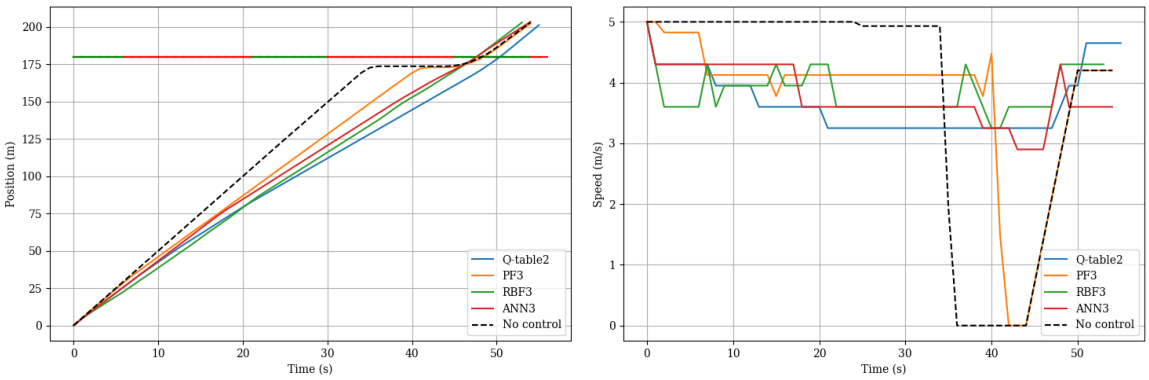

Figure 5-11: Cyclist trajectories with speed advice generated by three different RL algorithms extended with functions approximators and the two benchmarks, expressed in position (left) and speed (right). The colour of the line at x^{tl}=180 corresponds to the colour of the traffic light. This simulation was generated using the average cyclist type settings and the test episode setting.

prediction of the response of the cyclist to the speed advice is generated by sampling from this table. Since the predicted cyclist responses are generated by real samples, the distribution of the CRM matched with the real cyclist model. When the CRM encounters a state-action pair where no samples are available, the CRM assumes full compliance of the cyclist. To do this, an important assumption is made, namely that the maximum acceleration and deceleration of the cyclist are known beforehand. This information can be used to generate full compliance speed advice responses of the cyclist.

In reality, the cyclist response is influenced by a combination of states and actions. However, this would require the CRM to store samples of each possible state-action combination. Since this would drastically increase the table size, another assumption was made that the current speed of the cyclist and the speed advice given influence the cyclist's response the most. Therefore, the table only stores speed-action information to reduce the table size. Moreover, the size of the table of the sample-based CRM is also determined by the discretization step Δv of the speed of the cyclist.

When choosing a discretization step Δv for the table of the CRM, a trade-off must be made between the predictive performance of the CRM and the number of episodes required to fill the table. Therefore, three CRM's with different discretization steps Δv of 1, 0.1, and 0.01 are tested. From Table 5-20, it can be concluded that decreasing Δv increases the amount of required episodes to fill the table for more than 95% drastically. However, it cannot be concluded what the effects of Δv are on the performance of the algorithm. Therefore, experiments are carried out with all three CRM's. Supportive figures of the relation between table size and the required episodes to fill the CRM and the accuracy of the acceleration distribution can be found in Appendix D.

5-3-2 Dyna

The focus point of this case study are the two characteristics of Dyna that are believed to form obstacles for the practical implementation of Dyna in RL speed advice algorithms. First, the requirements of the model of the cyclist's response to the speed advice are assessed.

Table 5-20: Three different cyclist response models in the form of tables with different discretization step size $\Delta v(m/s)$. Smaller $\Delta v(m/s)$ results in a smaller standard deviarion σ but also more episodes to fill the table for more than 90 $\%$.

	Δv (m/s)	Required episodes (>95%)
CRM1	1	87
CRM2	0.1	835
CRM3	0.01	6235

As mentioned in Subsection 5-3-1, the cyclist response model (CRM) is used to predict the cyclist's acceleration based on its current speed and the action taken by the agent. This is achieved by storing real speed-action samples in a table and sampling during simulated experience. As Table 5-20 presents, many training episodes are required to obtain an accurate model of the cyclist response to the agent's speed advice. This first experiment investigates the relationship between the degree of completeness of the table of the CRM and the performance of the algorithm. Secondly, the increase in average response time is tested. In previous experiments, the average response time constraint of one second was never violated. However, due to the additional computation time imposed by the simulated experience, the average response time is found to increase significantly. To assess only the effect of Dyna on the performance of the RL algorithm and not the effect of function approximation, for each experiment in this case study a Q-table is used to describe the action-value function of the RL algorithm. The algorithm description of RL with Dyna using a Q-table can be found in Appendix A.

As described in Chapter 3, Dyna can make use of two types of action-value functions. A permanent action-value function, that only learns from real experience, and a transient action-value function that uses simulated experience to create a better local approximation of the action-value function. Combined permanent- and transient action-value functions learn their foundation using real experience and incorporate local information using simulated experience. To observe the effect of Dyna, different combinations of action-value functions are used. As can be seen in Table 5-21 and Table 5-22, when different action-value functions are applied, this results in different simulation parameters for real episodes and simulated episodes.

Table 5-21: Simulation parameters for real experience used for RL algorithms using a Q-table and Dyna.

	γ	α_0	d^{α}	$\alpha^{\min}$	ϵ_0	d^{ϵ}	$\epsilon^{\min}$
Transient	-	-	-	-	0	0	0
Permanent	0.98	0.2	1	0.2	0	0	0
Permanent + Transient	0.98	0.2	1	0.2	0	0	0

Table 5-22: Simulation parameters for simulated experience used for RL algorithms using a Q-table and Dyna.

	γ^s	ϵ_0^s	d^{ϵ^s}	$\epsilon^{\min^s}$	α_0^s	d^{α^s}	$\alpha^{\min^s}$
Transient	0.98	0.5	0.995	0.01	0.2	1	0.2
Permanent	-	-	-	-	-	-	-
Permanent + Transient	0.98	0.5	0.995	0.01	0.2	1	0.2

Experiment A: Influence of the cyclist response model First, the influence of the discretization step size of Δv of the CRM on the performance of the RL algorithm is assessed.

This is done by comparing three different RL algorithms where each RL algorithm uses a CRM with a unique discretization step size Δv, as described in Table 5-20. To investigate only the effect of the CRM, the action-value functions of the RL algorithms are only transient action-value functions. This means that these action-value functions are only defined by simulated experience from the CRM. The performance of these three algorithms using different CRM's are compared by simulating 10 test episodes for each RL algorithm.

It is expected that small CRM's can be filled with fewer episodes, giving them an advantage compared to larger CRM's in the first few episodes. However, larger CRM's are more accurate once they are filled with samples from a higher number of episodes due to the smaller discretization step size of Δv. To study this effect, this experiment is repeated for different amounts of episodes to have filled the tables of the CRM's beforehand, namely 0, 10, 100, and 1000 fill episodes. It must be noted that, since these algorithms use only transient action-value functions, these episodes have only been used to fill the table of the CRM, not to train the RL algorithm. In this experiment, the average response time constraint is neglected and 1000 simulated episodes per real time step of the cyclist are generated.

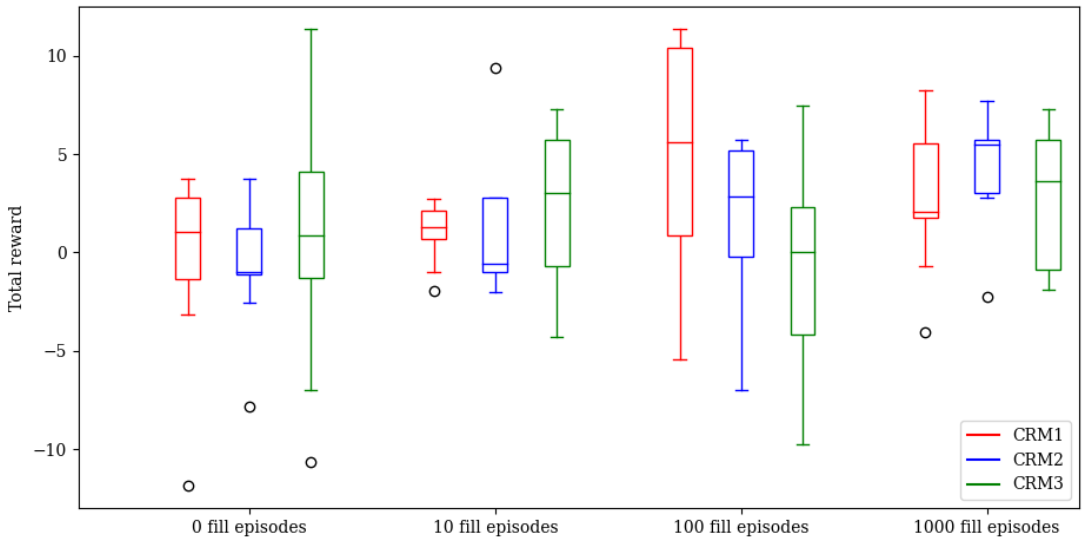

Figure 5-12: Boxplots of three different RL algorithms using Dyna with different CRM's. The total reward of each algorithm was assessed after 0, 10, 100, and 1000 training episodes to fill the table of the CRM.

First of all, Figure 5-12 shows that RL algorithms using Dyna are capable of finding an optimal policy in an unknown environment when a simple CRM is available with a coarse discretization step size of Δv and without any experience. Secondly, it can be concluded that the size of the table used for the CRM has no significant effect on the performance of the RL algorithm. This can be explained by the fact that the compliance is modelled in only 5 different zones, as explained in Chapter 4. However, in reality a cyclist's compliance will change continuously with its speed which might require CRM tables with a smaller Δv. Thirdly, it can be concluded that the number of episodes that are used to fill the tables of

the CRM's are not having a significant impact either. This can be due to the high number of simulated episodes per time step n^s that were used. When there is no sample present for a particular speed-action pair, full compliance of the cyclist is assumed. Apparently, the direction and a rough estimate of the magnitude of the cyclist response are already enough for Dyna to generate a near to optimal policy. Finally, it is found that the average response time constraint is violated drastically. Using 1000 simulated episodes per time step increased the average response time from $1.24 \cdot 10^{-3}$ seconds per time step to $1.07 \cdot 10^{2}$ seconds per time step, thereby violating the constraint of 1 second.

Experiment B: Influence of the amount of simulated episodes per time step As became clear from the previous experiment, the average response time constraint was violated by a large margin. This second experiment will assess the impact of the amount of simulated episodes per time step n^s on the performance and on the required average response time of the RL algorithm using Dyna.

In this experiment, three RL algorithms are constructed, where one algorithm uses only a permanent action-value function, one uses only a transient action-value function, and one combines both types of action-value functions. Since it was found that the discretization step Δv of the CRM's table and the number of episodes that are used to fill this table did not have a significant effect on the performance of the RL algorithm, CRM1 is used in this experiment. Moreover, each RL algorithm was trained for 1000 episodes using the real environment model. As can be seen in Figure 5-2, conventional RL algorithms using a Q-table required 50,000 episodes to converge to an optimal policy. Therefore, it can be concluded that 1000 episodes is not enough to train a RL algorithms using a Q-table to find an optimal policy. For permanent action-value functions the training episodes are used for learning action-values and describing them in the Q-table. For transient action-value functions, these training episodes are used to fill the tables of the CRM. For the combination of permanent- and transient action-value functions, the training episodes are used for both purposes. Similar to Experiment A, 10 test episodes were simulated for all three RL algorithms with different action-value functions. This experiment was repeated for each different amount of simulated episodes per timestep value n^s in Table 5-23.

Table 5-23: Average response time (s) for different action-value function types for different numbers of simulated episodes per time step.

n^s	Response time (s)		
	Transient	Permanent	Permanent + Transient
1	$5.65 \cdot 10^{-2}$	$1.24 \cdot 10^{-3}$	$8.42 \cdot 10^{-2}$
5	$4.09 \cdot 10^{-1}$	$1.24 \cdot 10^{-3}$	$6.16 \cdot 10^{-1}$
10	$1.08 \cdot 10^{0}$	$1.24 \cdot 10^{-3}$	$1.13 \cdot 10^{0}$
50	$5.33 \cdot 10^{0}$	$1.24 \cdot 10^{-3}$	$6.32 \cdot 10^{0}$
100	$1.53 \cdot 10^{1}$	$1.24 \cdot 10^{-3}$	$1.72 \cdot 10^{1}$
500	$5.52 \cdot 10^{1}$	$1.24 \cdot 10^{-3}$	$5.68 \cdot 10^{1}$
1,000	$1.07 \cdot 10^{2}$	$1.24 \cdot 10^{-3}$	$1.19 \cdot 10^{2}$

Several conclusions can be drawn from Figure 5-13. First, as expected, it was found that the RL algorithm using a permanent action-value function performs poorly after only 1000 training episodes. Secondly, the total reward that is obtained by the RL algorithm using only a transient action-value function increases with the amount of episodes simulated at each time

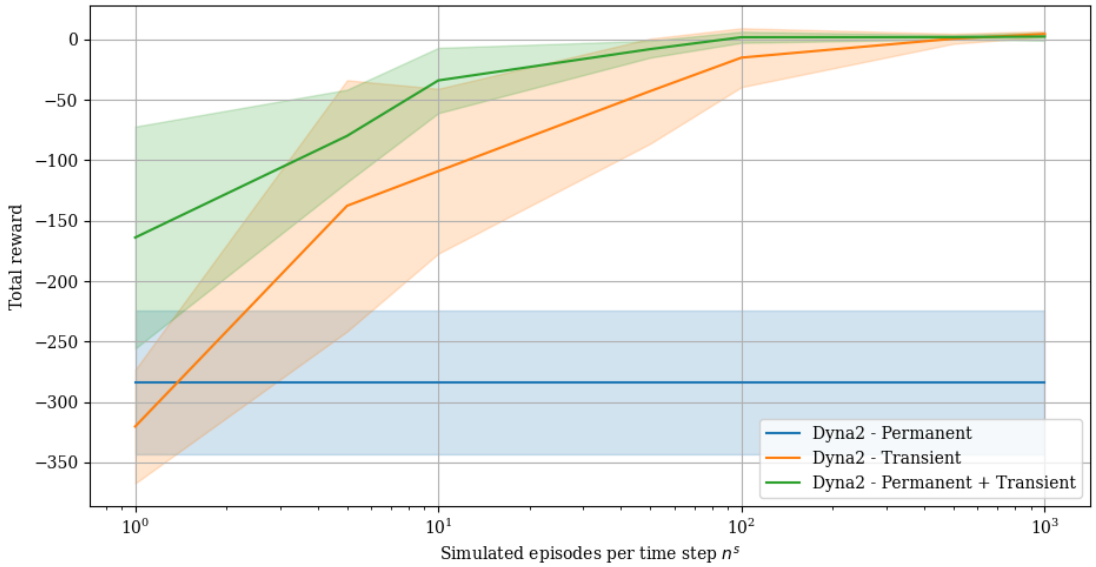

Figure 5-13: The total reward of three different RL algorithms using different types of action-value functions. Each algorithm uses a different method of filling its action-value function. Permanent action-value functions are learned by only real experience. Transient action-value functions are learned by only simulated experience. The combination of permanent- and transient action-value functions learn their foundation using real experience and incorporate local information using simulated experience. Ten test episodes were simulated for all three RL algorithms with different action-value functions. This experiment was repeated for each different amount of simulated episodes per timestep value n^s in Table 5-23.

step. Thirdly, the combined permanent- and transient action-value function seems to have a large advantage with respect to the only transient action-value function when decreasing the number of simulated episodes per time step n^s. It can be concluded that even only 1000 episodes of training episodes has had a positive contribution to the performance of the RL algorithm that combines a permanent- and transient action-value function. However, as can be concluded from Table 5-23, the highest number of simulated episodes per time step n^s that does not violate the average response time constraint lies close to 10. At 10 simulated episodes per time step, only the combined permanent- and transient action-value function is able to obtain relatively high returns.

5-3-3 Conclusions

Dyna is a powerful method of realizing high returns in unknown environments. It achieves this by learning a model of the environment to simulate experience. Without Dyna, users may experience undesired, irrational speed advice during a long exploration period of the agent. Dyna could be used to increase the performance of speed advice algorithms to a point that the human user benefits from the speed advice immediately. This stimulates user retention during the first training episodes, providing more time to improve the user's CRM. A disadvantage of choosing greedy policies is that the agent does not explore the environment. Therefore, it may require many episodes before the agent achieves high results in states that are not visited often such as high- or low speeds.

It was found that for the speed advice problem a simple model of the cyclist's response to the speed advice already provided satisfying results in early stages of the learning process. Reasonable returns were obtained with a fraction of the training episodes that were required when training a RL algorithm without Dyna.

These results did come at the cost of a significant increase of response time, violating the response time constraint. It can be concluded that decreasing the number of simulated episodes per time step decreases the response time. However, only RL algorithms using a combined permanent- and transient action-value function were able to deliver satisfying returns whilst taking the response time constraint into account.

5-4 Case study III: Practical implementation

It was found that the application of function approximation can reduce the amount of required training samples to learn an optimal policy. Moreover, Dyna can increase the performance of algorithms early on in the learning process, when there is little available information on the environment. Case study III aims to combine both techniques to find a practically implementable RL algorithm. This implies that real-world constraints, such as the response time constraint, cannot be violated anymore. First, a new step is introduced that can be used to efficiently extract some information from the cyclist, called the calibration run. Secondly, algorithms are trained using the information collected during this calibration run. This provides a baseline action-value function that is closer to the optimal action-value function than initiating the action-value functions randomly as was done in previous case studies. Thirdly, the performance of RL algorithms with different function approximators with Dyna are assessed. Finally, conclusions can be drawn on the practical implementation of these RL algorithms.

5-4-1 Calibration run

Exploration of the environment is a vital part of an agent's learning process. It is common practice that an agent starts with a high exploration rate ϵ to encourage the discovery of the environment. Since human users are involved in this learning process it is undesired to take random actions. Humans will not comply with this speed advice and will stop using the speed advice application.

To allow more exploration of the environment without impacting the user experience, users are asked to participate in a small setup experiment that investigates the cyclist speed preferences, called the calibration run. During this calibration run, the user is given speed advice that is designed to explore a wide range of the environment to find the cyclist response to different speed advises at different speeds. Since the users are aware of the experimental setting of the experiment, it is expected that the user experience will not be negatively affected. The calibration run is divided into different sections with characteristic action sequences, each aiming to explore a different part of the environment. In this case study, the calibration run was simulated. A practical field experiment is deferred to future research.

The sections of the calibration run and corresponding action sequences are defined in Table 5-24. During each section, the action sequences are repeated until a maximum duration is reached or when an endpoint speed threshold is crossed. These speed thresholds are based

on the current estimation of the cyclist's desired speed $E[v^{\text{ds}}]$. There is a low speed threshold v^{low} at $0.2 \cdot E[v^{\text{ds}}]$, a medium speed threshold v^{med} at $1.0 \cdot E[v^{\text{ds}}]$, and a high speed threshold v^{high} at $2.0 \cdot E[v^{\text{ds}}]$. When a section cannot be completed because of a traffic light encounter, the section is rescheduled for the next possible option.

Table 5-24: Overview of different calibration run sections and their corresponding action sequences. Action sequences are repeated until the maximum duration of the sequence is reached or the end point speed threshold is crossed.

	Action sequence	Max. duration (s)	Starting points	End points
Long hold	-	30	v^{med}	v^{med}
Hold	-	10	v^{med}	v^{med}
Acceleration steps	na, mcs	-	v^{med}, v^{low}	v^{high}, v^{med}
Aggressive acceleration	aa	-	v^{med}, v^{low}	v^{high}, v^{med}
Deceleration steps	nd, mcs	-	v^{med}, v^{high}	v^{low}, v^{med}
Aggressive deceleration	ad	-	v^{med}, v^{high}	v^{low}, v^{med}

In Figure 5-14, the order of calibration sections can be seen that are used to construct the calibration run. Figure 5-14 also presents the reference speed of the cyclist that is imposed by the speed advice generated by the agent's actions. Since it depends on the cyclist's preferences and the compliance of the cyclist, the duration of the calibration run section is unknown. This is represented by the dotted line. It must be noted that no traffic lights are encountered in this example. It can be concluded that the user visits a wide range of speeds whilst responding to different actions. During the long hold section, no speed advice is given. It is expected that the cyclist will cycle at its desired speed. Therefore, this speed data can be used to extract the cyclist's desired speed. The shorter hold section can be used to verify the cyclist desired speed and can serve as a small reset between other sections. The remaining sections are designed to extract information on how the cyclist responds to different speed advises at different speeds. This data can be stored in the CRM for later use and to find the cyclist's maximum acceleration $u^{\max}$.

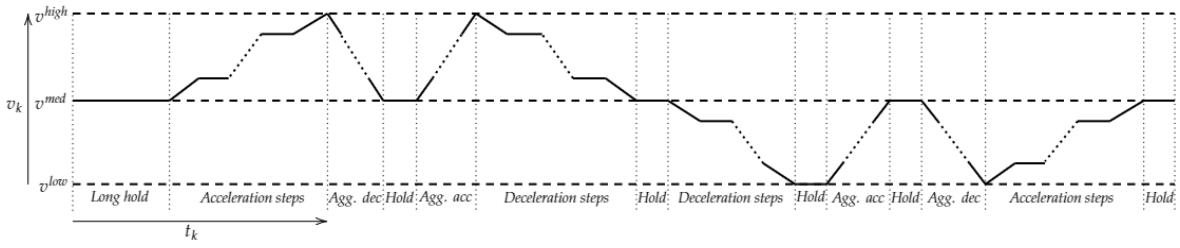

Figure 5-14: Reference speed imposed by the speed advice given during the calibration run. Each phase is characterised by a different action sequence. Phases terminate when a maximum duration is reached or a speed threshold is crossed.

For this case study, the three different cyclist types that are described in Section 5-1 participate in a calibration run, with the same action sequences as in Figure 5-14. Each cyclist executes the actions generated by the action sequences on a trajectory of two kilometres with a traffic light at one kilometre. As can be seen in Figure 5-15, the cyclists explore speeds and actions around their desired speed. It can also be seen that the action sequences are interrupted and resumed at the traffic light. The calibration run is repeated if the total action sequence is completed before the end of the two kilometres.

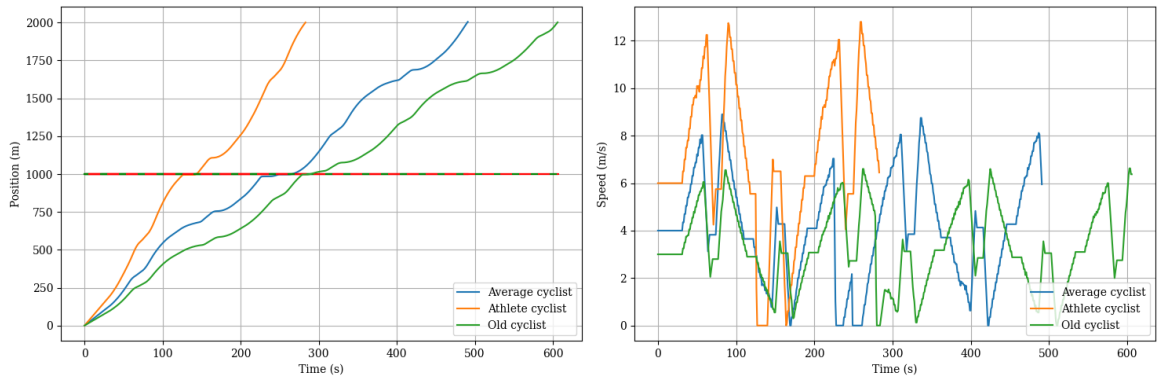

Figure 5-15: Cyclist trajectories of three different cyclist types that are exposed to the actions generated by the calibration run, expressed in position (left) and speed (right). The colour of the line at x^{tl}=1000 corresponds to the colour of the traffic light.

After this calibration run, the user is asked to provide some additional information. Namely, the user's weight, the weight of the user's bicycle, and the cyclist preferences as described in Table 5-5. The cyclist preferences can be adjusted by the user in a later stage, but then the algorithm must be retrained. Future research could look into adjusting the preference weights by learning from the user.

5-4-2 Baseline action-value functions

During the calibration run, information was extracted from the user. Similar to Dyna, this information can be used to simulate experience to train RL algorithms. First, the information on the desired speed v^{ds} and the cyclist preferences are required for the reward function. Secondly, the information on the cyclist's response to different speed advises at different speeds was used to fill the CRM. Thirdly, the maximum acceleration of the cyclist $u^{\max}$ can be used to construct a full compliance model as described in Table 4-1. This can be used to generate simulated experience when there is no real experience sample available in the CRM. Similar to simulating experience in case study II, the probability distribution of the traffic light phases is assumed to be known.

Since the calibration run is only executed once and only on a trajectory of two kilometres, it is expected that the information collected from this experiment is highly inaccurate. Besides the small sample size, the cyclist's measured speed during the calibration run can be compromised by traffic situations and by the bias that is imposed by the user's awareness of the experiment. Therefore, in this case study, the sensitivity to this uncertainty is evaluated for different RL algorithms. This is done by introducing errors in the cyclist's physical properties. In this experiment, the cyclist type "average cyclist" is used as an example to compare the robustness of the different RL algorithms. The estimated physical properties and their perturbed values can be found in Table 5-25.

For this part of the case study, four RL algorithms with different methods of describing the action-value function are selected from case study I, namely Q-table2, PF2, RBF2, and ANN3. During the selection of these RL algorithms, performance and average response time were taken into account. When RL algorithms scored equally high in terms of performance,

Table 5-25: Accurate physical properties of different cyclist types and an inaccurate approximation of the same physical properties.

	m^{c} (kg)	m^{b} (kg)	v^{ds} (m/s)	v^{l} (m/s)	v^{h} (m/s)	$v^{\max}$ (m/s)	$u^{\max}$ (m/s^2)	u^{eb} (m/s^2)
Average cyclist (accurate)	75	20	4.0	1.0	8.0	10.0	0.70	-3.0
Average cyclist (approximated)	70	22	4.5	1.1	9.0	9.2	0.78	-2.5
Athlete cyclist (accurate)	85	10	6.0	1.0	12.0	15.0	1.00	-4.0
Athlete cyclist (approximated)	82	8	5.3	1.3	10.6	13.0	0.90	-4.6
Old cyclist (accurate)	65	20	3.0	1.0	6.0	8.0	0.50	-3.0
Old cyclist (approximated)	69	24	3.4	1.7	6.8	7.3	0.45	-3.3

the algorithm with the lowest average response time was chosen to increase the number of simulated episodes that can be generated each time step. During the training process, simulated episodes are generated until the algorithm's performance converges. To simulate these episodes, the CRM that was constructed using the calibration run was used to generate cyclist responses and the approximated average cyclist physical properties were used to generate rewards. It must be noted that both the cyclist responses and the rewards that were generated are intentionally chosen not to be an accurate representation of the real environment.

5-4-3 Dyna with various RL algorithms

Offline training of RL algorithms using simulated experience provides a baseline action-value function that is closer to the optimal action-value function than initiating the action-value functions randomly as was done in previous case studies. Dyna can be used to locally correct inaccuracies in the baseline action-value function. The combined permanent- and transient action-value function strategy from case study II will be applied. In this experiment, the performance of four different pretrained RL extended with Dyna are assessed in a simulation setting resembling a real-world environment. The RL algorithms using Dyna with Q-tables, PFs, RBFs, and ANNs can be found in Appendix A.

The simulation parameters of the four RL algorithms can be found in Table 5-26. As mentioned before, the agent is not able to explore the environment due to the human in the loop. Therefore, the exploration rate ϵ is set constant to 0.01. Moreover, for each RL algorithm, the amount of simulated episodes per time step n^{s} are determined by finding the maximum amount of simulated episodes whilst not violating the response time constraint of one second. This metric is dependent of the computation speed of the device used to simulate these episodes. In this experiment an Intel(R) Core(TM) i7-8750H CPU @ 2.20 GHz and 16GB RAM memory was used. The learning rates in the real environment α and in the simulated environment α^s are properly selected to ensure fast learning without unstable behaviour. It can be seen that α^s is often higher than α to encourage local corrections to the action-value function.

Table 5-26: Simulation parameters for different RL algorithms using both real- and simulated experience.

	α	γ	ϵ	n^s	α^s	γ^s	ϵ_0^s	d^{ϵ^s}	ϵ_{min}^s
Q-table2 - dyna	0.2	0.98	0.01	10	0.2	0.98	0.5	0.7	0.01
PF2 - dyna	0.0005	0.98	0.01	30	0.005	0.98	0.5	0.85	0.01
RBF2 - dyna	0.005	0.98	0.01	10	0.01	0.98	0.5	0.7	0.01
ANN3 - dyna	0.001	0.98	0.01	1	0.005	0.98	0.5	0.5	0.01

The four algorithms with parameters as described in Table 5-26 are tested on a simulation with 300 episodes with a test episode after every 5 episodes. The results can be found in Figure 5-16 and Table 5-27. From Figure 5-16 it can be concluded that the ANN3-Dyna and Q-table2-Dyna are able to achieve high returns on the first episode. This means that these RL algorithms are robust to uncertainties in the cyclist information. For Q-table2-Dyna, this robustness to uncertainty was previously observed in case study II where high returns could be achieved using only a transient action-value function. The RL algorithms using linear function approximators are less robust to these uncertainties. Even though PFs outperformed RBFs in case study I, PF2-Dyna reveals its strength in this experiment. Due to the low number of variables in weight vector $\mathbf{w}$, many simulated episodes can be generated at each time step. Moreover, only a few simulated episodes are required for local corrections to the action-value function due to the low number of variables in the weight vector $\mathbf{w}$.

It can also be seen that the ANN3-Dyna is the only algorithm with a clear positive learning curve over the 300 episodes. This can be explained by the results from case study I, which concluded that the remaining RL algorithms require many more training episodes to converge to an optimal action-value function. However, when training the ANN3-Dyna for a longer period while only taking greedy actions, performance of the RL algorithm was found to deteriorate. When only taking the same, greedy actions, leading to trajectories yielding high returns a lack of diversity in the replay buffer can occur. By fitting the ANN on a static data set, overfitting can occur.

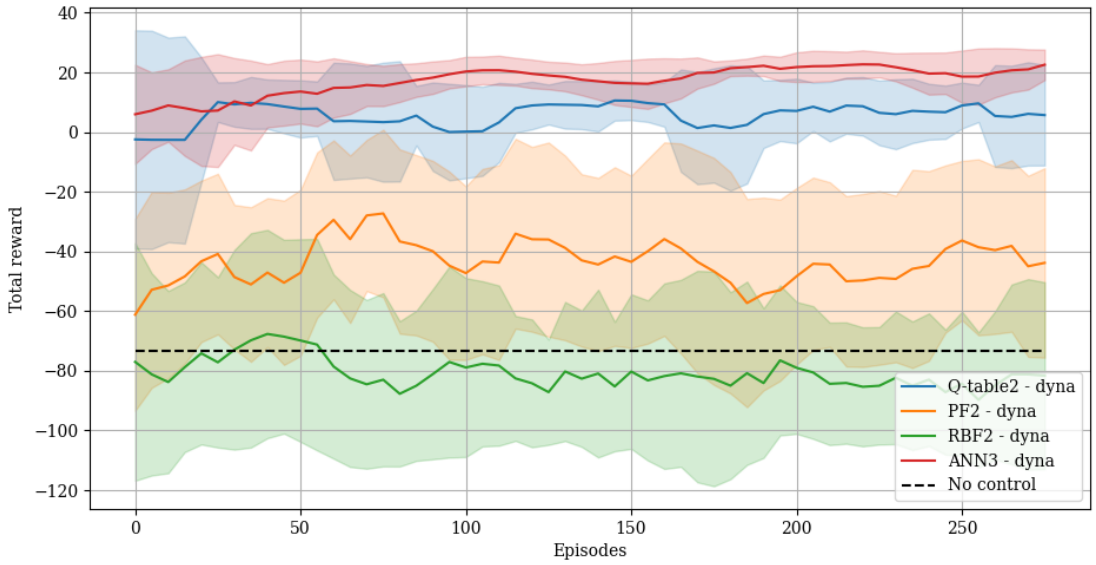

Figure 5-16: The learning curves of four different RL algorithms that have been trained by information collected from a calibration run. The RL algorithms differ in function approximation method used.

The performance of the different algorithms, as represented by Figure 5-16, is also reflected in their behaviour, seen in Figure 5-17. Whereas in case study I, RBF2 achieved high returns, in case study 3 RBF2-Dyna is unable to follow a strategy. This can be explained by the fact the RBF2-Dyna was pretrained with inaccurate estimates of the real cyclist's physical parameters. It can be concluded that RBF2-Dyna lacks the robustness to handle these uncertainties and

does not possess the adaptive properties to quickly adapt its action-value function to find an optimal policy for the real parameters. Despite also being a linear function approximation method, PF2-Dyna can achieve higher returns than RBF2-Dyna. As discussed previously, this can be explained by its adaptive properties. It can be seen that in the first few time steps, PF2-Dyna is unable to output desired behaviour. By adapting to the real environment after some time steps, PF2-Dyna finds the cyclist's desired speed, and can guide the cyclist through a green light when coming closer to the intersection. PF2-Dyna is capable of maximizing its return at that specific moment, but it is not able to look multiple steps in the future to identify actions that are less beneficial at this moment but maximize returns in a later stage. The focus of action-value functions using linear function approximators on maximizing the current rewards instead of the overall rewards was previously observed in case study I. This forward-looking strategy is visible for Q-table2-Dyna and ANN3-Dyna. An example of this is that both algorithms divert slightly from their desired speed early in the trajectory so the cyclist can smoothly cross a green light.

Table 5-27: Performance of a cyclist with speed advice generated by different RL algorithms.

	Avg. reward	Green ratio (%)	Avg. TTS (s)	Avg. speed (m/s)	Avg. energy (J)
Q-table2 - Dyna	7.32	96.1	48.21	4.3	2,986
PF2 - Dyna	-38.93	68.8	54.33	3.8	3,126
RBF2 - Dyna	-63.27	57.7	52.67	4.1	4,724
ANN3 - Dyna	18.21	97.2	50.17	4.1	2,732

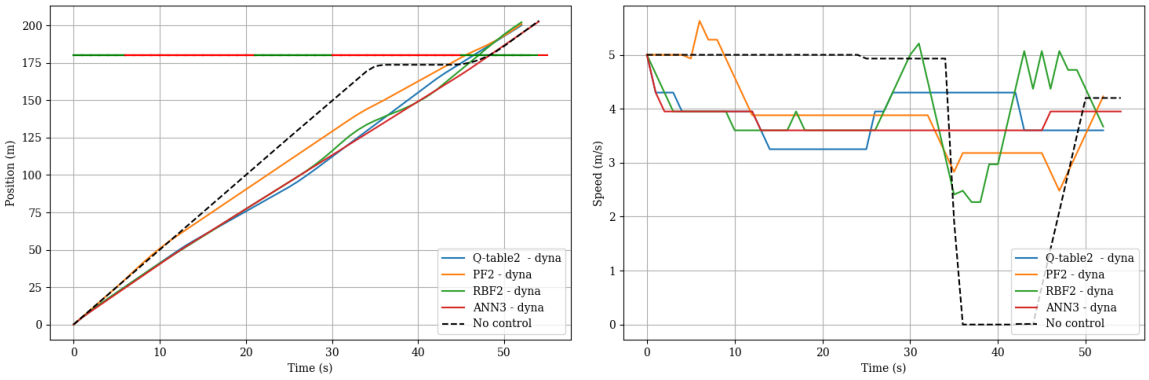

Figure 5-17: Trajectories of an average cyclist receiving speed advice generated by different RL algorithms, expressed in position (left) and speed (right). The colour of the line at x^{tl}=180 corresponds to the colour of the traffic light.

To verify the effectiveness of ANN3-Dyna on different cyclist types, case study III was repeated for the two remaining cyclist types, namely athlete cyclists and old cyclists. As can be seen in Figure 5-18, ANN3-Dyna is capable of learning to give personalised speed advice to other cyclists with different preferences as well. Table 5-28 illustrates that the cyclist's physical parameters, such as the desired speed v^{ds}, from Table 5-25 are taken into account in the RL algorithm.

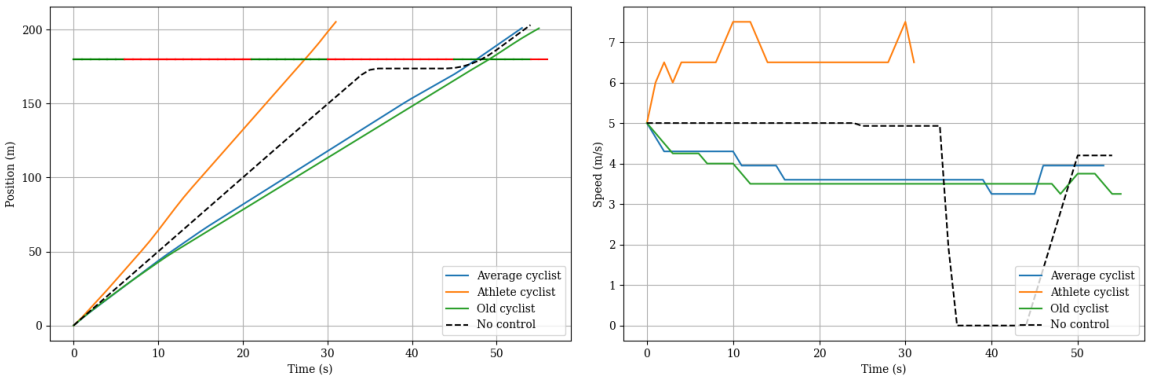

Figure 5-18: Trajectories of three different cyclist types receiving speed advice generated by a RL algorithm using Dyna and ANNs as a functions approximation method, expressed in position (left) and speed (right). The colour of the line at x^{tl}=180 corresponds to the colour of the traffic light. This simulation was generated using the average cyclist type settings and the test episode setting.

Table 5-28: Performance of different cyclist types with speed advice generated by the ANN3 RL algorithm. The characteristic of the cyclists can be found in Table 5-4 and Table 5-5.

	Green ratio (%)	Avg. TTS (s)	Avg. speed (m/s)	Avg. energy (J)
Average cyclist	97.2	50.17	4.1	2,732
Athlete cyclist	98.2	36.85	5.6	5,482
Old cyclist	96.2	69.11	3.0	2,054

5-4-4 Conclusions

Case study III aimed to mimic an initial user experience. This exposed several issues such as a lack of knowledge of the cyclist's characteristics, the response time constraint, and the inability to explore the environment. To overcome these issues, three methods were applied. First, a calibration run was executed to collect a rough estimate of the cyclist's physical characteristics. This information could then be used to properly initialize the agent's action-value function using simulated experience. Secondly, function approximation was used to decrease the amount of required training episodes to achieve high returns. Thirdly, Dyna was applied to the RL algorithms to locally correct the permanent action-value functions with transient action-value functions to increase performance early in the learning phase.

Comparing Table 5-19 with Table 5-28, it was found that proper initialization of the action-value function strongly increased the performance of the RL algorithms. The action-value functions of linear functions approximation methods lacked robustness to the uncertainties present when pretraining the RL algorithms. However, the simplicity of PFs allowed for effective local corrections to the action-value functions, thereby increasing performance compared to RBF function approximators. Both Q-table2-Dyna and ANN3-Dyna achieved high returns from the first episode.

Only ANN3-Dyna showed a positive learning curve over the 300 training episodes. The remaining RL algorithms required more training episodes. The highly adaptive characteristics of ANNs can cause overfitting when only greedy actions are taken. Therefore, it is suggested to simulate experience with a high exploration rate offline when more accurate knowledge is

acquired from real experiences. This process could be repeated after each fixed amount of real episodes to prevent overfitting.

Case study III was repeated for different cyclist types to verify the results under different circumstances. Overall, it can be concluded that a properly initialized RL algorithm using ANNs and Dyna is capable of giving optimal personalised speed advice to cyclists.

5-5 Conclusions

Previous research has established that it is possible to apply RL algorithms to provide cyclists approaching signalized intersections with personalised speed advice. However, these methods are not considered practically implementable due to an unrealistic amount of required training episodes with a high exploration rate. This case study used function approximation and Dyna to overcome these problems and make RL algorithms for giving personalised speed advice to cyclist's approaching intersections practically implementable. Experiments were carried out where a traffic environment was simulated where an agent could collect rewards by choosing proper speed advice actions that guide cyclists through an actuated traffic light.

Case study I studied the possibility to reduce the number of training samples required to train an RL algorithm an optimal policy whilst remaining competitive in terms of performance with respect to conventional RL algorithms. It was found that linear function approximation methods such as PFs and RBFs require less training episodes than benchmark RL algorithms. However, their poor descriptive properties imposed by an inability to combine state-information caused their performance to deteriorate compared to the benchmark RL algorithm. Using ANNs to approximate the action-value function instead of using a Q-table reduced the amount of required episodes from 50,000 to 300 whilst increasing performance.

Case study II addresses the problem of poor performance in early learning phases due to exploration. It was concluded that corrections to the action-value function made with a simple model can already increase the agent's performance to a near-optimal level on the first real episode. However, this is under the assumption that enough episodes can be simulated after each time step. The amount of required simulated episodes per time step n^{s} can be reduced by exploiting a combination of a permanent- and a transient action-value function.

The aim of case study III was to combine function approximation and Dyna to overcome both problems with RL algorithms currently restraining them from practical implementation. To assess if these additions to the conventional RL algorithms allow for practical implementation, an initial user experience was simulated. This means there was no prior knowledge of the cyclist's characteristics, the response time constraint could not be violated and exploration of the environment is discouraged. By executing a small experiment with the user some prior information on the cyclist could be extracted. This information was used to train a baseline permanent action-value function. The performance of four RL algorithms using Dyna and different methods of describing their action-value functions were assessed. These experiments confirmed that RL algorithms using Dyna and ANNs to describe their action-value function are well capable of achieving high returns on the first episode, require a small amount of episodes to adapt to their environment, and are robust to uncertainties in the cyclist characteristics.

Chapter 6

Conclusions and recommendations

Being a safe and healthy alternative for polluting and space-inefficient motorised vehicles, cycling can strongly improve living conditions in urban areas. By combining innovative traffic communication and reinforcement learning (RL), personalised speed advice can be given to cyclists approaching a signalized intersection. This reduces travel time and increases safety whilst taking cyclist preferences into account. A major disadvantage of RL is the need for a great deal of training experience where, especially in the early stages of the learning process, poor performance is inevitable. Since humans are unable to take part in this long and uncomfortable learning process, the current RL approaches to this problem are not considered practically implementable. In this thesis, a novel RL approach is developed to establish a practical implementation of RL algorithms for giving personalised speed advice to cyclists approaching intersections. This is accomplished by using function approximators to reduce the required amount of training experience and Dyna to achieve high performance, even on the first user experience. In this chapter, the conclusions of this thesis are drawn and recommendations of future research are given.

6-1 Conclusions

The research in this thesis focuses on overcoming problems imposed by introducing human-machine interaction in RL problems. This topic was researched for the use case of giving personalised speed advice to cyclists approaching signalized intersections. The goal of this thesis is defined as follows.

> *"Practical implementation of reinforcement learning algorithms for giving personalised speed advice to cyclists approaching intersections using function approximation and Dyna."*

The RL algorithm is considered to be practically implementable when the four predefined requirements are met. These requirements will be discussed next.

1. *The speed advice technology is accessible and user friendly.*
   Developments in cooperative traffic communication technology have enabled new possibilities in the field of intelligent speed advisory systems (ISAS). Cyclists can use their smartphones to exchange messages through a centralized server that is connected to traffic infrastructure such as traffic lights. Communicating through a smartphone allows frequent, personalised speed advice over an entire trajectory without invasive technological changes to the bicycle. The speed advice can be communicated through a range of methods such as a screen, vibrations, or sounds. Moreover, the centralized server allows the algorithm to collect and store data to be post-processed offline. Considering these factors, the speed advice technology is considered accessible and user friendly.

2. *The speed advice significantly improves the user's cycling experience compared to cycling without speed advice.*
   When comparing simulations with and without speed advice, it can be concluded that introducing speed advice improves the performance of the cyclist. Without speed advice, an average cyclist can cross a green light 34.1% of the time when arriving at the signalized intersection. With the same amount of energy, the same cyclist that is receiving speed advice using the novel RL approach can to cross a green light 97.2% of the time. This has a major positive impact on the total energy consumption, travel time, and safety of the cyclist.

3. *The agent must learn to give high-quality speed advice within a reasonable number of traffic light encounters.*
   In the case study, a conventional RL using a Q-table with coarse discretization steps to describe its action-value function required 50,000 episodes of simulation to converge to an optimal policy. This is considered to be an unrealistic amount of training episodes for a human cyclist to execute before the agent is able to generate sensible speed advice. Moreover, the large Q-tables used by these conventional RL algorithms demand a high memory capacity.

   By approximating action-value functions, the number of variables that must be learned and stored can be reduced to only the weights of a weight vector. In case study I, three types of function approximation methods were assessed. It was found that linear function approximation methods such as polynomial functions (polynomial function (PF)s) and radial basis functions (radial basis function (RBF)s) can reduce the required number of training episodes and memory capacity. However, since these methods evaluate each state value separately, significant correlations between states were not described accurately. This resulted in performance deterioration compared to conventional RL algorithms. On the other hand, nonlinear function approximation methods using artificial neural networks (ANNs) were not only able to reduce the amount of required training episodes by 99.5% but also significantly increase the performance of the speed advice with respect to conventional RL algorithms. Since RL algorithms using ANNs as a function approximator only required 300 training episodes, this requirement is considered to be met.

4. *The speed advice may never negatively impact cycling experience.*
   The benefits of giving speed advice to cyclists approaching signalized intersections can have an opposite effect when not executed properly. This means that cyclists will spend

more travel time, use an unnecessary amount of energy, and can encounter dangerous traffic situations. Once this occurs, cyclists will stop trusting and using speed advice. Therefore, the speed advice must be of high quality on the initial user experience and cannot negatively impact the cycling experience. This poses a problem as agents learn by taking random actions to explore an unknown environment.

In case study II, it was found that by extending conventional RL algorithms with Dyna, high returns could be achieved in an unknown environment. This is achieved by simulating experience using a model of the environment. It was concluded that, even with an inaccurate model of the environment, the RL algorithm was able to generate the desired results. However, this is under the assumption of an abundance of simulation time and that the method of describing the action-value function is robust to the uncertainties of the environment model. Nevertheless, case study II proved that it is possible for RL algorithms to meet this requirement.

It can be concluded that all the separate requirements for the practical implementation of RL algorithms for the speed advice problem can be met. In case study III, an initial user experience was simulated to test if these requirements can be met simultaneously. This included the major constraints that are accompanied by practical implementation in real-world traffic problems such as a lack of knowledge of the cyclist's characteristics, a time constraint of one second to respond to traffic, and the inability to explore the environment. Both function approximation and Dyna were combined to meet the predefined requirements in this setting. Moreover, a method called the calibration run was introduced to extract some prior knowledge of the cyclist. This prior information of the cyclist was used to properly initialize the action-value functions of the RL algorithms.

After testing different RL algorithms, it was found that one algorithm was well capable of satisfying each requirement simultaneously. The RL algorithm using Dyna and ANNs as an action-value function approximator was able to catch a green light 97.2% of the time while using the same amount of energy as a cyclist without speed advice that was able to catch a green light 34.1% of the time. From the behaviour analysis of the algorithm, it was found that it was able to formulate a profound speed advice strategy that prevented unnecessary accelerations, avoided the risk of catching red lights, and adhere to the predefined cyclist preferences.

The required technology used in this case study already exists. More smart traffic infrastructure is being rolled out over the Netherlands and a smartphone application has never been so accessible to the public. Therefore, it can be concluded that the practical implementation of RL algorithms for giving personalised speed advice to cyclists approaching signalized intersections can be achieved using function approximation and Dyna.

6-2 Future research

During the research, some interesting fields of research were found that could have high potential of making a contribution to the ISAS for cyclists. Since these topics were outside the scope of this thesis, they are deferred to future research.

1. **Testing the algorithm**: The case study in this thesis has presented some promising results. However, since some assumptions were made during the simulation case study, it can be interesting to investigate the performance of the algorithm in a real-world test scenario.

2. **Variable cyclist response model discretization step size**: In the case study in this thesis a relatively simple cyclist response model (CRM) proved to be effective. This could mainly be because the CRM was used to describe a simplified model of cyclist compliance instead of real cyclist behaviour. A CRM that decreases bin size once more samples are collected is suggested such that the CRM can always generate real-world samples and increases accuracy over time.

3. **Incorporate traffic features**: Adding extra relevant information to the system can benefit the performance of the RL algorithm. Actuated traffic light phases are influenced by incoming traffic. By incorporating shared traffic data, the RL algorithm could use traffic streams to predict the future traffic light phase more accurately.

4. **Complex intersections**: Research can be done on extending speed advice for intersections with multiple lanes and traffic lights. It could also optimize speed advice for not one, but multiple consecutive signalized intersections.

5. **Adaptive reward functions**: RL algorithms can adapt to the user. Every user has its own cycling preferences. However, the reward function is set manually in advance. Research can be done on learning the preferences of the cyclist.

6. **Queues**: Predicting and avoiding red lights may not be effective when there is a queue present that forces the cyclist to brake when arriving at the intersection. This topic could be combined with predicting future traffic light states.

7. **Multiple cyclists**: The current research only takes one cyclist into account. However, cyclists influence each other's driving behaviour. Influences of other cyclists could be overtaking other cyclists, idling behind slower cyclist groups or having to brake for a queue at the intersection. This type of reinforcement learning is called multi-agent reinforcement learning and is still a challenging field of research.

Appendix A

Algorithms

Algorithm 1 RL algorithm (Q-table)

procedure LEARN
  Initialize $P^{\mathrm{t}}(B^{\mathrm{i}}, t^{\mathrm{p}}), P^{\mathrm{s}}(B^{\mathrm{i}}, t^{\mathrm{p}}), R$
  Initialize $Q(s, a)$, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$ arbitrarily
  loop
    $s \leftarrow s_0$ ▷ Start new episode
    $a \leftarrow \epsilon$-greedy$(s; Q; \epsilon)$
    while $s \neq s^{\mathrm{t}}$ and $s \neq s^{\mathrm{eb}}$ **do**
      Take action a
      Observe s' and r
      $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_a Q\left(s', a\right) - Q(s, a) \right]$
      $a \leftarrow \epsilon$-greedy$(s'; Q; \epsilon)$,
      $s \leftarrow s'$, $a \leftarrow a'$,
    end while
  end loop
end procedure

Algorithm 2 RL algorithm (linear function approximators)

procedure LEARN
  Initialize $P^{\mathrm{t}}(B^{\mathrm{i}}, t^{\mathrm{p}}), P^{\mathrm{s}}(B^{\mathrm{i}}, t^{\mathrm{p}}), R$
  Initialize weight vector $\mathbf{w}$ arbitrarily
  loop
    $s \leftarrow s_0$ ▷ Start new episode
    $a \leftarrow \epsilon$-greedy$(s; \hat{Q}; \epsilon)$
    while $s \neq s^{\mathrm{t}}$ and $s \neq s^{\mathrm{eb}}$ **do**
      Take action a
      Observe s' and r
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[r + \gamma \hat{Q}\left(s', a', \mathbf{w}\right) - \hat{Q}(s, a, \mathbf{w}) \right] \nabla \hat{Q}(s, a, \mathbf{w})$
      $a \leftarrow \epsilon$-greedy$(s'; \hat{Q}; \epsilon)$,
      $s \leftarrow s'$, $a \leftarrow a'$,
    end while
    if $\alpha \leq \alpha^{\min}$ **then**
      $\alpha = d^{\alpha}\alpha$
    end if
    if $\epsilon \leq \epsilon^{\min}$ **then**
      $\epsilon = d^{\epsilon}\epsilon$
    end if
  end loop
end procedure

Algorithm 3 RL algorithm (artificial neural networks)

procedure LEARN
  Initialize $P^{\mathrm{t}}(B^{\mathrm{i}}, t^{\mathrm{p}}), P^{\mathrm{s}}(B^{\mathrm{i}}, t^{\mathrm{p}}), R$
  Initialize target weight vector $\mathbf{w}^{\mathrm{t}}$ and local weight vector $\mathbf{w}^{\mathrm{l}}$ arbitrarily
  loop
    $s \leftarrow s_0$ ▷ Start new episode
    $a \leftarrow \epsilon$-greedy$(s'; \hat{Q}^{\mathrm{l}}; \epsilon)$
    while $s \neq s^{\mathrm{t}}$ and $s \neq s^{\mathrm{eb}}$ **do**
      Take action a
      Observe s' and r
      Store (s, a, r, s')-pair in replay memory
      Sample i (s, a, r, s')-pairs from replay memory
      $\mathbf{w}^{\mathrm{l}} = \min_{\mathbf{w}^{\mathrm{l}}} \sum_0^{\mathrm{i}} \left(r_{\mathrm{i}} + \gamma \max_{a'_{\mathrm{i}}} \hat{Q}_{\mathrm{i}} \left(s'_{\mathrm{i}}, a'_{\mathrm{i}}, \mathbf{w}^{\mathrm{l}} \right) - \hat{Q}_{\mathrm{i}} \left(s_{\mathrm{i}}, a_{\mathrm{i}}, \mathbf{w}^{\mathrm{l}} \right) \right)^2$
      $\mathbf{w}^{\mathrm{t}} = \mathbf{w}^{\mathrm{l}}\tau + \mathbf{w}^{\mathrm{t}}(1 - \tau)$
      $a \leftarrow \epsilon$-greedy$(s'; \hat{Q}^{\mathrm{l}}; \epsilon)$,
      $s \leftarrow s'$, $a \leftarrow a'$,
    end while
  end loop
end procedure

Algorithm 4 RL algorithm using Dyna (Q-table)

procedure LEARN
  Initialize $P^{\mathrm{t}}(B^{\mathrm{i}}, t^{\mathrm{p}}), P^{\mathrm{s}}(B^{\mathrm{i}}, t^{\mathrm{p}}), R$
  Initialize $Q(s, a)$, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$ arbitrarily
  loop
    $\overline{Q} \leftarrow Q$
    $s \leftarrow s_0$ ▷ Start new real episode
    $a \leftarrow \epsilon$-greedy$(s; \overline{Q}; \epsilon)$
    while $s \neq s^{\mathrm{t}}$ and $s \neq s^{\mathrm{eb}}$ **do**
      Take action a
      Observe s' and r
      CRM$(s, a) \leftarrow s'$
      $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_a Q\left(s', a\right) - Q(s, a) \right]$
      SEARCH(s',Q,$P^{\mathrm{t}}(B^{\mathrm{i}}, t^{\mathrm{p}})$, $P^{\mathrm{s}}(B^{\mathrm{i}}, t^{\mathrm{p}})$, R)
      $a \leftarrow \epsilon$-greedy$(s'; \overline{Q}; \epsilon)$,
      $s \leftarrow s'$, $a \leftarrow a'$,
    end while
  end loop
end procedure
procedure SEARCH(s',Q,$P^{\mathrm{t}}(B^{\mathrm{i}}, t^{\mathrm{p}})$, $P^{\mathrm{s}}(B^{\mathrm{i}}, t^{\mathrm{p}})$, R)
  for n^s **do**
    $\overline{Q} \leftarrow Q$
    $\bar{s} \leftarrow s$ ▷ Start new simulated episode
    $\bar{a} \leftarrow \epsilon$-greedy$(\bar{s}; \overline{Q}; \epsilon^s)$
    while $\bar{s} \neq \bar{s}^t$ and $\bar{s} \neq \bar{s}^{\mathrm{eb}}$ **do**
      $\bar{s}' \leftarrow (\mathrm{CRM}(\bar{s}, \bar{a}),\ P^{\mathrm{t}}(\overline{B}^{\mathrm{i}}, \bar{t}^{\mathrm{p}}),\ P^{\mathrm{s}}(\overline{B}^{\mathrm{i}}, \bar{t}^{\mathrm{p}}))$
      Observe $\bar{r}$
      $\overline{Q}(\bar{s}, \bar{a}) \leftarrow \overline{Q}(\bar{s}, \bar{a}) + \alpha^s \left[\bar{r} + \gamma^s \max_{\bar{a}} \overline{Q}\left(\bar{s}', \bar{a}\right) - \overline{Q}(\bar{s}, \bar{a}) \right]$
      $\bar{a}' \leftarrow \epsilon$-greedy$(\bar{s}'; \overline{Q}; \epsilon^s)$,
      $\bar{s} \leftarrow \bar{s}'$, $\bar{a} \leftarrow \bar{a}'$,
    end while
  end for
end procedure

Algorithm 5 RL algorithm using Dyna (linear function approximators)

procedure LEARN
  Initialize $P^{\mathrm{t}}(B^{\mathrm{i}}, t^{\mathrm{p}}), P^{\mathrm{s}}(B^{\mathrm{i}}, t^{\mathrm{p}}), R$
  Initialize real weight vector $\mathbf{w}$ and simulated weight vector $\overline{\mathbf{w}}$ arbitrarily
  loop
    $\overline{\mathbf{w}} \leftarrow \mathbf{w}$
    $s \leftarrow s_0$ ▷ Start new episode
    $a \leftarrow \epsilon$-greedy$(s; \hat{\overline{Q}}; \epsilon)$
    while $s \neq s^{\mathrm{t}}$ and $s \neq s^{\mathrm{eb}}$ **do**
      Take action a
      Observe s' and r
      CRM$(s, a) \leftarrow s'$
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[r + \gamma \hat{Q}\left(s', a', \mathbf{w}\right) - \hat{Q}(s, a, \mathbf{w}) \right] \nabla \hat{Q}(s, a, \mathbf{w})$
      SEARCH(s',$\hat{Q}$,$P^{\mathrm{t}}(B^{\mathrm{i}}, t^{\mathrm{p}})$, $P^{\mathrm{s}}(B^{\mathrm{i}}, t^{\mathrm{p}})$, R)
      $a \leftarrow \epsilon$-greedy$(s'; \hat{\overline{Q}}; \epsilon)$,
      $s \leftarrow s'$, $a \leftarrow a'$,
    end while
    if $\alpha \leq \alpha^{\min}$ **then**
      $\alpha = d^{\alpha}\alpha$
    end if
    if $\epsilon \leq \epsilon^{\min}$ **then**
      $\epsilon = d^{\epsilon}\epsilon$
    end if
  end loop
end procedure
procedure SEARCH(s',$\hat{Q}$,$P^{\mathrm{t}}(B^{\mathrm{i}}, t^{\mathrm{p}})$, $P^{\mathrm{s}}(B^{\mathrm{i}}, t^{\mathrm{p}})$, R)
  for $\underline{n}^s$ **do**
    $\hat{\overline{Q}} \leftarrow \hat{Q}$
    $\bar{s} \leftarrow s$ ▷ Start new simulated episode
    $\bar{a} \leftarrow \epsilon$-greedy$(\bar{s}; \hat{\overline{Q}}; \epsilon^s)$
    while $\bar{s} \neq \bar{s}^t$ and $\bar{s} \neq \bar{s}^{\mathrm{eb}}$ **do**
      $\bar{s}' \leftarrow (\mathrm{CRM}(\bar{s}, \bar{a}),\ P^{\mathrm{t}}(\overline{B}^{\mathrm{i}}, \bar{t}^{\mathrm{p}}),\ P^{\mathrm{s}}(\overline{B}^{\mathrm{i}}, \bar{t}^{\mathrm{p}}))$
      Observe $\bar{r}$
      $\overline{\mathbf{w}} \leftarrow \overline{\mathbf{w}} + \alpha \left[r + \gamma \hat{\overline{Q}}\left(s', a', \overline{\mathbf{w}}\right) - \hat{\overline{Q}}(s, a, \overline{\mathbf{w}}) \right] \nabla \hat{\overline{Q}}(s, a, \overline{\mathbf{w}})$
      $\bar{a}' \leftarrow \epsilon$-greedy$(\bar{s}'; \hat{\overline{Q}}; \epsilon^s)$,
      $\bar{s} \leftarrow \bar{s}'$, $\bar{a} \leftarrow \bar{a}'$,
    end while
    if $\alpha^s \leq \alpha^{\min^s}$ **then**
      $\alpha^s = d^{\alpha^s}\alpha^s$
    end if
    if $\epsilon^s \leq \epsilon^{\min^s}$ **then**
      $\epsilon^s = d^{\epsilon^s}\epsilon^s$
    end if
  end for
end procedure

Algorithm 6 Dyna RL algorithm (artificial neural networks)

procedure LEARN
  Initialize $P^{\mathrm{t}}(B^{\mathrm{i}}, t^{\mathrm{p}}), P^{\mathrm{s}}(B^{\mathrm{i}}, t^{\mathrm{p}}), R$
  Initialize target weight vector $\mathbf{w}^{\mathrm{t}}$ and local weight vector $\mathbf{w}^{\mathrm{l}}$ arbitrarily
  loop
    $\overline{\mathbf{w}}^{\mathrm{t}} \leftarrow \mathbf{w}^{\mathrm{t}}$
    $\overline{\mathbf{w}}^{\mathrm{l}} \leftarrow \mathbf{w}^{\mathrm{l}}$
    $s \leftarrow s_0$ ▷ Start new episode
    $a \leftarrow \epsilon$-greedy$(s; \overline{\hat{Q}}^{\mathrm{l}}; \epsilon)$
    while $s \neq s^{\mathrm{t}}$ and $s \neq s^{\mathrm{eb}}$ **do**
      Take action a
      Observe s' and r
      CRM$(s, a) \leftarrow s'$
      Store (s, a, r, s')-pair in replay memory
      Sample i (s, a, r, s')-pairs from replay memory
      $\mathbf{w}^{\mathrm{l}} = \min_{\mathbf{w}^{\mathrm{l}}} \sum_0^{\mathrm{i}} \left(r_{\mathrm{i}} + \gamma \max_{a'_{\mathrm{i}}} \hat{Q}_{\mathrm{i}} \left(s'_{\mathrm{i}}, a'_{\mathrm{i}}, \mathbf{w}^{\mathrm{l}} \right) - \hat{Q}_{\mathrm{i}} \left(s_{\mathrm{i}}, a_{\mathrm{i}}, \mathbf{w}^{\mathrm{l}} \right) \right)^2$
      $\mathbf{w}^{\mathrm{t}} = \mathbf{w}^{\mathrm{l}} \tau + \mathbf{w}^{\mathrm{t}} (1 - \tau)$
      SEARCH$(s', \hat{Q}^{\mathrm{l}}, \hat{Q}^{\mathrm{t}}, P^{\mathrm{t}}(B^{\mathrm{i}}, t^{\mathrm{p}}), P^{\mathrm{s}}(B^{\mathrm{i}}, t^{\mathrm{p}}), R)$
      $a \leftarrow \epsilon$-greedy$(s'; \overline{\hat{Q}}^{\mathrm{l}}; \epsilon)$,
      $s \leftarrow s'$, $a \leftarrow a'$,
    end while
  end loop
end procedure
procedure SEARCH$(s', \hat{Q}^{\mathrm{l}}, \hat{Q}^{\mathrm{t}}, P^{\mathrm{t}}(B^{\mathrm{i}}, t^{\mathrm{p}}), P^{\mathrm{s}}(B^{\mathrm{i}}, t^{\mathrm{p}}), R)$
  for n^s **do**
    $\overline{\hat{Q}}^{\mathrm{t}} \leftarrow \hat{Q}^{\mathrm{t}}$
    $\overline{\hat{Q}}^{\mathrm{l}} \leftarrow \hat{Q}^{\mathrm{l}}$
    $\bar{s} \leftarrow s$ ▷ Start new simulated episode
    $\bar{a} \leftarrow \epsilon$-greedy$(\bar{s}; \overline{\hat{Q}}^{\mathrm{l}}; \epsilon^s)$
    while $\bar{s} \neq \bar{s}^t$ and $\bar{s} \neq \bar{s}^{\mathrm{eb}}$ **do**
      $\bar{s}' \leftarrow (\mathrm{CRM}(\bar{s}, \bar{a}), P^{\mathrm{t}}(\overline{B}^{\mathrm{i}}, \bar{t}^{\mathrm{p}}), P^{\mathrm{s}}(\overline{B}^{\mathrm{i}}, \bar{t}^{\mathrm{p}}))$
      Observe $\bar{r}$
      $\overline{\mathbf{w}}^{\mathrm{l}} = \min_{\overline{\mathbf{w}}^{\mathrm{l}}} \sum_0^{\mathrm{i}} \left(r_{\mathrm{i}} + \gamma \max_{a'_{\mathrm{i}}} \overline{\hat{Q}}_{\mathrm{i}}^{\mathrm{t}} \left(s'_{\mathrm{i}}, a'_{\mathrm{i}}, \overline{\mathbf{w}}^{\mathrm{t}} \right) - \overline{\hat{Q}}_{\mathrm{i}}^{\mathrm{l}} \left(s_{\mathrm{i}}, a_{\mathrm{i}}, \overline{\mathbf{w}}^{\mathrm{l}} \right) \right)^2$
      $\overline{\mathbf{w}}^{\mathrm{t}} = \overline{\mathbf{w}}^{\mathrm{l}} \tau + \overline{\mathbf{w}}^{\mathrm{t}} (1 - \tau)$
      $\bar{a}' \leftarrow \epsilon$-greedy$(\bar{s}'; \overline{\hat{Q}}^{\mathrm{l}}; \epsilon^s)$,
      $\bar{s} \leftarrow \bar{s}'$, $\bar{a} \leftarrow \bar{a}'$,
    end while
  end for
end procedure

Appendix B

Tables

Test scenario

During the test episode, the environment conditions must be equal for each test episode in order to compare the performance of different algorithms at different stages in the learning process. This also applies to he traffic light transition probability $P^{\mathrm{t}}(B^{\mathrm{i}}, t^{\mathrm{p}})$ and the switching traffic light phase probability $P^{\mathrm{s}}(B^{\mathrm{i}}, t^{\mathrm{p}})$. As can be seen in Table B-1 and Table B-2, the termination and switching process of the traffic lights is now deterministic instead of stochastic as in Table 5-2 and Table 5-3.

Table B-1: Traffic light termination probabilities $P^{s}(B^{\mathrm{i}}|B^{\mathrm{j}})$ for the test episodes

	$t^{\mathrm{p}}=1$	$t^{\mathrm{p}}=2$	$t^{\mathrm{p}}=3$	$t^{\mathrm{p}}=4$	$t^{\mathrm{p}}=5$	$t^{\mathrm{p}}=6$	$t^{\mathrm{p}}=7$	$t^{\mathrm{p}}=8$	$t^{\mathrm{p}}=9$	$t^{\mathrm{p}}=10$	$t^{\mathrm{p}}=11$	$t^{\mathrm{p}}=12$
$P^{\mathrm{t}}(B^1\|t^{\mathrm{p}})$	0	0	0	0	0	1	1	1	1	1	1	1
$P^{\mathrm{t}}(B^2\|t^{\mathrm{p}})$	0	0	0	0	1	1	1	1	1	1	1	1
$P^{\mathrm{t}}(B^3\|t^{\mathrm{p}})$	0	0	0	0	1	1	1	1	1	1	1	1
$P^{\mathrm{t}}(B^4\|t^{\mathrm{p}})$	0	0	0	0	0	0	1	1	1	1	1	1
$P^{\mathrm{t}}(B^5\|t^{\mathrm{p}})$	0	0	0	0	0	0	0	0	1	1	1	1

Table B-2: Traffic light transition probabilities $P^{s}(B^{\mathrm{i}}|B^{\mathrm{j}})$ for the test episodes.

	B^1	B^2	B^3	B^4	B^5
$P^{s}(B^{\mathrm{i}}\|B^1)$	0	0	1	0	0
$P^{s}(B^{\mathrm{i}}\|B^2)$	0	0	0	1	0
$P^{s}(B^{\mathrm{i}}\|B^3)$	0	0	0	1	0
$P^{s}(B^{\mathrm{i}}\|B^4)$	0	0	0	0	1
$P^{s}(B^{\mathrm{i}}\|B^5)$	1	0	0	0	0

Appendix C

Figures

Visualization of linearly approximated action-value functions

An advantage of linear function approximation methods is that these methods can be visually analyzed. This gives information on which features play a significant role in the generated of the action-values. Since the traffic light block B^{i} and the action features are discrete, separate subplots are created for each unique combination of these discrete features. The features of both linear function approximation methods that describe the action-value functions can be made by scaling each feature between a predefined interval.

Polynomial functions

The visualizations of each the action-value functions that are approximated using PF can be found in Figure C-1, Figure C-2 and Figure C-3.

Radial basis functions

The visualizations of each the action-value functions that are approximated using PF can be found in Figure C-4, Figure C-5 and Figure C-6. It can be seen that the RBFs have a higher flexibility than the PFs in Figure C-1, Figure C-2 and Figure C-3.

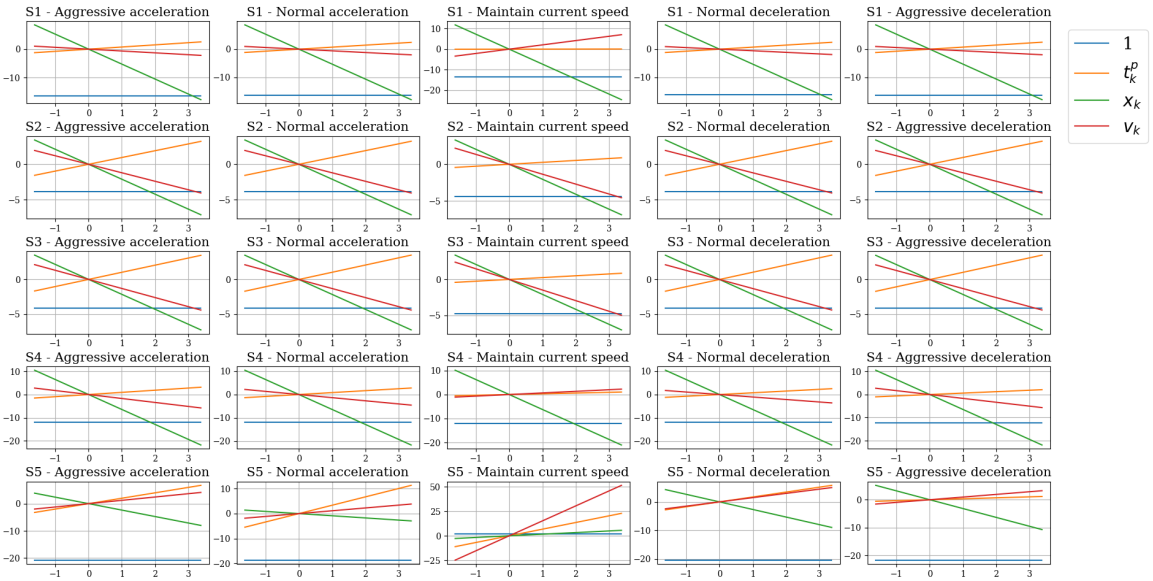

Figure C-1: Visualization of polynomial function PF1 defining the action value function at different traffic phase block-action pairs.

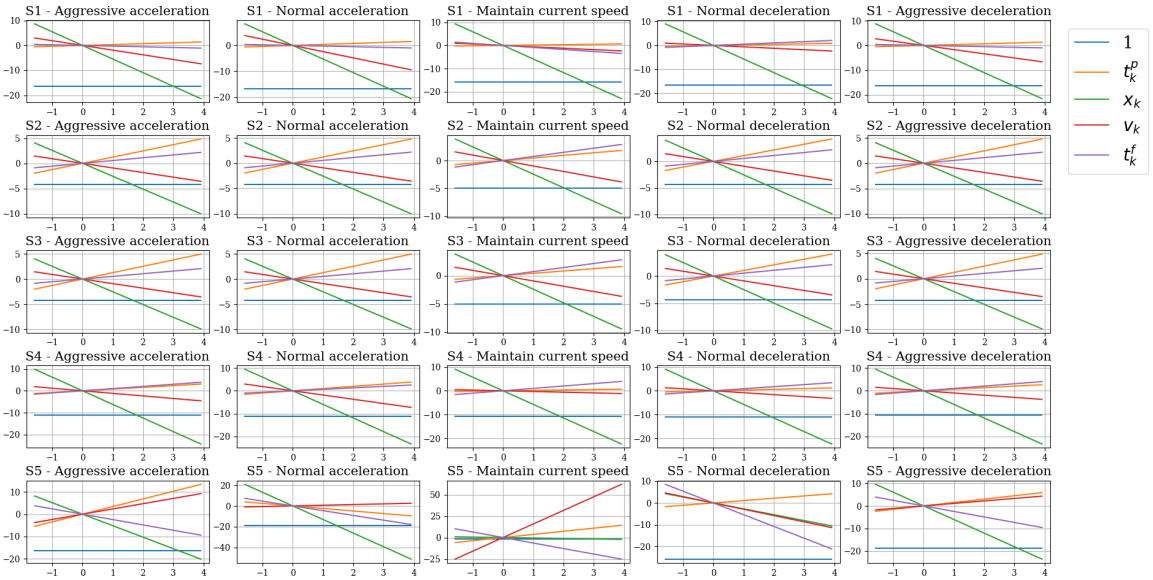

Figure C-2: Visualization of polynomial function PF2 defining the action value function at different traffic phase block-action pairs.

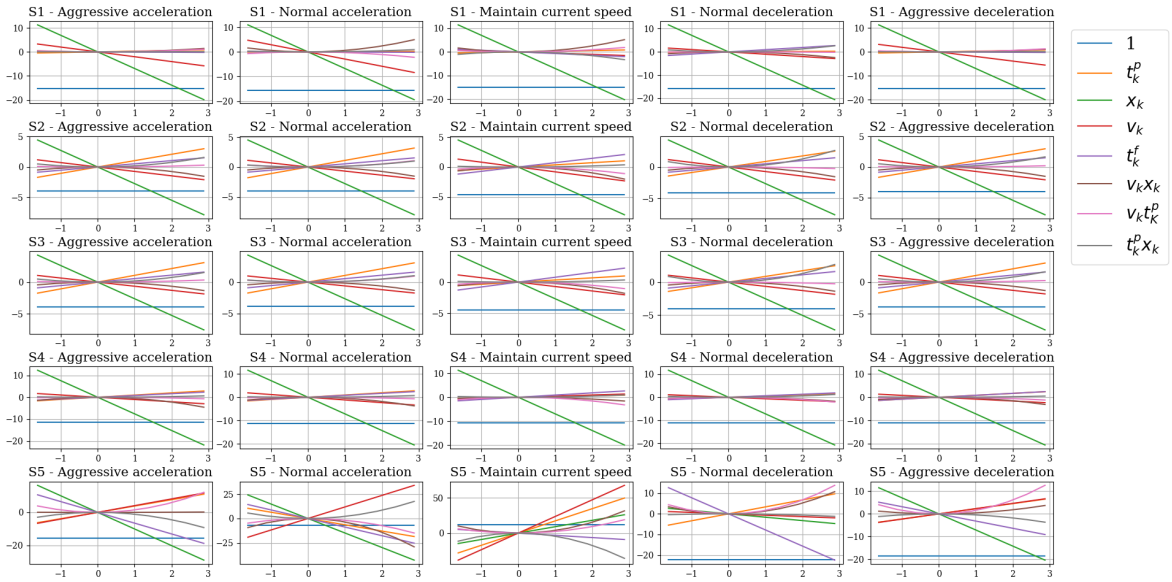

Figure C-3: Visualization of polynomial function PF3 defining the action value function at different traffic phase block-action pairs.

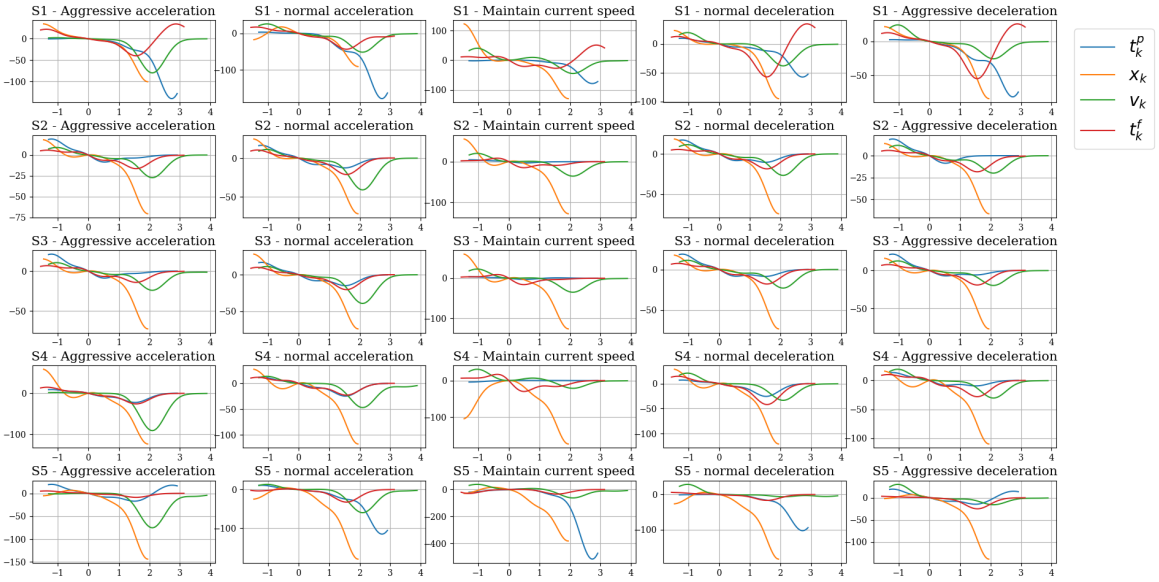

Figure C-4: Visualization of radial basis function RBF1 defining the action value function at different traffic phase block-action pairs.

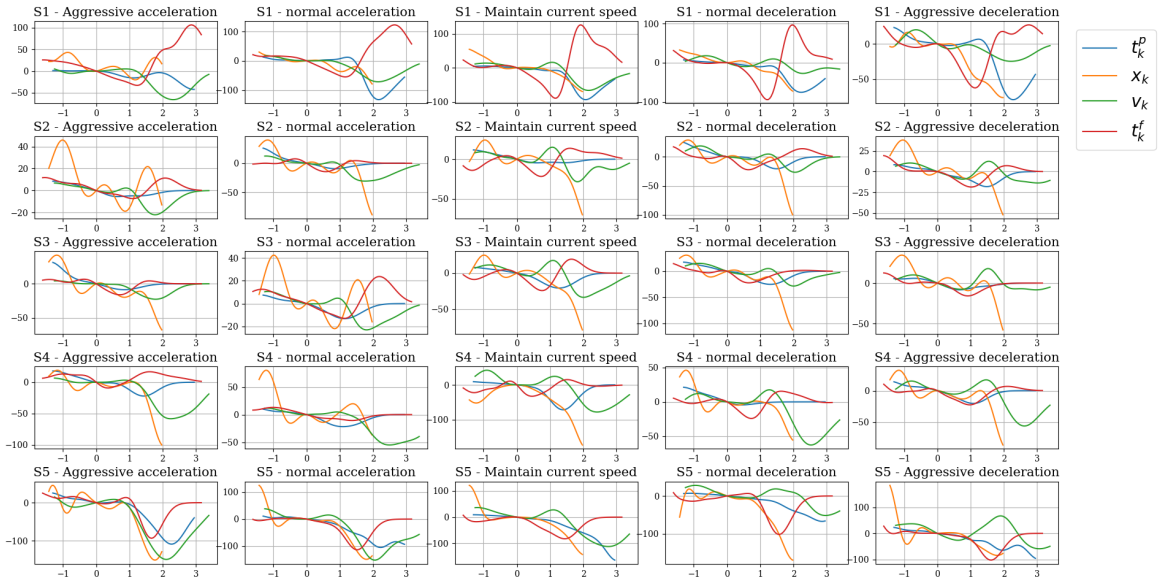

Figure C-5: Visualization of radial basis function RBF2 defining the action value function at different traffic phase block-action pairs.

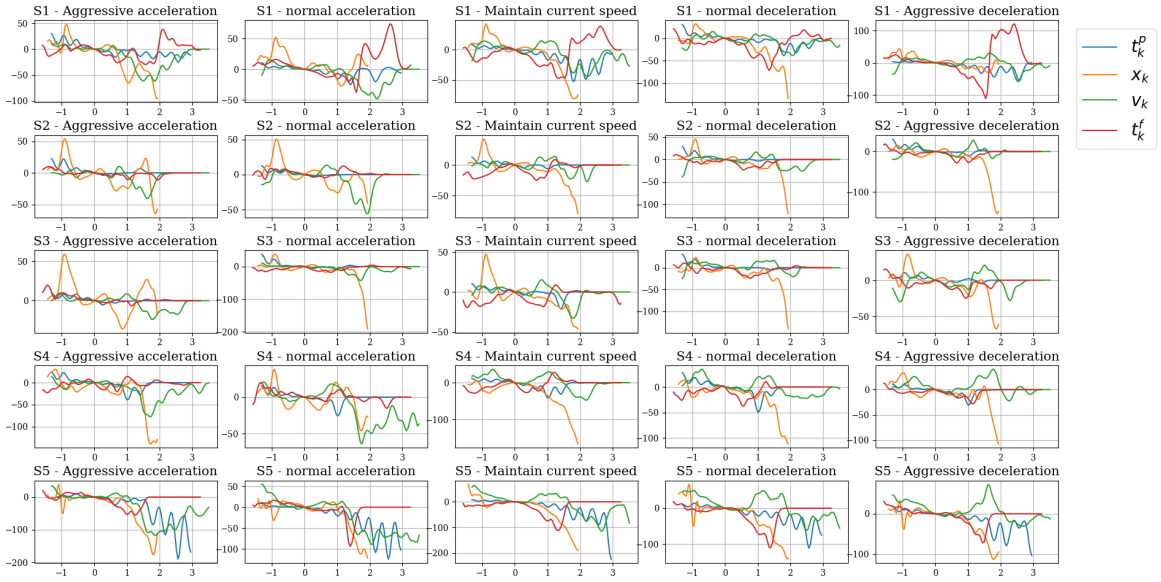

Figure C-6: Visualization of radial basis function RBF3 defining the action value function at different traffic phase block-action pairs.

Appendix D

Cyclist response model

Three different CRM-sizes are assessed based on their performance and the amount of episodes that are required to fill the CRM tables to cover a large area of the state-action space.

To test the amount of episodes that are required to fill the CRM's table, 10,000 episodes were simulated to collect cyclist response samples for the CRM. From Figure D-1, it can be seen that the amount of episodes required to fill the table scales exponentially with the size of the CRM table.

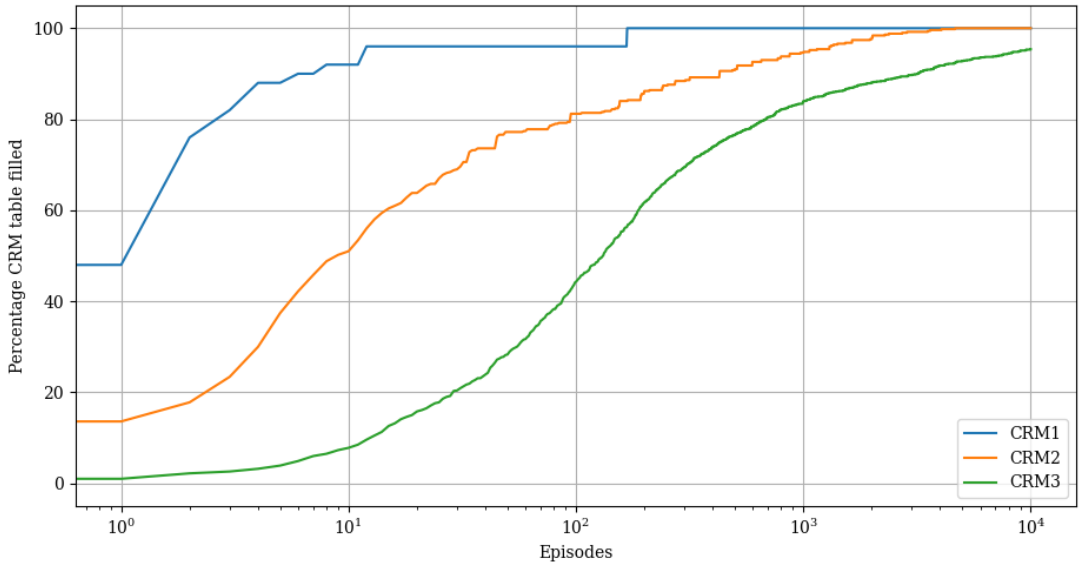

Figure D-1: Amount of episodes required to fill tables used for the cyclist response model, plotted on a logarithmic scale.

To test the accuracy of the CRM, 10,000 cyclist responses to random state-action pairs are generated. Each acceleration was counted and used to create an acceleration distribution. This experiment was repeated for CRM1, CRM2 and CRM3, but also for a full compliance

model and for the real simulated cyclist's acceleration distribution. In Figure D-2 it can be seen that CRM's with a smaller discretization steps fit the real cyclist's acceleration response more accurately than CRM's with smaller table sizes. It can be seen that each CRM is a significant improvement to the full compliance model. However, it must be noted that a resemblance in acceleration distribution does not necessarily prove an accurate approximation of the real cyclist's acceleration. A CRM could inaccurately predict cyclist response and still display an acceleration distribution resembling the real cyclist's acceleration distribution.

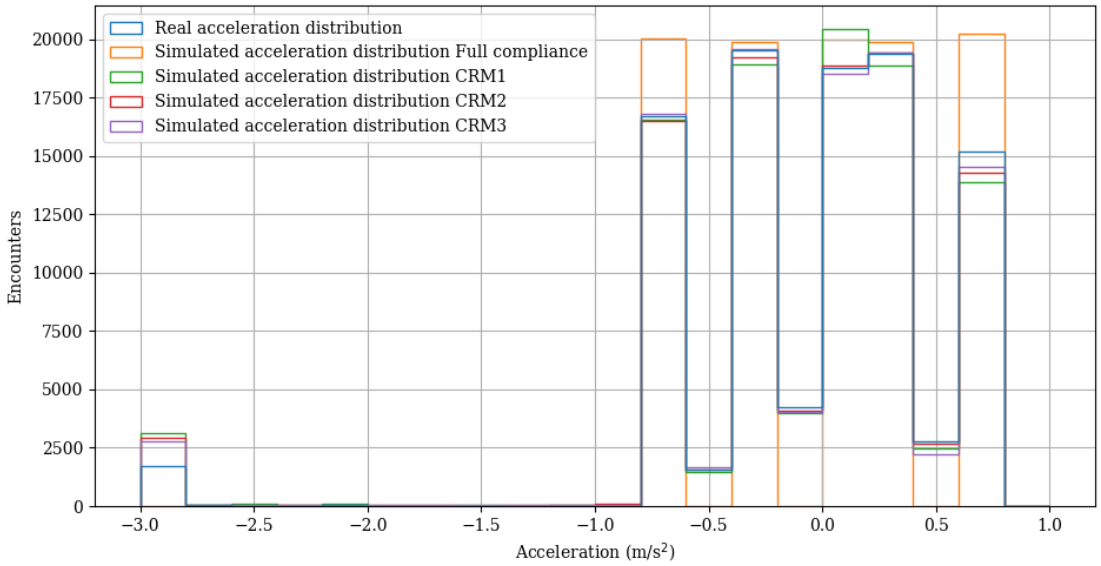

Figure D-2: Acceleration distribution of the real model, a full compliance model and the three sample-based cyclist response models.

Bibliography

[1] H. Ahmadian, J. E. Mottershead, and M. I. Friswell. Regularisation methods for finite element model updating. *Mechanical Systems and Signal Processing*, 12(1):47–64, 1998.

[2] M. Alsabaan, W. Alasmary, A. Albasir, and K. Naik. Vehicular networks for a greener environment: A survey. *IEEE Communications Surveys and Tutorials*, 15(3):1372–1388, 2013.

[3] J. Andres, T. Kari, J. Von Kaenel, and F. F. Mueller. Co-riding with my eBike to get green lights. In *Proceedings of the Designing Interactive Systems Conference*, pages 1251–1263, 2019.

[4] J. Andrew Bagnell, J. Kober, and J. Peters. Reinforcement Learning in Robotics: A Survey. In *Learning Motor Skills: From Algorithms to Robot Experiments*, volume 97, pages 9–67. Springer International Publishing, 2014.

[5] J. Apple, P. Chang, A. Clauson, H. Dixon, H. Fakhoury, M. Ginsberg, E. Keenan, A. Leighton, K. Scavezze, and B. Smith. Green driver: AI in a Microcosm. *Proceedings of the National Conference on Artificial Intelligence*, 2:1311–1316, 2011.

[6] M. Becker. Personalised speed advice for cyclists approaching intersections using reinforcement learning: A literature survey. Technical report, Delft University of Technology, 2020.

[7] R. Bellman. Markovian decision processes. *Mathematics in Science and Engineering*, 130(C):679–684, 1977.

[8] R. Blokpoel and W. Niebel. Advantage of cooperative traffic light control algorithms. In *IET Intelligent Transport Systems*, volume 11, pages 379–386, 2017.

[9] R. Bodenheimer, A. Brauer, D. Eckhoff, and R. German. Enabling GLOSA for adaptive traffic lights. In *IEEE Vehicular Networking Conference*, volume 2015, pages 167–174, 2015.

[10] L. Chen and C. Englund. Cooperative intersection management: a survey. *IEEE Transactions on Intelligent Transportation Systems*, 17(2):570–586, 2016.

[11] R. Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Proceedings of the 5th international conference on Computers and games*, pages 72–83, 2007.

[12] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.

[13] A. Dabiri and A. Hegyi. Personalised optimal speed advice to cyclists approaching an intersection with uncertain green time. In *European Control Conference*, pages 1666–1671, 2018.

[14] A. Dabiri, A. Hegyi, and B. Goñi-Ros. Optimal speed advice for cyclists using a roadside sign at signalized intersections with uncertainty in traffic light timing. *Journal of the Transportation Research Board*, 2673(7):239–247, 2019.

[15] A. Dabiri, A. Hegyi, and S. Hoogendoorn. Automatic learning of cyclist's compliance for speed advice at intersections - a reinforcement learning-based approach. In *IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2375–2380, 2019.

[16] A. Dabiri, A. Hegyi, and S. Hoogendoorn. Optimized speed trajectories for cyclists, based on personal preferences and traffic light information-a stochastic dynamic programming approach. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–17, 2020.

[17] M. De Angelis, A. Stuiver, F. Fraboni, G. Prati, V. M. Puchades, F. Fassina, D. de Waard, and L. Pietrantoni. Green wave for cyclists: Users' perception and preferences. *Applied Ergonomics*, 76:113–121, 4 2019.

[18] R. W. Denney, E. Curtis, and P. Olson. The national traffic signal report card. *National Transportation Operations Coalition (NTOC)*, 2012.

[19] J. Fajans and M. Curry. Why bicyclists hate stop signs. *UC Berkeley ACCESS Magazine*, 1(18):28–31, 2001.

[20] A. Gavriilidou, W. Daamen, Y. Yuan, and S. P. Hoogendoorn. Modelling cyclist queue formation using a two-layer framework for operational cycling behaviour. *Transportation Research Part C: Emerging Technologies*, 105:468–484, 2019.

[21] A. Gosavi. Reinforcement learning: A tutorial survey and recent advances. *INFORMS Journal on Computing*, 21(2):178–192, 2009.

[22] X. He, H. X. Liu, and X. Liu. Optimal vehicle speed trajectory on a signalized arterial with consideration of queue. *Transportation Research Part C: Emerging Technologies*, 61:106–120, 2015.

[23] G. Z. Holland, E. J. Talvitie, and M. Bowling. The effect of planning shape on Dyna-style planning in high-dimensional state spaces. *arXiv*, 2019.

[24] P. Jing and R. J. Williams. Efficient learning and planning within the Dyna framework. *Adaptive Behavior*, 1(4):437–454, 1993.

[25] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations - Conference Track Proceedings*, 2015.

[26] E. Koukoumidis, L. S. Peh, and M. R. Martonosi. SignalGuru: Leveraging mobile phones for collaborative traffic signal schedule advisory. In *Proceedings of the 9th International Conference on Mobile Systems, Applications and Services*, pages 127–140. ACM Press, 2011.

[27] P.S. Lin, A. Fabregas, and H. Chen. Impact of detection and communication degradation on traffic signal operations. Technical report, Center for Urban transportation Research at university of South Florida, 2009.

[28] M. Lu, R. Blokpoel, and M. Joueiai. Enhancement of safety and comfort of cyclists at intersections. In *IET Intelligent Transport Systems*, volume 12, pages 527–532, 2018.

[29] V. C. Magana and M. Munoz-Organero. Artemisa: A Personal Driving Assistant for Fuel Saving. *IEEE Transactions on Mobile Computing*, 15(10):2437–2451, 10 2016.

[30] G. Mahler and A. Vahidi. Reducing idling at red lights based on probabilistic prediction of traffic signal timings. In *Proceedings of the American Control Conference*, pages 6557–6562, 2012.

[31] G. Mahler and A. Vahidi. An optimal velocity-planning scheme for vehicle energy efficiency through probabilistic prediction of traffic-signal timing. *IEEE Transactions on Intelligent Transportation Systems*, 15(6):2516–2523, 2014.

[32] S. Mandava, K. Boriboonsomsin, and M. Barth. Arterial velocity planning based on traffic signal information under light traffic conditions. In *Proceedings of the IEEE Conference on Intelligent Transportation Systems*, pages 160–165, 2009.

[33] M. Miyatake, M. Kuriyama, and Y. Takeda. Theoretical study on eco-driving technique for an electric vehicle considering traffic signals. In *Proceedings of the International Conference on Power Electronics and Drive Systems*, pages 733–738, 2011.

[34] V. Mnih, K. Kavukcuoglu, and D. Silver. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[35] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130, 1993.

[36] R. Nalavde and A. Surve. Driver assistant services using ubiquitous smartphone. In *Proceedings of the 2015 International Conference on Communication and Signal Processing*, pages 430–434, 2015.

[37] C. W. Pai and R. C. Jou. Cyclists' red-light running behaviours: An examination of risk-taking, opportunistic, and law-obeying behaviours. *Accident Analysis and Prevention*, 62:191–198, 2014.

[38] J. Parkin and J. Rotheram. Design speeds and acceleration characteristics of bicycle traffic for use in planning, design and appraisal. *Transport Policy*, 17(5):335–341, 9 2010.

[39] B Peng, X. Li, J. Gao, J. Liu, and K. F. Wong. Deep Dyna-Q: Integrating planning for task-completion dialogue policy learning. In *Association for Computational Linguistics*, volume 1, pages 2182–2192, 2018.

[40] V. Protschky, S. Feit, and C. Linnhoff-Popien. Extensive traffic light prediction under real-world conditions. In *IEEE Vehicular Technology Conference*. Institute of Electrical and Electronics Engineers Inc., 11 2014.

[41] J. Pucher and R. Buehler. Making Cycling Irresistible: Lessons from The Netherlands, Denmark and Germany. *Transport Reviews*, 28(4):495–528, 7 2008.

[42] N. Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.

[43] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachusetts, second edition, 2018.

[44] H. Rakha and R. K. Kamalanathsharma. Eco-driving at signalized intersections using V2I communication. In *Proceedings of the IEEE Conference on Intelligent Transportation Systems*, pages 341–346, 2011.

[45] G. Reggiani, A. Dabiri, W. Daamen, and S. Hoogendoorn. Clustering-based methodology for estimating bicycle accumulation levels on signalized links: a case study from the Netherlands. *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019*, pages 1788–1793, 2019.

[46] R. A. Retting. Establishing a uniform definition of red-light running crashes. *Institute of Transportation Engineers*, 76(3):20–22, 2006.

[47] S. Ruder. An overview of gradient descent optimization algorithms. *Insight Centre for Data Analytics*, 3:1–14, 2016.

[48] W. J. Schakel and B. Van Arem. Improving traffic flow efficiency by in-car advice on lane, speed, and Headway. *IEEE Transactions on Intelligent Transportation Systems*, 15(4):1597–1606, 2014.

[49] P. Schepers, D. Twisk, and E. Fishman. The Dutch road to a high level of cycling safety. *Safety Science*, 92:264–273, 2017.

[50] M. Seredynski, B. Dorronsoro, and D. Khadraoui. Comparison of green light optimal speed advisory approaches. In *Proceedings of the 16th IEEE Conference on Intelligent Transportation Systems*, pages 2187–2192, 2013.

[51] D. Silver, A. Huang, and C. J. Maddison. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529, 2016.

[52] D. Silver, R. S. Sutton, and M. Müller. Sample-based learning and search with permanent and transient memories. In *Proceedings of the 25th International Conference on Machine Learning*, pages 968–975, New York, New York, USA, 2008. Association for Computing Machinery (ACM).

[53] D. K. Smith and D. P. Bertsekas. Dynamic Programming and Optimal Control. *The Journal of the Operational Research Society*, 47(6):833, 1996.

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

[55] R. Stahlmann, M. Möller, A. Brauer, R. German, and D. Eckhoff. Technical evaluation of GLOSA systems and results from the field. In *Proceedings of the IEEE Vehicular Networking Conference*, 2016.

[56] S. Su, X. Li, J. Gao, J. Liu, and Y. Chen. Discriminative deep Dyna-Q: robust planning for dialogue policy learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 3813–3823, 2018.

[57] R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991.

[58] R. S. Sutton, C. Szepesvari, A. Geramifard, and M. P. Bowling. Dyna-style planning with linear function approximation and prioritized sweeping. *In Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, pages 528–536, 2012.

[59] R. S. Sutton, C. Szepesvári, and H. R. Maei. A convergent O(n) algorithm for off-policy temporal-difference learning with linear function approximation. In *Advances in Neural Information Processing Systems*, pages 1609–1616, 2009.

[60] S. Theodoridis and K. Koutroumbas. *Pattern recognition and neural networks*. Springer, 2001.

[61] S. Thrun and A. Schwartz. Issues in using function approximation for reinforcement learning. *In Proceedings of the 4th Connectionist Models Summer School Hillsdale*, pages 1–9, 1993.

[62] T. Tielert, M. Killat, H. Hartenstein, R. Luz, S. Hausberger, and T. Benz. The impact of traffic-light-to-vehicle communication on fuel consumption and emissions. In *Internet of Things*, 2010.

[63] R. S. Trayford, B. W. Doughty, and J. W. van der Touw. Fuel economy investigation of dynamic advisory speeds from an experiment in arterial traffic. *Transportation Research Part A: General*, 18(5-6):415–419, 1984.

[64] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double Q-Learning. In *AAAI Conference on Artificial Intelligence*, pages 2094–2100, 2016.

[65] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[66] J. Wollaeger, S. A. Kumar, S. Onori, D. Filev, U. Ozguner, G. Rizzoni, and S. Di Cairano. Cloud-computing based velocity profile generation for minimum fuel consumption: A dynamic programming based solution. In *Proceedings of the American Control Conference*, pages 2108–2113, 2012.

[67] Y. Zhao, S. Li, S. Hu, L. Su, S. Yao, H. Shao, H. Wang, and T. Abdelzaher. GreenDrive: A smartphone-based intelligent speed adaptation system with real-time trafiic signal prediction. In *Proceedings of the 8th IEEE International Conference on Cyber-Physical Systems*, pages 229–238, 2017.

[68] Y. Zhao, S. Yao, H. Shao, and T. Abdelzaher. CoDrive: cooperative driving scheme for vehicles in urban signalized intersections. In *IEEE International Conference on Cyber-Physical Systems*, pages 308–319, 2018.

[69] Z. Zhou, X. Li, and R. N. Zare. Optimizing chemical reactions with deep reinforcement learning. *ACS Central Science*, 3(12):1337–1344, 2017.

[70] M. Zweck and M. Schuch. Traffic light assistant: Applying cooperative ITS in European cities and vehicles. In *Proceedings of the 2nd International Conference on Connected Vehicles and Expo*, pages 509–513, 2013.

Glossary

List of Acronyms

RL	reinforcement learning
TD	temporal-difference
MDP	Markov decision process
ISAS	intelligent speed advisory systems
SGD	stochastic gradient descent
PF	polynomial function
RBF	radial basis function
ANN	artificial neural networks
CRM	cyclist response model

List of Symbols

RL algorithm parameters

α	Learning rate
α^{s}	Simulated learning rate
$\alpha^{\mathrm{min^s}}$	Simulated minimum learning rate
α^{min}	Minimum learning rate
α_0	Initial learning rate
α_0^{s}	Simulated initial learning rate
ϵ	Exploration rate
ϵ^{s}	Simulated exploration rate

$\epsilon^{\min^{\mathrm{s}}}$	Simulated minimum exploration rate
$\epsilon^{\min}$	Minimum exploration rate
ϵ_0	Initial exploration rate
ϵ_0^{s}	Simulated initial exploration rate
γ	Discount factor
$\hat{Q}$	Approximated action-value function
$\overline{\mathbf{w}}^{\mathrm{l}}$	Transient local weight vector
$\overline{\mathbf{w}}^{\mathrm{t}}$	Transient target weight vector
$\mathbf{w}^{\mathrm{l}}$	Local weight vector
$\mathbf{w}^{\mathrm{t}}$	Target weight vector
$\mathcal{A}$	Action space
$\mathcal{S}$	State space
$\mathcal{S}^{\mathrm{c}}$	Cyclist state space
$\mathcal{S}^{\mathrm{l}}$	Traffic light state space
$\overline{\hat{Q}}$	Approximated transient action-value function
$\overline{\hat{Q}}^{\mathrm{l}}$	Approximated local transient action-value function
$\overline{\hat{Q}}^{\mathrm{t}}$	Approximated transient target action-value function
$\bar{a}$	Transient action
$\bar{a}'$	Transient future action
$\bar{s}$	Transient state
$\bar{s}^{\mathrm{eb}}$	Transient emergency brake state
$\bar{s}^{\mathrm{t}}$	Transient terminal state
$\bar{s}'$	Transient future state
ϕ	Feature vector
σ	Kernel width parameter
τ	Soft target update parameter
a	Action
a'	Future action
C	Test episode interval
c	Kernel center location
$d^{\alpha^{\mathrm{s}}}$	Simulated decay rate learning rate
d^{α}	Decay rate learning rate
$d^{\epsilon^{\mathrm{s}}}$	Simulated decay rate exploration rate
d^{ϵ}	Decay rate exploration rate
n^{k}	Number of kernels
n^{s}	Number of simulated episodes per time step
Q	Action-value function
s	State
s^{t}	Terminal state
s'	Future state

s^{eb}	Emergency brake state
y	Activation function output
z	Weighted sum of perceptron's input features
$\overline{Q}$	Transient action-value function

Reward function symbols

$\bar{r}$	Transient reward
F^{e}	Energy factor
F^{i}	Idling factor
F^{s}	Safety factor
F^{a}	Advice factor
F^{ds}	Desired speed factor
F^{h}	High speed factor
F^{l}	Instability factor
R	Total reward
r	Reward
R^{a}	Advice reward
R^{i}	Idling reward
R^{s}	Safety reward
R^{t}	Travel time reward
R^{h}	High speed reward
R^{l}	Instability reward
W^{e}	Energy weight
W^{i}	Idling weight
W^{s}	Safety weight
W^{a}	Advice weight
W^{ds}	Desired speed weight
W^{h}	High speed weight
W^{l}	Instability weight

Physical parameters

Δt	Time step size
$\overline{B}^{\text{i}}$	Transient traffic light phase block i
$\bar{t}^{\text{p}}$	Transient traffic light phase time
ρ	Air density (kg/m^3)
$\tilde{u}$	Suggested acceleration (m/s)
A_{f}	Frontal surface of the cyclist (m^2)
B^{i}	Traffic light phase block i
C_{ad}	Aerodynamic drag constant
C_{rr}	Rolling resistance coefficient of the wheels
e	Slope of the road (rad)

E_{ac}	Cyclist's energy consumption associated with acceleration (J)
E_{ad}	Cyclist's energy consumption associated with aerodynamic drag (J)
E_{cyc}	Cyclist's total energy consumption (J)
E_{rr}	Cyclist's energy consumption associated with rolling resistance (J)
E_{rs}	Cyclist's energy consumption associated with road slope (J)
E_{max}	Cyclist's maximum total energy consumption (J)
g	Gravitational constant (m/s^2)
k	Time step counter
k^{t}	Terminal time step
k^{eb}	Emergency brake time step
m^{c}	Mass of the cyclist (kg)
m^{w}	Rotational mass of the wheels (kg)
t	Time
t^{p}	Traffic light phase time
t^{eb}	Required time to come to a full stop when using an emergency brake
t^{eta}	Estimated time of arrival
t^{f}	Focus time
t^{ttg}	Time until the next green traffic light phase
u	Cyclist's acceleration (m/s^2)
u^{max}	Cyclist's maximum acceleration (m/s)
v	Cyclist's speed (m/s)
v^{w}	Rotational speed of the wheels (r/s)
v^{ds}	Cyclist's desired speed (m/s^2)
v^{high}	Calibration run high speed threshold (m/s^2)
v^{h}	Cyclist's high speed threshold (m/s^2)
v^{low}	Calibration run low speed threshold (m/s^2)
v^{l}	Cyclist's instability speed threshold (m/s^2)
v^{med}	Calibration run desired speed threshold (m/s^2)
v^{max}	Cyclist's maximum speed (m/s)
x	Cyclist's position
x^{tl}	Position of the traffic light
x^{max}	Position of the end point of the episode
Z_{++}	Very fast speed zone
Z_{+}	Fast speed zone
Z_{--}	Very slow speed zone
Z_{-}	Slow speed zone
$Z_{\pm}$	Desired speed zone