Longitudinal Control for Autonomous Vehicles

A comparison between Reinforcement Learning and Optimal Control

G.L.G.J. Faassen

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Longitudinal Control for Autonomous Vehicles

A comparison between Reinforcement Learning and Optimal Control

Master of Science Thesis

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

G.L.G.J. Faassen

March 29, 2019

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

The work in this thesis is supported by BMW Group. Their cooperation is hereby gratefully acknowledged.

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Longitudinal Control for
Autonomous Vehicles

by

G.L.G.J. Faassen

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: March 29, 2019

Supervisor(s):

Dr. M. Alirezaei

L. Puccetti M.Sc.

Reader(s):

Prof. dr. ir. J. Hellendoorn

Dr. R. Ferrari

Abstract

In the automotive industry automation is popular and every year car OEMs advance their technology to be able to drive autonomously. Longitudinal control of the vehicles is an important part of the complete autonomous driving system. The difficulty of this control problem lies with changing longitudinal dynamics and the lack of full-state system information. This complicates controller design when using classic model-based approaches such as Optimal Control (OC). Currently the controllers are still manually tuned by control engineers in the vehicle. This is time consuming and expensive, therefore other methods for controller design such as learning are explored. Reinforcement Learning (RL) is one of those methods.
To examine the potential benefits of learning a controller, this work will make a comparison between RL and OC. For RL, an actor-critic structure using deterministic policy gradient is applied. Due to partially observable system dynamics OC is used as an optimal output feedback controller. The comparison complies speed control of an autonomous vehicle. The RL agent will learn a controller by training on a nonlinear high fidelity vehicle model.
In this work it was demonstrated that RL can reach the same performance as OC when all environmental settings are comparable. When environmental settings deviate, it was is found that RL outperforms OC. To verify the simulated results all controllers were confirmed in an experimental real-life setting.
In conclusion, this proved a promising benefit of learning with respect to classical controller computation, when dealing with partially available system information.

Table of Contents

Acknowledgements **ix**

1 Introduction **1**

1-1 Related Work . . . 2

1-2 Research Objective . . . 3

1-3 Thesis Outline . . . 3

2 Theoretical Framework **5**

2-1 Optimal Control . . . 5

2-1-1 Classical Optimal Control . . . 5

2-1-2 Optimal Output Control . . . 6

2-2 Reinforcement Learning . . . 7

2-2-1 Agent-Environment Interaction . . . 7

2-2-2 Policy . . . 8

2-2-3 Reward Function . . . 8

2-2-4 Value Function . . . 8

2-2-5 Function Approximation . . . 9

2-2-6 Actor-Critic . . . 11

2-3 Summary . . . 13

3 Controller Design for Longitudinal Vehicle Dynamics **15**

3-1 Vehicle Model . . . 15

3-2 Reinforcement Learning Set-up . . . 17

3-3 Optimal Output Control Benchmark . . . 19

3-4 Summary . . . 20

4 Simulation Study **21**
4-1 Learning the Controller . . . 21
4-2 Results . . . 24
4-2-1 Driving at 20 km/h on a flat road . . . 25
4-2-2 Driving at 40 km/h on a flat road . . . 28
4-2-3 Driving at 20 km/h on a road with 10% inclination . . . 32
4-3 Discussion . . . 34

5 Experimental Comparison **37**
5-1 Experimental Preliminaries . . . 37
5-2 Results . . . 38
5-2-1 Driving at 20 km/h . . . 38
5-2-2 Driving at 40 km/h . . . 40
5-3 Discussion . . . 41

6 Conclusion **43**
6-1 Future Work . . . 44

A Optimal Control **47**
A-1 Regular Optimal Control . . . 47
A-2 Output Feedback Control . . . 47

B Reinforcement Learning Solving Methods **51**
B-1 Value Function Methods . . . 52
B-1-1 Monte Carlo . . . 52
B-1-2 Dynamic Programming . . . 52
B-1-3 Temporal Difference . . . 53
B-2 Policy Methods . . . 54
B-2-1 Monte Carlo . . . 55
B-2-2 Dynamic Programming . . . 55
B-2-3 Policy Gradient . . . 56

Glossary **63**
List of Acronyms . . . 63

List of Figures

2-1 The agent-environment interaction in Reinforcement Learning 7
2-2 Model of one node 10
2-3 Fully connected network with one hidden layer 11
2-4 The Actor-Critic Architecture 12

3-1 Nonlinear high fidelity vehicle model with RL agent 16
3-2 Detailed Agent Structure 17
3-3 Bent Identity Activation Function 18
3-4 Example Quadratic Feature Layer 18
3-5 TD error of network with QFL and FC layer 19

4-1 Neural Network Architecture 22
4-2 Lookup table for time constant τ 24
4-3 Development of learned gain and accumulated rewards per test run at 20 km/h . 25
4-4 Development of state response whilst learning optimal gain at 20 km/h 26
4-5 Performance of RL and OC in altered system at 20 km/h 27
4-6 State response RL and OC in altered system at 20 km/h 28
4-7 Development of learned gain and accumulated rewards per test run at 40 km/h . 29
4-8 Development of state response whilst learning optimal gain at 40 km/h 29
4-9 Performance of RL and OC in altered system at 40 km/h 30
4-10 State response RL and OC in altered system at 40 km/h 31
4-11 Development of learned gain and accumulated rewards per test run at 20 km/h with 10% inclination 33
4-12 State response RL and OC in altered system at 20 km/h with 10% inclination . . 33

5-1 State response of driving 20 km/h 39
5-2 State response of driving 40 km/h 40

List of Tables

4-1 Settings of the Agent 23
4-2 Environmental Settings Vehicle Model for two cases. 23
4-3 Values of RL and OC at 20 km/h. 25
4-4 Values of RL, OC and $OC_{altered}$ at 20 km/h. 27
4-5 Values of RL and OC at 40 km/h. 29
4-6 Values of RL, OC and $OC_{altered}$ at 40 km/h. 30
4-7 Values of RL and OC at 20 km/h with 10% inclination 33

5-1 Corresponding gains for driving 20 km/h in second gear. 38
5-2 Corresponding gains for driving 40 km/h in third gear. 38

Acknowledgements

I would like to thank BMW Group in Munich for giving me the opportunity to conduct my research at their Autonomous Driving Campus. Because of it I was able to discuss my work and that of others with fellow students and colleagues which has been very interesting. Also was I able to experience the latest advancements in the field on first hand. A special word of gratitude goes to my supervisor Luca Puccetti M.Sc. for always being patient and helpful in times I did not understand how my learning algorithm had learned to solve itself.

From Delft University of Technology I like to express my appreciation for my supervisor Dr. Mohsen Alirezaei who, despite the distance, always found the patience to discuss my problems over Skype. Also from Delft I would like to thank my committee members, Prof. dr. ir. J. Hellendoorn and Dr. R. Ferrari.

Also I would like to thank my friends here and in Munich for keeping me company in the biergartens and during the oktoberfest. Last but not least, I want to thank my parents for their endless support and believe.

Delft, University of Technology — G.L.G.J. Faassen
March 29, 2019

Chapter 1

Introduction

Due to general developments in automation and successive DARPA challenges in 2004, 2005 and 2007, research on autonomous driving has gained growing interests. By excluding humans from the driving loop, autonomous vehicles are expected to increase road safety as well as improve the road transport efficiency [1, 2].

The autonomous driving system consists of various separate driving functions. Control of the vehicle's longitudinal dynamics is among these functions. As the vehicle follows a certain trajectory, its longitudinal dynamics change and therefore its inherent control changes as well. To account for these system changes, the controller has to be set for every possible situation separately by composing a gain-schedule. Currently, this schedule is constructed manually by control engineers in the vehicle. As manual tuning of the gains is time consuming and expensive, other methods for longitudinal control are explored. In this work a classical control theory method in the form of Optimal Control (OC) and a novel Machine Learning (ML) method, Reinforcement Learning (RL) will be discussed and compared.

OC is a model-based method, it therefore requires full-state feedback information for optimal design [3, 4]. Unfortunately, full-state information is not always available. In the vehicle some signals, like acceleration, are too noisy to be useful for feedback and others such as engine temperature or load, are variable. This makes the vehicle system only partially observable. Partial feedback information complicates the controller design with model-based methods such as OC. ML methods, such as RL, could offer a solution when dealing with partially observable and changing system dynamics.

In recent years RL has gained popularity in the ML community through the likes of autonomously playing and mastering games like Atari, Chess, Shogi and Go [5, 6, 7]. RL bases its learning scheme on the learning behaviour of humans and animals. It consists of an agent that interacts with an environment, i.e. the system. The environment is a black-box to the agent, therefore full-state information is not needed for learning. The agent tries to find an optimal path in the environment by gaining experience from the interaction [8]. Every step, the agent obtains a reward signal from the environment, this reward is a quantitative value representing the action's *success*. The agent, therefore, learns the consequences of its actions

by *trial-and-error*; it eventually learns which state-action combination yields the maximum reward.

Consider a case where an infant is learning how to walk. It knows that it can get from point A to point B by crawling. At a given moment it tries to stand up and succeeds; standing yields some form of reward so from now on it knows how to do it. From the upward standing position it tries to walk but will inevitably fall down the first time it tries. Though the infant has fallen down, it knows how to get up so it can try again from standing up and it does not need to start from crawling all over again. The infant gets feedback from the environment, interpreting the information through its actions and their consequences. The ability of channelling this information is learning.

1-1 Related Work

Other research that has been conducted in the field of longitudinal control include speed control, i.e. Cruise Control (CC), and speed control with a leading vehicle, i.e. Adaptive Cruise Control (ACC). While solving these control problems, various approaches have been investigated. To obtain a general idea of what methods are already used for longitudinal control, the most relevant and recent achievements are summarized. Furthermore, RL solutions specifically focussed on this topic have been examined. Last, comparable RL methods used in non-automotive applications are discussed.

Addressing both the CC and ACC problem: Zhao *et al.* introduce a novel variation on adaptive dynamic programming [9] and Camacho *et al.*, Li *et al.* and Schmied *et al.* propose various forms of model predictive control methods [10, 11, 12]. However, the proposed methods mostly rely on specific traffic scenarios. This is a drawback as highly automated driver assist functions are in need of a much broader scope of traffic scenarios [13]. Another disadvantage of model predictive control methods is their need for proper estimation of the system dynamics. Polack *et al.* suggest a model-free method to determine a low-level controller for an autonomous vehicle. They approximate the longitudinal dynamics of the vehicle by an ultra-local differential relation of order 1 [14].

As RL has proven to be able to find close-to-optimal solutions compared to model predictive approaches [15], it has also been implemented in longitudinal control problems for automated driving. Zhu *et al.* use RL to derive an optimal acceleration strategy for ACC with respect to a lead vehicle [16]. Desjardins *et al.* approach a Cooperative Adaptive Cruise Control (CACC) system with RL. A CACC system uses Vehicle-to-Vehicle (V2V) or Road-to-Vehicle (R2V) communication [17]. Xu *et al.* apply a parameter tuning method for learning a PI-controller for application in CC [18]. This method uses a Least Squares Policy Iteration to update the controller parameters. Bischoff *et al.* use a model-based policy search algorithm named PILCO to learn throttle valve control [19]. Considering these approaches, emphasis in this work is on speed control and learning a proportional controller using the Deterministic Policy Gradient (DPG).

RL has been used in a various settings for PID parameter auto tuning also in non-automotive context. Wang *et al.* propose an algorithm to automatically tune a PID controller using a Radial Basis Function (RBF) neural network with Temporal Difference (TD) learning [20]. On basis of the previous proposition, comparable work in a more application focus way has

been done. Sedighizadeh *et al.* have also used an adaptive PID controller with an RBF network updated through TD learning but applied on wind turbine control [21]. Kashki *et al.* use an automated PID tuning method to optimise a controller for an automatic voltage regulator [22]. The controller optimisation makes use of CARLA, an algorithm that relies on policy iteration. El Hakim *et al.* control soccer robots through PID, these controllers are self tuned using a multi-agent system with Q-learning [23].

1-2 Research Objective

As many driving scenarios can occur, longitudinal control of an autonomous vehicle requires a properly derived gain-schedule. Various model-based methods for longitudinal control have been proposed. These methods rely on proper system information, which is not always available. Learning a controller through a model-free approach can therefore offer a potential solution.

In this thesis a comparison between two controllers for speed control will be made. A controller will be derived using the available system information via OC. Another controller will be learned using RL. This results in the following research question:

"Can Reinforcement Learning be beneficial for longitudinal control of autonomous vehicles?"

1-3 Thesis Outline

The thesis is structured in a total of six chapters.

Chapter 2 contains preliminary information. First a brief explanation on OC and how it changes into Optimal Output Control (OOC) when only limited state information is available will be given. Secondly important concepts for understanding RL will be explained.

Chapter 3 comprises the controller design. First will be explained how the system is defined. Then the implementation of the RL-framework will be discussed. Last, the implementation of OC will be explained.

Chapter 4 presents and discusses the results of the comparison between RL and OC interacting with a simulated vehicle model.

Chapter 5 presents and discusses the results of an experimental comparison done between RL and OC in a real vehicle.

Chapter 6 will conclude on the work and recommendations on future work will be given.

Chapter 2

Theoretical Framework

In this chapter the theoretical preliminaries are presented. Reinforcement Learning (RL) will be compared with a benchmark controller in the form of Optimal Control (OC). It is explained how OC behaves when only limited state information is fed back. Further an elaboration is given on what RL is, what its main components are and how it works.

The controller in the real vehicle operates in discrete time. Therefore, concepts in this work will mostly be explained in discrete time.

2-1 Optimal Control

"The objective of optimal control theory is to determine the control signals that will cause a process to satisfy the physical constraints and at the same time minimize some performance criterion." - Donald E. Kirk [3]

For optimal controller design the complete set of state variables is required. Unfortunately, in the vehicle, certain signals are too noisy where others are variable rendering them useless as feedback. When only partial state information is available, the approach of how an optimal controller is derived changes. The difference between classical full-state feedback control and its partial-state feedback control counterpart, i.e. output feedback control, will be discussed.

2-1-1 Classical Optimal Control

"To determine a control signal that satisfies physical constraints", meaning a control law has to be found that drives a system to an equilibrium. Describing a linear time-invariant system in discrete state-space form [4]

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k. \end{aligned} \tag{2-1}$$

The performance criterion is formulated as a quadratic cost function through a linear-quadratic regulator [24]

$$J = \sum_{k=0}^{\infty} x_k^T Q x_k + u_k^T R u_k. \tag{2-2}$$

Here Q and R are weighting matrices that determine the importance of certain states and inputs. The optimisation problem is then defined by minimising this quadratic cost function

$$\min_K J. \tag{2-3}$$

From this minimisation an optimal control law is derived

$$u_k = -K x_k, \tag{2-4}$$

where K is the controller gain.

2-1-2 Optimal Output Control

In the case of partial state feedback the optimisation problem changes, the control law is now given by

$$u_k = -K_y y_k. \tag{2-5}$$

In this control law K_y represents the controller gain in the case where only certain observable states, i.e. the output, are are considered. The optimisation problem now becomes

$$\min_{K_y} J \tag{2-6}$$

with

$$J = x_0^T \Bigg[\sum_{k=0}^{\infty} ((A - BK_yC)^T)^k (Q + C^T K_y^T R K_y C)(A - BK_yC)^k \Bigg] x_0. \tag{2-7}$$

Now the solution to the optimisation problem depends on the initial state x_0, i.e. $K_y(x_0)$. A way to mathematically eliminate the controller's dependence on the initial state is to average the performance for a set of initial states that are linearly independent [25, 26]. This could mean assuming that the initial state is a random variable which is evenly distributed on the surface of the unit circle. The average performance is then expressed as

$$\hat{J} = \text{trace}\Bigg(\sum_{k=0}^{\infty} ((A - BK_yC)^T)^k (Q + C^T K_y^T R K_y C)(A - BK_yC)^k \Bigg). \tag{2-8}$$

The optimal controller parameters can then be determined through an iterative process, e.g. via Algorithm 1 in Appendix A.

Further elaboration on the elimination of the initial state-dependence and how the optimal output control parameters are calculated is given in Appendix A.

2-2 Reinforcement Learning

An RL agent learns how to solve sequential decision making problems through interaction with an environment and gathering experience via *trial-and-error* [8]. How the agent knows where to go and how it is able to tell how *good* it has performed thus far will be explained.

2-2-1 Agent-Environment Interaction

RL considers an agent that interacts with an environment (Figure 2-1). As in the presented example in chapter 1, the infant represents the agent and the room where it is trying to stand up and walk is the environment. At each time step t, the agent obtains an impression of the environment's state $s_t \in \mathcal{S}$, $\mathcal{S}$ being the set of possible states. The agent then determines a suitable action $a_t \in \mathcal{A}(s_t)$, where $\mathcal{A}(s_t)$ is the set of possible actions available in s_t. Suppose the agent executes action a_t, through this interaction the agent receives reward $r_{t+1} \in \mathcal{R}$ and next state s_{t+1} from the environment. In this case, standing is the infant's reward and the upward position will then be the next state. Multiple iterations like this form an episode. In episodic tasks the objective for the agent is to reach a terminal state up to which the reward is maximised [8, 27].

Here state s_t and action a_t are the same as the systems observable states y_k and input u_k respectively (Equation 2-1). For unity and clarity the notation from RL, s_t and a_t, will be used henceforth.

The agent-environment interaction in Reinforcement Learning

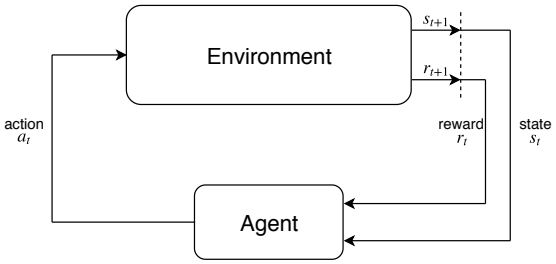


Figure 2-1: The agent receives state and reward on time step t, s_t and r_t respectively. A suitable action a_t, is returned to the environment, which then gives a new state and reward for time step $t+1$, s_{t+1} and r_{t+1} respectively.

Markov Decision Process As in this work the RL algorithm is considered to be discrete, the problem it solves, i.e. the environment, can be modelled as a Markov Decision Process (MDP) [28, 29]. An MDP can be defined as:

Definition 2-2.1. *A **Markov Decision Process** is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, which consists of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a state transition probability function $\mathcal{P}$ and a reward function $\mathcal{R}$.*

Markovian dynamics rest on the idea that information given by the current state is enough to yield an optimal decision, formally known as the Markov Property [27]. Meaning that the agent takes action a_t in state s_t to transit to state s_{t+1} which yields immediate reward r_{t+1}.

2-2-2 Policy

The behaviour of the agent is determined by the policy π. The policy dictates which action is taken while being in a certain state. It maps states to actions, exactly like the control law in Equation 2-5 [8, 27]. For the presented example in chapter 1, the infant's inputs are the movement of its arms and legs. Its initial policy for getting from A to B is crawling and the optimal policy is walking. To reach the optimal policy the infant moves about, i.e. explores different states. Through exploration it finds out how to stand. As standing is a more optimal policy, its policy changes accordingly. Eventually by exploring various possibilities from standing, the infant is able to walk and its policy changes again to the optimal policy. Distinction can be made between a deterministic and stochastic policy π. A deterministic policy π is a function in the form of $\pi : \mathcal{S} \to \mathcal{A}$, defined as

$$\pi(s) = a. \tag{2-9}$$

While a stochastic policy π is in the form of $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, defined as the probability distribution

$$\mathbb{P}(a|s) = \pi(s, a). \tag{2-10}$$

In this work a deterministic policy will be used as its form coincides with a linear controller.

2-2-3 Reward Function

The agent tries to learn the optimal policy, but to do so it has to know *how good* it is to be in a certain state or to take a certain action. The notion of how good states and actions are is defined by the reward r_t. The reward is a single value that varies from step to step and the agent tries to maximise the accumulated reward, exactly as how in optimal control the cost has to be minimised. The reward is specified by the reward function. This function is fixed and embedded in the environment, it cannot be altered by the agent [8, 27].

2-2-4 Value Function

The reward gives an immediate notion of *how good* it is to be in a certain state or take a certain action. More important is the value function, it describes what the best course of action for the agent is over time. It represents the accumulated rewards which the agent has received up to the point of evaluation. Noticeable is that the value function is exactly the same as the cost function in optimal control. Most RL algorithms rely on the estimation of the value function to learn an optimal policy [8, 27] .

The state-value function, $V^\pi(s)$, denotes the expected accumulated reward for starting in state s, and following policy π. This is formulated as

$$V^\pi(s) = \mathbb{E}_\pi\Big\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \Big\}. \tag{2-11}$$

Here $\mathbb{E}_\pi\{\cdot\}$ represents the agent's expected value when following policy π and γ is a discount factor that can be used to put less weight on rewards further in the future. In a similar fashion the state-action-value function $Q^\pi(s,a)$, denotes the expected return for taking action a, starting in state s and following policy π from the next time step on. This is formulated as

$$Q^\pi(s,a) = \mathbb{E}_\pi\Big\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \Big\}. \tag{2-12}$$

The state-action-value function is the same as the state-value function with an additional degree of freedom taken into consideration. Both value functions can be connected as follows

$$V^\pi(s) = Q^\pi(s, \pi(s)). \tag{2-13}$$

As the state-action-value function will primarily be used, further concepts will be explained with it. For readability it will be called just the value function. To determine an optimal policy, an optimal value function has to be estimated [30]. When the optimal policy is denoted by π^* the optimal value function can be described as:

$$Q^*(s,a) = \max_\pi Q^\pi(s,a). \tag{2-14}$$

In this optimal value function the action a is selected according to the optimal policy. The optimal value function Q^* can be also be expressed as the Bellman optimality equation

$$Q^*(s,a) = \mathbb{E}\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, \pi(s_{t+1})) | s_t = s, a_t = a\}. \tag{2-15}$$

2-2-5 Function Approximation

As optimal control calculates the cost function, in RL the value function has to be estimated. The cost function and value function are important parts of the optimisation problem as the are the performance measures. Therefore the form of these functions defines how well the system is controlled [3, 8]. In OC, as described in the previous section, the cost is calculated by a pre-defined cost function [4, 24]. In RL the form of the value function is not pre-defined and therefore a function approximator is needed to estimate the function [31]. Advancements in Artificial Intelligence (AI) make it possible for Artificial Neural Networks (ANNs) to learn and solve complex problems [32]. Function approximation is among such problems and therefore an ANN is implemented in the RL framework for the estimation of the value function [33, 34].

Artificial Neural Networks ANNs are inspired loosely on the biological neural network architecture of the human brain [35, 36]. The concept is that neurons are connected via synapses and signals are passed between and through them [37]. In an ANN these neurons are represented by nodes and are modelled as functions. Nodes can take an arbitrary number of inputs, those inputs are then weighted and put through an input function, usually a summation. When the weighted inputs are summed a bias is included and to obtain the output the

result is passed through an activation function. A node can then be mathematically expressed as

$$o_i = \phi(\sum_{j=1}^{n} w_{i,j} p_j + b_i). \tag{2-16}$$

Where the output of the i-th node is o_i. The total amount of inputs is represented by n and the j-th input is p_j.The weights and biases are given by $w_{i,j}$ and b_i respectively, and ϕ is the activation function. The relationship that Equation 2-16 describes is illustrated in Figure 2-2.

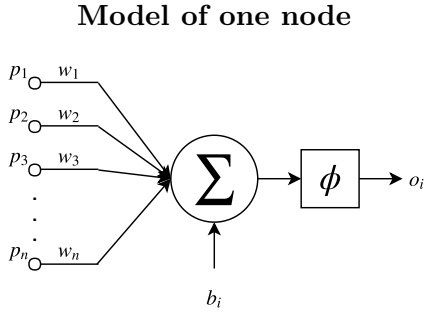


Figure 2-2: One node with n inputs and activation function ϕ.

The activation function can be seen as an abstraction of the rate of the action potential firing in a cell [36]. The activation function determines the behaviour of the node. In its simplest form the activation function is the identity function, this is a linear activation function. But as problems that are presented to ANNs can be of various difficulties, also the activation functions vary [38]. When approximating nonlinear functions, a linear activation as the above mentioned identity function does not suffice and nonlinear activation functions are used to introduce nonlinearity to the approximation [39].

By connecting nodes together in a layered structure a network can be formed (Figure 2-3). A network always consists of at least two layers, an input and an output layer respectively. Layers in between (if any) are called hidden layers. The output of the nodes in a given layer are the inputs for the subsequent layer and so forth. The signals passed through the network can only go in one direction and networks like these are therefore feedforward. Neural networks with an architecture of this form are also called multilayer perceptron or vanilla neural networks [37, 40]. Even though the network only consists of local connectivities, by adding nodes and layers the network as a whole is able to acquire all the information needed to approximate the desired function. The network is said to be fully connected when all nodes of one layer, are connected to all the nodes of the next layer.

To update the network's weights and biases to approximate the desired function better, a loss function is defined. The loss function is used to measure the deviation between the network's predicted value and the actual value. Then, using back propagation, the loss is propagated backward through the network to adjust the weights and biases in such a way that it minimises the error it represents. How the network's weights are updated using RL is explained in the next section and which type of activation function are considered and tested is discussed in chapter 3 [38].

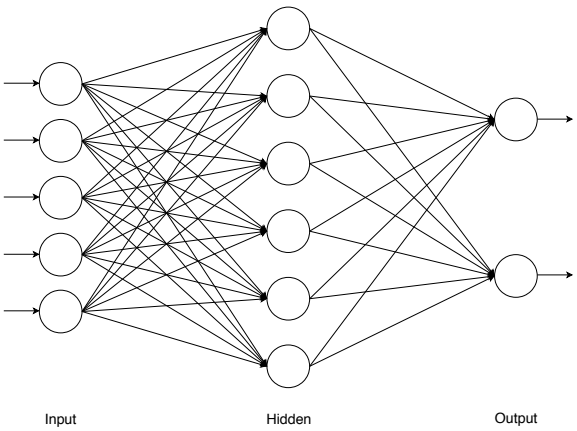


Figure 2-3: Example of a fully connected neural network consisting of an input layer with 5 inputs, one hidden layer with 6 nodes and an output layer with 2 outputs.

2-2-6 Actor-Critic

Within RL there are various approaches to learning optimal policies or value functions. Three methods can be distinguished [8]:

– Critic only

– Actor only

– Actor-Critic

Critic only or value function methods, try to estimate and optimise the value function and define the policy only implicitly. Actor only or policy methods determine the policy directly from the obtained rewards. Actor-critic methods can be seen as a combination of both value function and policy methods. As in this work actor-critic is used only this method will be explained, the interested reader is referred to Appendix B for more on critic only and actor only methods.

The actor-critic framework defines the agent in an evaluation and a control part. Here the critic is the evaluating part which estimates the value function. The actor is the control part, containing the policy and thus determining how to act in a given state. The goal in RL is to obtain an optimal policy, to achieve this a properly estimated value function is needed. For a proper value function a good policy is required, in this way critic and actor work together to reach optimal performance. As can be seen in Figure 2-4 the critic determines the value function and passes a notion of it to the actor, which uses this to update its policy. Then the actor determines a control input that yields a reward and thus an adjustment of the value function. In the given case the Deterministic Policy Gradient (DPG) is used and therefore the critic passes the gradient of value function to the actor [8, 27, 41].

The critic tries to approximate and evaluates the value function. To properly learn value function it must find a solution to the Bellman equation (Equation 2-15). To do this it needs to minimise the difference between the right and left hand size of the equation, i.e. the Bellman residual, or the Temporal Difference (TD) error. The TD-error can be defined as

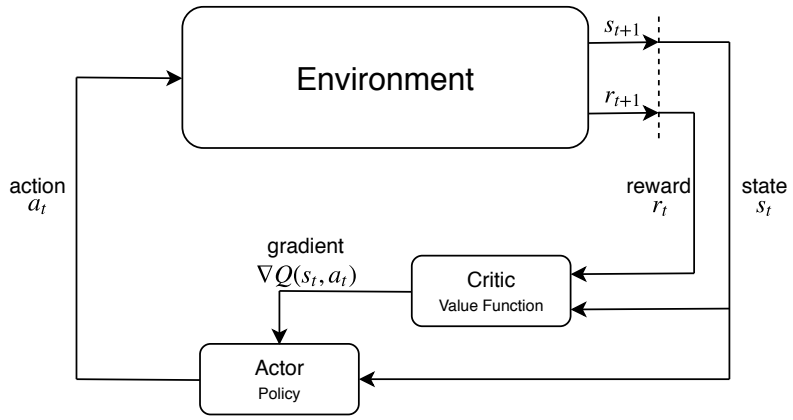


Figure 2-4: The critic learns and evaluates the value function where after it passes its gradient on to the actor. The actor uses this gradient to learn a more optimal policy where after it determines which action to take in a certain state.

$$\delta_{TD} = r(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1}, \theta) - Q(s_t, a_t, \theta). \tag{2-17}$$

Here θ is a parameter vector that contains the weights and biases of the neural network. This network sits inside the critic and represents the approximation of the value function. To get a better estimate of the value function, a loss function is determined using the TD-error, the parameter vector is then updated as follows

$$\text{loss} = 0.5\, \delta_{TD}^T \delta_{TD}, \tag{2-18}$$

$$\nabla_\theta \text{loss} = \delta_{TD} \nabla_\theta \delta_{TD}. \tag{2-19}$$

$$\nabla_\theta \text{loss} = \delta_{TD} \nabla_\theta (-Q(s_t, a_t, \theta)). \tag{2-20}$$

From the gradient of this loss function the new parameter vector can be obtained

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta \text{loss}. \tag{2-21}$$

Where α is a learning rate and $\alpha \nabla_\theta loss$ is a Vanilla Gradient Descent optimiser, chosen for simplicity but any optimiser of choice can be used. When the critic has learned the next iteration of the value function, it evaluates the function and passes its gradient $\nabla Q(s_t, a_t)$ on to the actor. The actor also consists of a learn and an act part. It uses the value function to learn the policy. It also does this by updating a parameter vector ϑ which represent the policy, i.e. the controller. To improve the policy in the actor, the DPG is used [42]. To evaluate the policy, the gradient of the expected return J is calculated

$$\begin{aligned}\nabla_\vartheta J(\pi_\vartheta) &= \nabla_\vartheta \mathbb{E}_s[\mathbb{E}_a[Q(s_t, a_t)]], \\ &= \nabla_\vartheta \mathbb{E}_s[[Q(s_t, \pi_\vartheta(s_t))]], \\ &= \mathbb{E}_s[\nabla_a Q(s_t, a_t)|_{a=\pi_\vartheta(s_t)} \nabla_\vartheta \pi_\vartheta(s_t)].\end{aligned} \tag{2-22}$$

This gradient is then used to update the parameter vector

$$\vartheta_{t+1} = \vartheta_t - \alpha \nabla_\vartheta J(\pi_\vartheta). \tag{2-23}$$

Where again α is a learning rate and $\alpha \nabla_\vartheta J(\pi_\vartheta)$ is a Vanilla Gradient Descent optimiser, chosen for simplicity but any optimiser of choice can be used.

2-3 Summary

In this chapter the theory used in this work is explained. As not all the state information is available, regular OC cannot be applied to derive a controller. Optimal Output Control (OOC) is, in contrast to regular OC, dependant on the initial state of the system. To eliminate this dependency the performance is averaged for a set of initial states that are evenly distributed on the unit circle.

RL considers an agent that gathers experience by interacting with an environment. This agent consists of two parts, an actor and a critic. As the agent explores the environment the critic forms the state-action-value function from observed data. This value function represents the accumulated rewards in each state from an initial state and it expresses a notion of how *good* it is for the agent to be in a certain state and execute a certain action. The actor then determines the agent's actions according to the policy. The purpose of the RL algorithm is to derive an optimal policy, critic and actor work together to achieve this.

Function approximation is an important part of any optimisation problem. In OC the cost function that is to be minimised is expressed in a quadratic form. In RL the value function needs to be estimated and it does not have a predefined form. Therefore a function approximator in the form of an artificial neural network is used.

Chapter 3

Controller Design for Longitudinal Vehicle Dynamics

Before a controller can be applied in a vehicle for commercial use, extensive testing in simulation environments is required. In this chapter the two models, one used for design and one for testing the controllers will be addressed. The models are inferred from a BMW test vehicle. The Reinforcement Learning (RL) set-up for learning the controller will be discussed, this includes the reward function, the network configuration and state reconstruction for better learning performance. Also the implementation of the Optimal Control (OC) controller will be explained.

3-1 Vehicle Model

The environment is an important part of the RL algorithm. In this work, a linearised vehicle model as well as a high fidelity nonlinear model are used.

Nonlinear Model The detailed nonlinear model[1] consists of two main components: The vehicle dynamics model and a low-level controller (Figure 3-1). The vehicle dynamics include driving resistances, sensor noise and communication delays. The low-level controller linearises effect of the drive-train, i.e. combustion engine, hydraulic brakes and automatic transmission. The low-level controller obtains speed input and outputs drive or brake torque to the vehicle dynamics. The model is build for the simulation of longitudinal dynamics at low speeds. The simulations that are executed are speed control only with a proportional controller. To mimic the real world setting the acceleration is not observed, only speed is fed back.

[1]The model is based in the BMW 7 series, the most used test vehicle at the autonomous driving campus.

Nonlinear high fidelity vehicle model with RL agent

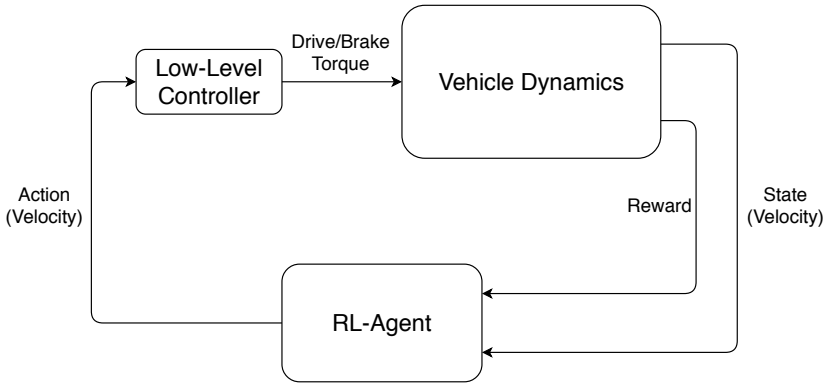


Figure 3-1: Simulation model including low-level controller and RL agent.

Linearised Model From the nonlinear vehicle model a linearised state-space model is deducted to establish the first design of the agent and to determine the optimal controller. The state-space is based on a second order transfer function that represents the linearised drive-train and is given by (both are used in a discrete form but for explanatory reasons are here stated in continuous time):

$$G(\xi) = \frac{1}{\tau^2\xi^2 + 2\tau\xi + 1}. \tag{3-1}$$

Which translates to

$$\dot{x} = \begin{bmatrix} 0 & 1 \\ \frac{-1}{\tau^2} & \frac{-2}{\tau} \end{bmatrix} x + \begin{bmatrix} 0 \\ \frac{1}{\tau^2} \end{bmatrix} u. \tag{3-2}$$

The first state represents the acceleration, the second is an internal state where time constant τ defines the behaviour of the drive-train. To mimic the real world situation the velocity needs to be observed so an integrator is implemented. From this we infer

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & \frac{-1}{\tau^2} & \frac{-2}{\tau} \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{\tau^2} \end{bmatrix} u. \tag{3-3}$$

Where measurements of only the first state, the velocity, are available

$$y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} x. \tag{3-4}$$

3-2 Reinforcement Learning Set-up

As stated in subsection 2-2-6, the actor and critic in the agent approximate the control law and value function respectively. Therefore they both consist of a learning part and an evaluative/act part. Inside the agent a ring buffer is implemented to store the visited states, performed actions and their corresponding rewards, i.e. the experience [43]. The experience storage contains M tuples of states, actions and rewards from which the actor and critic randomly samples N_a and N_c tuples respectively for learning. The implementation of the buffer in the actor is depicted in Figure 3-2.

Detailed Agent Structure

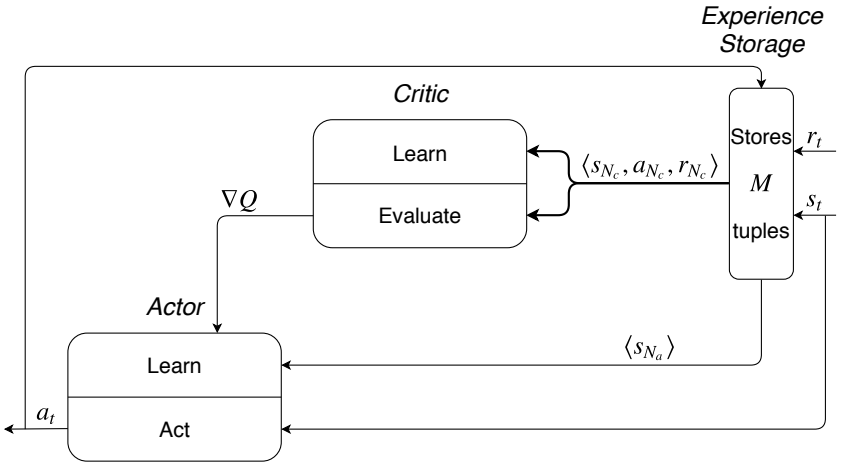


Figure 3-2: Detailed actor-critic structure inside the agent. The experience storage stores M tuples of states, actions and rewards. The critic samples N_c of these tuples for learning, whereas the actor samples N_a tuples of states to learn. Then, with the current state, the next action is determined following the policy in the actor.

Reward Function To match the quadratic cost function of OC, the reward function is also defined in a quadratic form. It represents the sum of the squared observable state and weighted control action

$$r(y_k, u_k) = -y_k^T y_k - 0.1 u_k^T u_k, \tag{3-5}$$

in RL notation it is

$$r(s_t, a_t) = -s_t^T s_t - 0.1 a_t^T a_t. \tag{3-6}$$

Network Configuration The estimation of the value function depends on the neural network. As discussed in subsection 2-2-5, the amount of nodes, layers and which type of activation is used are significant. With an eye on more complicated future controller design and for the

sake of a RL algorithm in the most general form, it makes sense to implement a function approximator with nonlinear properties and universal approximation capabilities. Throughout the years various activation functions have been tested on different problems and as of 2018, rectifier functions such as Rectified Linear Unit (ReLU) activation function are the most popular [44, 45, 46]. An altered form of the rectifier unit is the Bent Identity activation function, it can be mathematically described by Equation 3-7 and is visualised in Figure 3-3.

$$\phi(p_t) = \frac{\sqrt{p_t^2 + 1} - 1}{2} + p_t \tag{3-7}$$

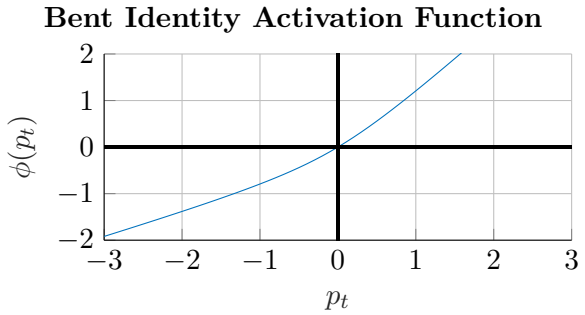


Figure 3-3: Bent Identity Activation Function.

For linear systems with quadratic reward function, e.g. Equation 3-6, it is known that the infinite horizon state value function is also quadratic [47]. Therefore, besides a network with only Fully Connected (FC) layers and nonlinear activation, also a network with a layer including quadratic function approximation is considered. In this work called a Quadratic Feature Layer (QFL). In this layer every input is multiplied with itself and every other input. For every n inputs in this layer there are $\frac{n(n+1)}{2}$ outputs and there are no weights or biases, an example is given in Figure 3-4. After the QFL, a fully connected layer outputs the weighted values of the QFL to one node, i.e. the value function.

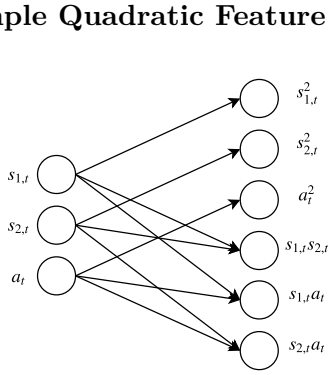


Figure 3-4: Example of a quadratic feature layer with three inputs and six outputs.

To compare the performance of both networks, with the QFL and the FC layer, they are tested with the linear model described by Equation 3-3 and Equation 3-4. The network's goal is to estimate the value function and thereby minimising the Temporal Difference (TD)

error. Then, comparing the TD-error each network yields, a conclusion can be drawn upon their performance. The network that yields to the smallest error, performs best. In this test the network with a QFL has the configuration: 2-3-1, the network with a FC layer has the configuration 2-30-1. As can be observed in Figure 3-5 the TD-error of the QFL is much lower than the error to which the FC layer converges. The TD-error of the FC layer converges to an order of 10^{-2} whereas the TD-error for the QFL converges to an order of 10^{-9}. Because the QFL yields a more promising result it will be used henceforth.

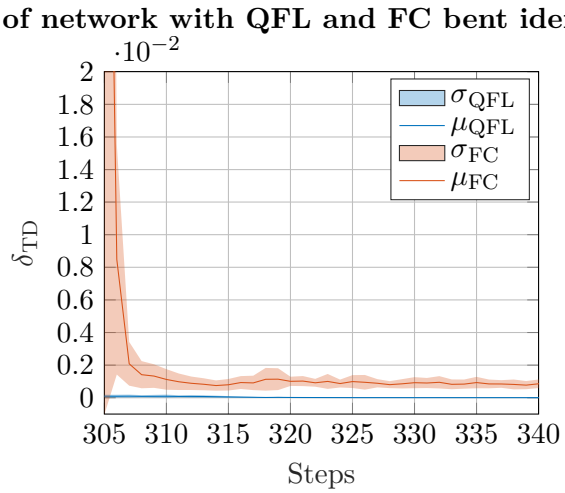


Figure 3-5: The TD-error of both a network with the QFL and FC layer is shown. Both networks have run 10 times with 500 episodes and 100 steps per episode, the plot is 50.000 steps long but is truncated for clarity. Due to the fact the agent does not learn when it builds up the experience in the buffer the first error is plotted at 305.

State Reconstruction As the system dynamics are only partially observable and to further improve the critic's estimation of the value function, one state is reconstructed. The state is constructed by a learned observer inside the critic, it consists of a fully connected layer that approximates the missing state via a Finite Impulse Response (FIR) filter on past actions. The FIR filter uses back propagation to restore the missing state information. In this case it only reconstructs the second state, and not the third, as this unobserved information has highest influence on the observable state. To reconstruct the second state 40 previous actions are appended to the input of the network. This amount of actions is enough for the finite impulse response to settle within a bound so that the state can be reconstructed and the observer is not over fitting. For a more elaborate explanation, please refer to Puccetti *et al.* [48].

3-3 Optimal Output Control Benchmark

For a fair comparison, the optimal output controller has to be set up following the same specifications as the RL agent. As stated in section 2-1 the cost function is specified by

$$\hat{J} = \text{trace}\Big(\sum_{k=0}^{\infty} ((A - BK_yC)^T)^k (Q + C^T K_y^T R K_y C)(A - BK_yC)^k \Big). \tag{3-8}$$

The continuous system matrices A, B and C are given by Equation 3-3 and Equation 3-4, they are discretised following standard state-space discretisation [3]. The OC cost function will be weighted according to the RL reward function, as stated by Equation 3-6. Therefore weighing matrices Q and R are

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad R = 0.1. \tag{3-9}$$

Algorithm 1 in Appendix A then states how the optimal output controller gain is derived.

3-4 Summary

In this chapter the vehicle models used for training and testing are described. Furthermore is explained how the acRL controller is learned and the OC controller is derived.

A high fidelity nonlinear model is used to simulate the vehicle. This model consists of the vehicle dynamics and a low-level controller. The vehicle dynamics include driving resistance, sensor noise and communication delays. The low-level controller linearises the effects of the drive-train. For both RL and OC only the velocity is observable.

The agent consists of an actor and a critic which in term both consist of a learning and an evaluative/act part. The reward that the agent obtains from the environment is from a quadratic reward function that has the same form as the OC cost function. A buffer inside the agent stores tuples of states, actions and rewards on which the actor and critic learn their policy and value function respectively. In the critic a neural network approximates the value function. Two types of network are taken into consideration, a FC network with nonlinear activation functions called bent identity and a QFL with quadratic function approximation. Both are tested on the linear system. The TD-error is considered as a measure of performance for the networks. The QFL yields a TD-error in the order of 10^{-9} where as the FC layer settles at TD-errors of the order 10^{-2}. Therefore the network with QFL is chosen to continue with. To further improve learning a state reconstruction is performed. This reconstruction uses a fully connected layer with a FIR filter on past actions to approximate one missing state of the system, i.e. the acceleration.

For the implementation of an Optimal Output Control (OOC) specific weighing matrices are needed. These matrices are set in such a way the cost function matches the RL reward function. Q is a zero-matrix where only position (1,1) has a value of 1, because the only signal that passed through for feedback is velocity. The value of R is set to 0.1.

Chapter 4

Simulation Study

As described in the previous chapter, a nonlinear vehicle model is used to learn a controller with the RL agent in a simulation. In this chapter the environmental settings such as speed, gear and sampling time will be discussed. Also the agents configuration of network, batch and episode size will be specified. After that the results of learning in the simulated model will be discussed.

4-1 Learning the Controller

For the RL agent to be able to learn, some parameters have to be set in the environment and in the agent. Important components to consider in both actor and critic are the structure of the network, the optimiser used to update the weights and biases in the network and the size of the storage buffer, batch and episodes.

Agent In the actor the network is not as extended as in the critic, here it represents the value of the linear controller gain. It contains one input, being the current state s_t, and one output, a_t, the weight that links these two is the controller gain K_p. The critic's network has a more extensive structure:

- The input consists of the current observable state, s_t, 40 appended previous actions, $a_{t-40}, ..., a_{t-1}$ and the current action, a_t.
- The first layer
  - Passes through the current observable state
  - Passes the appended previous actions through a fully connected layer with linear activation functions to reconstruct the missing state
  - Passes through the current action
- The second layer passes all three inputs through a quadratic feature layer

– The third layer passes all six inputs through a fully connected layer with linear activation functions to form the state-action value function

This network architecture is illustrated in Figure 4-1.

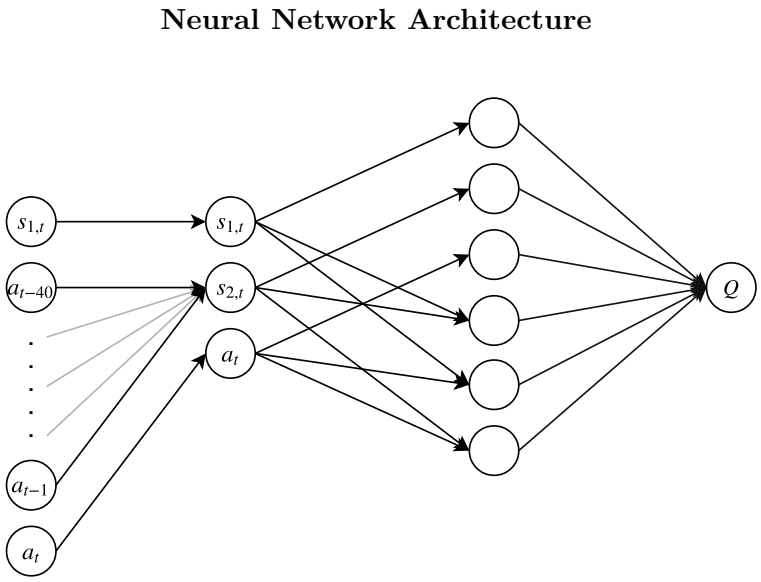


Figure 4-1: A schematic display of the network's configuration with FIR filter, Quadratic Feature Layer and Fully Connected layers.

The optimiser determines how fast the network's parameters can converge to their optimal value. As the actor and the critic work together, their respective performance is linked. It is important to choose proper optimisers to suit both learning trajectories. Meaning that the the optimiser in the critic has to be chosen to be as fast as possible. The optimiser in the actor on the other hand cannot be too fast. As the actor only has to learn one value, the approximation of the value function in the critic might not be able to keep up and it can end up not learning at all. This causes the agent to perform suboptimal. Therefore as optimiser in the critic the Levenberg-Marquardt (LM) algorithm is used and in the actor the Stochastic Gradient Descent (SGD) algorithm.

The storage buffer contains past state and actions from which the actor and critic sample. The buffer stores the 500 last state-action pairs and their rewards. For every step the critic samples a random batch of 300 tuples of states, actions and rewards from the buffer and pushes them through the network. Then using temporal difference learning, the weights of the network are adjusted to be a better fit of the state-action-value function Q. In a similar way, the actor samples a random batch of 100 tuples of states to update its weight, i.e. the controller gain, following the Deterministic Policy Gradient (DPG) method. The length of each episode is set at 140 steps. This is because the state reconstructor in the critic needs 40 past actions to be able to reconstruct the unobserved state. As in the 40 first steps nothing is learned, the episodes are effectively a 100 steps long. Learning the value function in the critic is done with a discount of 0.95. The discount is needed for the rewards to converge to a finite sum. In addition it determines how rewards that lie in the future are valued, more discount places less emphasis on the future and more on immediate reward [49]. It is desirable

to consider rewards as far in the future as possible, therefore the discount factor needs to be high. In the current system setting a discount of 0.95 was the maximum for which the agent was still able to learn properly. Table 4-1 summarizes these settings.

Table 4-1: Settings of the Agent

Parameter	**Actor**	**Critic**
Optimiser	Stochastic Gradient Descent	Levenberg-Marquardt
Storage Buffer (same for both)	500 tuples	500 tuples
Batch Size	100 tuples	300 tuples
Steps per Episode	140 steps	140 steps
Discount factor	-	0.95

Environment In the environment, i.e. the vehicle model, there are only certain parameters that can be modified, other influences like load, drag coefficient or engine specifications are fixed. The main environmental parameters are shown in Table 4-2.

Table 4-2: Environmental Settings Vehicle Model for two cases.

Parameter	**Symbol**	**Case 1**	**Case 2**
Speed	$s_{velocity}$	20 km/h	40 km/h
Gear	g	2	3
Time constant	τ	0.910	0.632
Sampling time	T	0.02	0.02

Two cases for the environment are set, first driving 20 km/h in second gear, and secondly driving 40 km/h in third gear. The sampling time is set to 20 milliseconds as this is the sampling frequency of the controller in the actual vehicle. An important note to make here is the time constant τ mentioned in chapter 3. In the vehicle model τ is incorporated via a lookup table and is therefore not a fixed value as it depends on the revolutions of the engine. Though τ varies slightly during the run using a lookup table, it converges to a specific value as the agent is able to stabilise the system. This τ is then used to determine the optimal output controller, as can be seen in Equation 3-3 the time constant has a big influence on the system dynamics and is therefore crucial to the determination of the gain. The range of τ according to the lookup table is plotted in Figure 4-2.

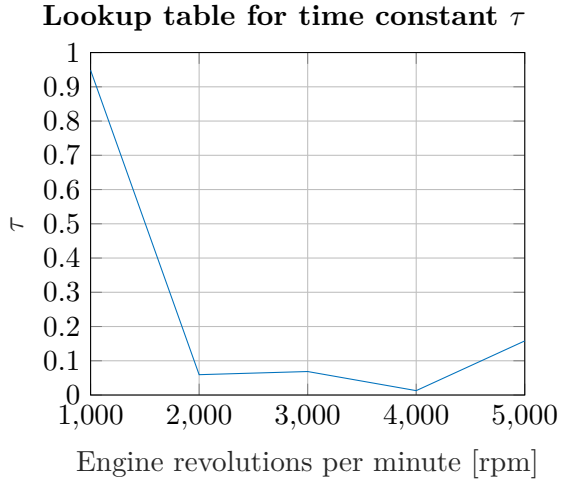


Figure 4-2: The normalised lookup table for time constant τ that is used by the vehicle model.

4-2 Results

To compare the behaviour of RL and OC in different environmental scenarios and to see if model-free learning has an advantage over model-based control, three scenarios are presented.

– Driving at 20 km/h on a flat road.
  - Agent will be trained for driving in second gear and compared to OC set for driving in second gear.
  - Agent will be trained for driving in second gear and compared to OC set for driving in first gear.

– Driving at 40 km/h on a flat road.
  - Agent will be trained for driving in third gear and compared to OC set for driving in third gear.
  - Agent will be trained for driving in third gear and compared to OC set for driving in second gear.

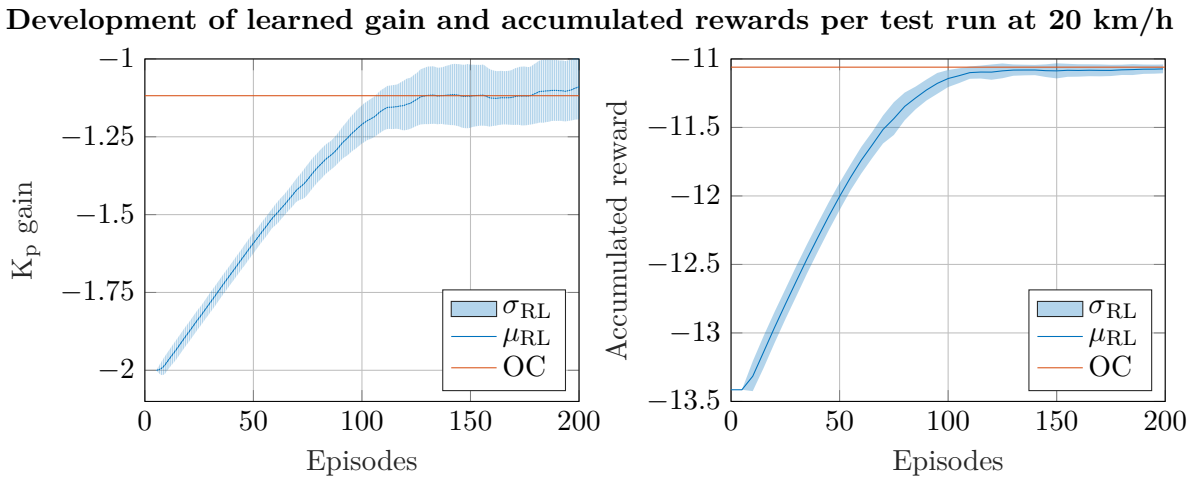
– Driving at 20 km/h on a road with 10% inclination.
  - Agent will be trained for driving in second gear and compared to OC set for driving in second gear on a flat road.

To measure the performance of the policy, a recurring test run is introduced as performance validation. After every umpteenth episode, the agent executes a 1 episode test run. In this run the agent starts from a specific initial state, without learning, and with the controller settings at that given time. Note that the test run does not necessarily have the same length as a normal training episode. It has to be long enough to let the system settle so all rewards can be accumulated, when the run is too short the test could indicate a better or worse performance than that is actually true. As this run is identical every time, it is a good way to measure the performance. The initial state, i.e. the offset $\delta_{velocity}$, from which the test run will start is chosen to be -3 km/h. The initial state can be chosen arbitrary from within the

probable operating region of the controller. Also an initial gain has to be given, this gain has to stabilise the system and in this case $K_p = -2$ is used, again, this can be chosen arbitrary.

4-2-1 Driving at 20 km/h on a flat road

RL and OC in same system settings The agent is trained to drive at 20 km/h in second gear, the optimal controller is also configured for driving at 20 km/h in second gear. From Figure 4-3b it can be inferred that the performance of the agent matches the optimal controller after training for about 110 episodes. This seems reasonable as the controller gains are approximately the same from this point on. The similarities in performance are to be expected since the optimal controller is derived from a the linearised system of the vehicle model in this specific set point. Exact values for the gains and rewards of RL and OC are shown in Table 4-3.



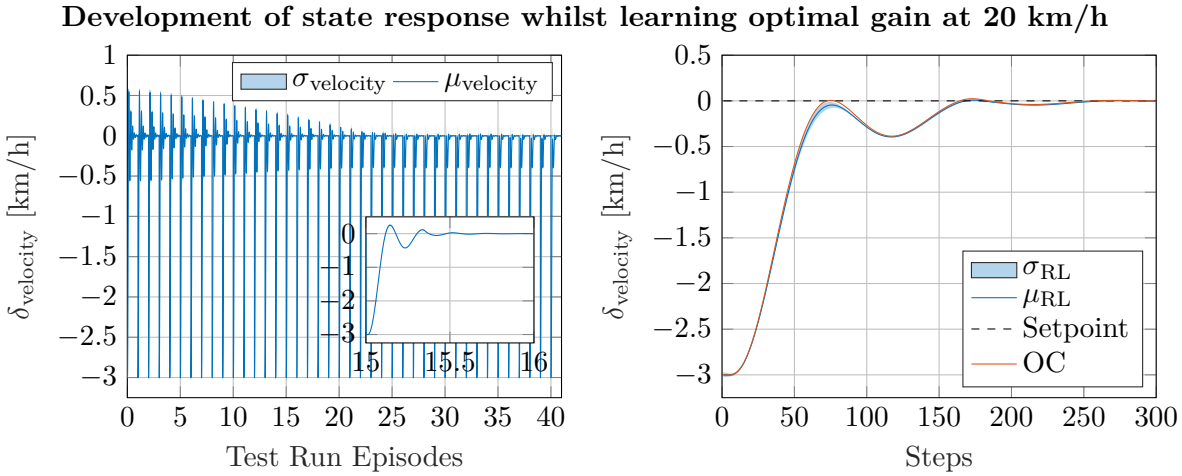
(a) Learned K_p-gain **(b)** Acculmulated Rewards per Validation Run

Figure 4-3: The development of the controller gain K_p and the accumulated reward in the test run are compared to their optimal output control counterpart. This learning simulation has run 20 times, each 200 episodes with 140 steps. The test during training runs every 5 episodes and is 500 steps long.

Table 4-3: Values of RL and OC at 20 km/h.

	K_p-gain	Accumulated Reward
RL	-1.1013	-11.07
OC	-1.1181	-11.06

Figure 4-4a depicts how the state response develops while the agent learns a more optimal controller gain. The behaviour of the state response is in line with what is to be expected. The agent's initial controller estimation is set at $K_p = -2$ and therefore implying a powerful state response with corresponding overshoot. As the agent learns a more appropriate gain the state response becomes more similar to the optimal control state response as can be observed in Figure 4-4b.



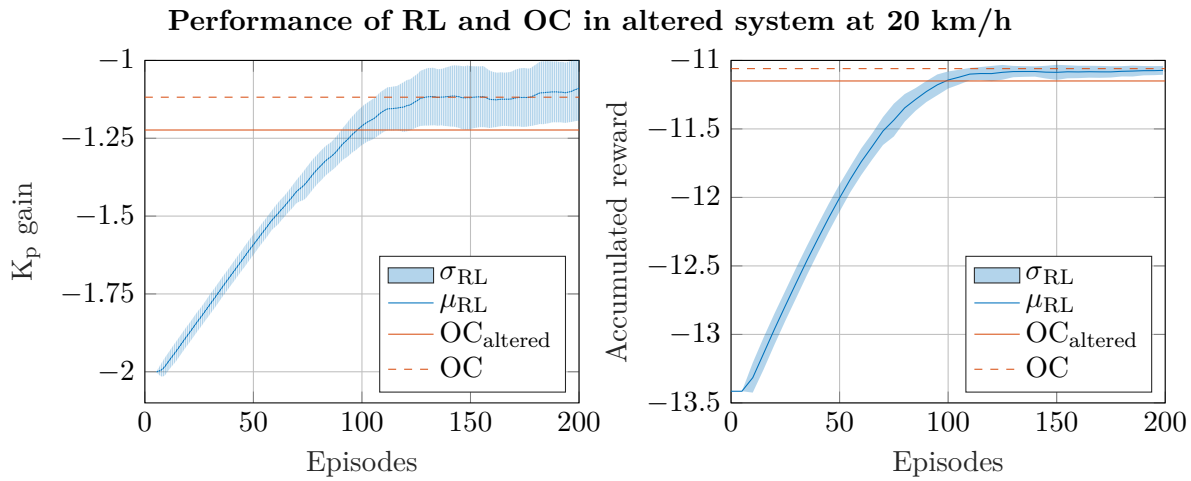
(a) Development of State Response

(b) State Response Final Test Run

Figure 4-4: The development of the state response while the agent is learning the optimal gain. The final test run is compared with the state response of the optimal control gain, they are almost identical. This similarity is expected as the optimal controller was derived for the linearised system of this environment. The test during training runs every 5 episodes and it is 500 steps long. Figure 4-4b is only plotted until 300 steps for clarity.

RL and OC in altered system settings To highlight the difference between non-model based learning and model-based control with poor model information the following situation is presented. The system does no longer align with the configuration for which the optimal controller is derived. In this case the car is still driving at 20 km/h in second gear, but the OC controller is determined for driving at 20 km/h in first gear. With this gear change the time constant τ changes accordingly as the engine makes more revolutions to maintain the same speed.

In Figure 4-5a the difference is shown between the learned RL gain, the OC gain for the corresponding system and the OC gain for the altered system. It can be observed that for the altered system, i.e. driving in first gear, OC suggests a more aggressive controller gain than it did for the driving in second gear. Figure 4-5a shows that for the altered system the RL agent also converges to a slightly higher value for the gain but proportionally is still much more conservative. As to be expected, in Figure 4-5b can than be seen that the performance of the optimal controller drops. Exact values for the gains and rewards of RL and OC are shown in Table 4-4.



(a) K_p-gains **(b)** Acculmulated Rewards per Test Run

Figure 4-5: Comparison between RL and OC for two cases. OC derived for the proper system (dashed) and OC derived for an altered system (solid) where the gear is changed. The poor model information leads to a higher gain and worse performance. This learning simulation has run 20 times, each 150 episodes with 140 steps. The test during training runs every 5 episodes and it is 500 steps long.

Table 4-4: Values of RL, OC and $OC_{altered}$ at 20 km/h.

	K_p-gain	Accumulated Reward	Time constant τ
RL	-1.1013	-11.07	n.a.
OC	-1.1181	-11.06	0.910
$OC_{altered}$	-1.2235	-11.15	0.186

When looking at the state response of the two controllers, Figure 4-6, the consequence of the higher gain of the altered OC can be observed. The optimal controller is now more powerful and will thus have overshoot where as the RL controller and the previous optimal controller do not.

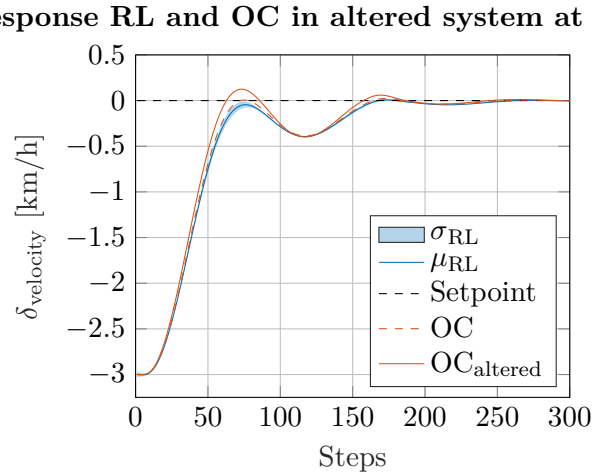
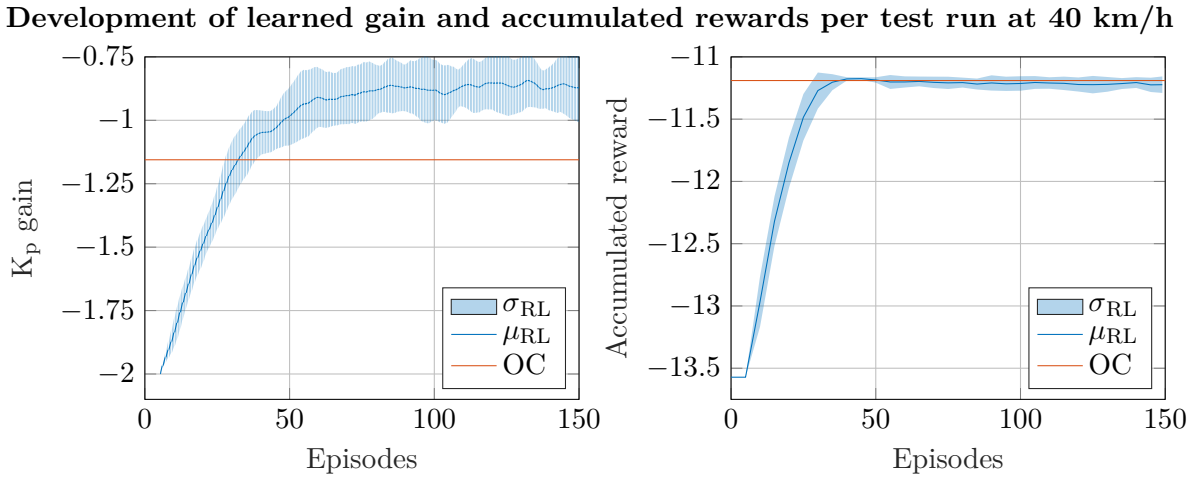


Figure 4-6: Comparison between RL and OC for two cases. OC derived for the proper system (dashed) and OC derived for an altered system (solid) where the gear is changed. The poor model information leads to a higher gain which results in overshoot of the state response. The test during training runs every 5 episodes and it is 500 steps long. The figure is only plotted until 300 steps for clarity.

4-2-2 Driving at 40 km/h on a flat road

RL and OC in same system settings The agent is trained to drive at 40 km/h in third gear, the optimal controller is also configured for driving at 40 km/h in third gear. As from Figure 4-7b can be inferred is that the performance of the RL and OC controller are again almost the same. As opposed to driving at 20 km/h in this case the agent takes on a more conservative gain than OC suggests. Noticeable is also that the agent converges to its optimal gain much earlier than in the 20 km/h case. In the previous case both gain and accumulated reward settled at around 110 episodes. Here after approximately 60 the optimal gain is reached whereas after 40 episodes the accumulated reward is on par with OC.

Observing the development of the state response in Figure 4-8a, the state response has a lot of overshoot with an initial gain of -2. As the gain of the agent converges the state response gradually behaves like a first order system. Comparing then the state responses in Figure 4-8b it can be seen that the OC controller has some overshoot, this is due to its higher controller gain whereas RL is more conservative and thus has a smoother state response.

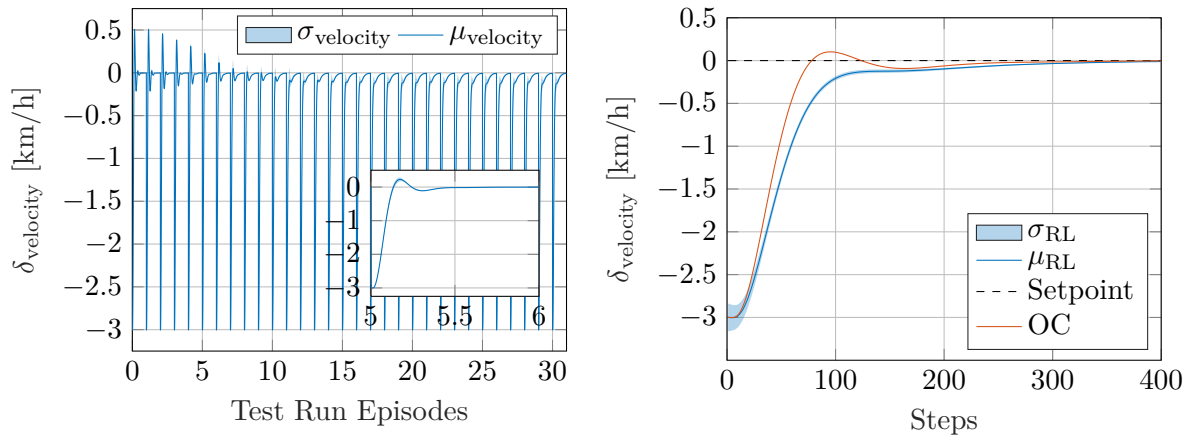


(a) Learned K_p-gain **(b)** Acculmulated Rewards per Test Run

Figure 4-7: Development of the controller gain K_p and the accumulated reward in the test run compared with OC. Noticeable is that RL converges to a more conservative gain than OC while their performance match. This learning simulation has run 20 times, each 150 episodes with 140 steps. The test during training runs every 5 episodes and it is 500 steps long.

Table 4-5: Values of RL and OC at 40 km/h.

	K_p-gain	Accumulated Reward
RL	-0.8773	-11.21
OC	-1.1556	-11.19

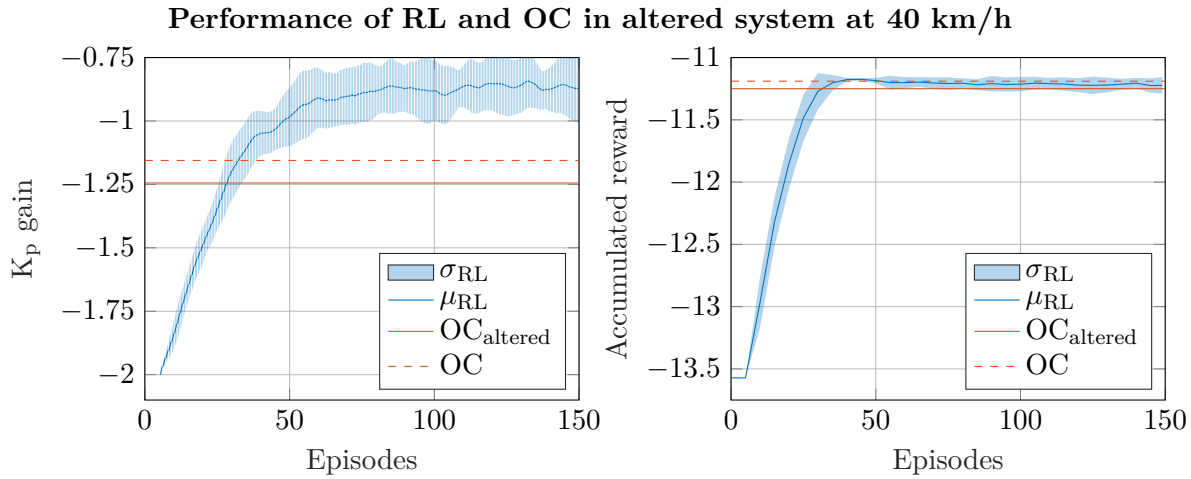


(a) Development of State Response **(b)** State Response Final Test Run

Figure 4-8: Development of the state response while the agent is learning the optimal gain. Because of the more conservative gain of RL its state response is damped and does not has overshoot like OC. The final test run is compared with the state response of the optimal control gain. The test during training runs every 5 episodes and it is 500 steps long. Figure 4-8b is only plotted until 400 steps for clarity.

RL and OC in altered system settings Again to compare the difference between non-model based learning and poor model-based control, a situation is presented in which the system configuration of the OC controller are no longer accurate. The agent is still trained for driving at 40 km/h in third gear, whereas the OC controller is derived for driving at 40 km/h in second gear.

When observing Figure 4-9a the same behaviour of the OC controller towards the RL agent is seen. As the gear is changed, the time constant τ increases and leads to a higher gain for OC. The accumulated rewards are less but still do not differ that much from the RL agent, see Figure 4-7b. The corresponding values for controller gain, accumulated reward and time constant are shown in Table 4-6.



(a) K_p-gains **(b)** Acculmulated Rewards per Test Run

Figure 4-9: Comparison between RL and OC for two cases. OC derived for the proper system (dashed) and OC derived for an altered system (solid) where the gear is changed. The false system information leads to an even higher gain for optimal control and a corresponding worse performance. This learning simulation has run 20 times, each 150 episodes with 140 steps. The validation during training runs every 5 episodes and it is 500 steps long.

Table 4-6: Values of RL, OC and $OC_{altered}$ at 40 km/h.

	K_p-gain	Accumulated Reward	Time constant τ
RL	-0.8773	-11.21	n.a.
OC	-1.1556	-11.19	0.632
$OC_{altered}$	-1.2445	-11.25	0.600

When looking at Figure 4-10 the same behaviour as seen in the 20 km/h case can be observed. Because of the change in the system, the OC controller is now more powerful and thus will have even more overshoot.

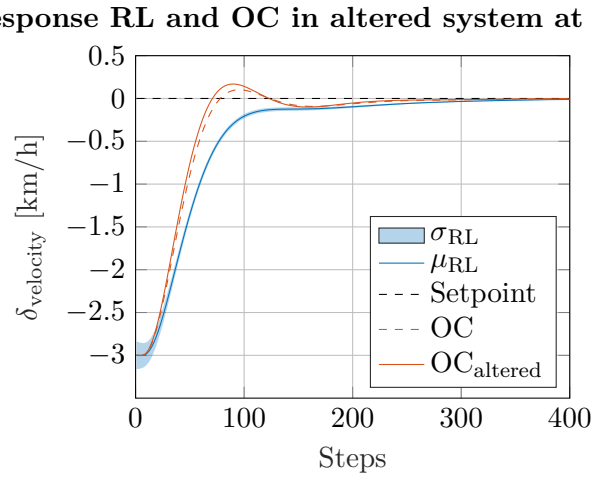


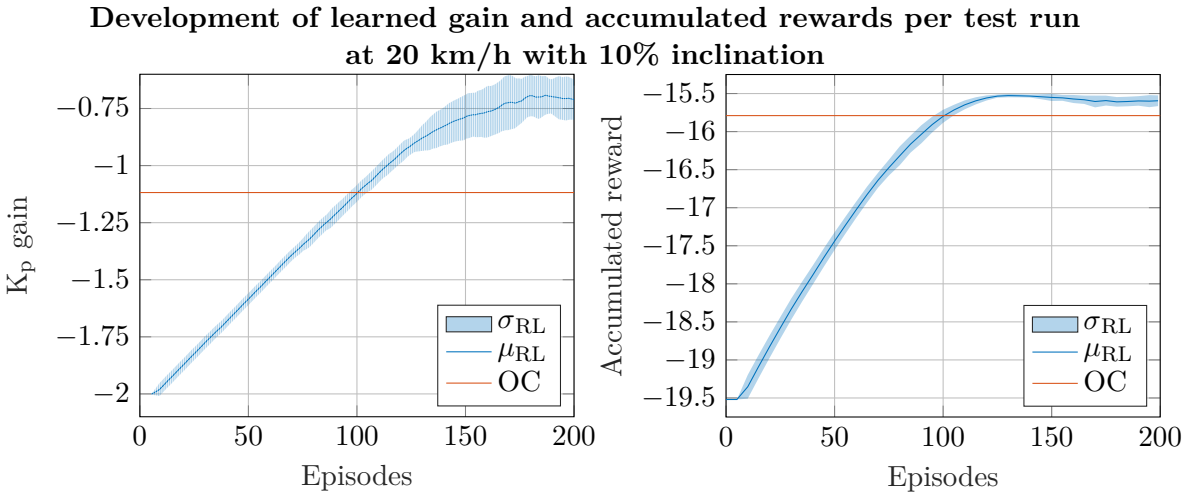
Figure 4-10: Comparison between the RL and OC for two cases. OC derived for the proper system (dashed) and OC derived for an altered system (solid) where the gear is changed. The poor model information leads to a higher gain and therefore results in even more overshoot. The test during training runs every 5 episodes and it is 500 steps long. The figure is only plotted until 400 steps for clarity.

4-2-3 Driving at 20 km/h on a road with 10% inclination

A scenario is presented in which the change of the system is more extreme than only a gear change. In this scenario the car is driving at 20 km/h uphill with a 10% inclination, i.e. 5.71°. Where the RL agent can train on the model with this system setting the linear model has no way of adjusting to a road inclination, as τ does not change, the OC controller is the same as driving on a flat road at 20 km/h.

As the environment changes to include the inclination, a big difference can be observed. From Figure 4-11a can be derived that the RL controller converges to a much lower gain than it did when the car was driving on a flat road. Noticeable is also that it takes longer for the agent to converge to this gain, approximately 175 episodes. The lower gain then also yields a better result, i.e. accumulated reward, as can be seen in Figure 4-11b. Exact values of gain and accumulated reward are shown in Table 4-7.

When looking at the state response of the two controllers it can be seen that the RL controller, with its much lower gain, yields a state response that is damped more than the optimal controller. The OC controller being more forceful rises faster and converges earlier but still shows a small oscillation before it settles.



(a) Learned K_p-gain **(b)** Acculmulated Rewards per Validation Run

Figure 4-11: Development of the controller gain K_p and the accumulated reward in the test compared with OC. Because the optimal controller cannot account for the road inclination its gain and performance differ from RL. This learning simulation has run 10 times, each 150 episodes with 140 steps. The test during training runs every 5 episodes and it is 500 steps long.

Table 4-7: Values of RL and OC at 20 km/h with 10% inclination

	K_p-gain	Accumulated Reward
RL	-0.6989	-15.59
OC	-1.1181	-15.79

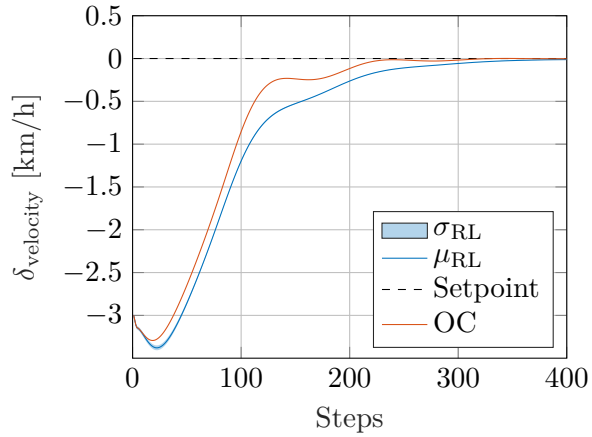


Figure 4-12: Comparison between the RL and OC state response for when the optimal controller is determined for an altered system. As the controllers do not compare the state responses are quite different. The RL controller is conservative and yields a damped state response. The gain of OC is higher, the state response rises faster but settles with a small oscillation. The test during training runs every 5 episodes and it is 500 steps long. The figure is only plotted until 400 steps for clarity.

4-3 Discussion

To show that the performance of a learned controller can match optimal control, a comparison is made where both controllers are simulated for the same speed and gear. In both cases their performance is comparable. The similarity in performance is not unexpected as a linearised model of the nonlinear high fidelity model is used to determine the optimal controller. This controller is then used in the comparison with the vehicle model set to the same specifications as the linearised model. Noteworthy is that for driving at 20 km/h RL converges to almost the exact value as OC and therefore their performance is the same. But for driving at 40 km/h same performance for RL as OC was reached with two relatively dispersed gains, $K_{p,RL} = -0.8773$ and $K_{p,OC} = -1.1556$. It seems that the RL agent finds a different optimal path to stabilise the system with less control action than OC. A reason for this can be the way how RL and OC both handle the dependency of the initial state on the control problem. As mentioned in section 2-1, OC makes an assumption that distributes a set of initial states on the unit circle and then averages the performance over them. So the emphasis lies completely on the optimal gains corresponding that specific region (the unit circle). On the other hand the RL agent is relatively free to explore the forthcoming performance from various initial states as it can sample an arbitrary number of initial states from any region. The region from which RL samples its initial states can be chosen arbitrary. As it is not bounded, the exploration is much richer and may very well lead to a control path matching the performance of optimal control, but requires less control action. Also interesting is that RL is able to converge to an optimal gain much faster while driving at 40 km/h than driving at 20 km/h. A reason for this could be that at higher speeds the low-level controller exerts less influence on the system and the system becomes easier to control for RL. Also, what could help the agent at higher speed is the fact that the speed deviations that can occur are relatively smaller. Looking at the state response it can also be observed that in the 20 km/h scenario the RL and OC align. Noteworthy is that the response resembles a second order system whereas when properly controlled, the response of a first order system is expected. The main reason for this is the influence of the low-level controller in the system. When looking at the 40 km/h case the response is much more like a first order system, as expected.

Secondly a comparison is made between OC and RL where the optimal controller is designed with poor model information. A situation is presented in which the vehicle is driving again at 20 and at 40 km/h but in both cases the OC controller was set to drive in a different gear as RL. This simulation defines the actual use case for RL in autonomous driving controller design. OC is a model-based controller design method, meaning that when derived it is only optimal for one specific model setting. When this changes, the controller performs suboptimal. This is where RL takes advantage of the fact that it is non-model based. The RL agent can adapt when the system changes, it is not bound to a specific setting of the model. In both cases when comparing RL with OC in a changed model environment, it can be clearly seen that where RL is able to change and maintain its performance level whereas OC can not and performs worse. When observing the state responses of the altered OC controllers with the RL controller, the effect of more overshoot is to be expected with a stronger controller.

Last a comparison was made between RL and OC for a situation in which the car is driving uphill on a road with a 10% inclination. This comparison illustrates an extreme situation change and emphasises the important of accurate system knowledge in the case of OC even more. As accurate system knowledge is not always available the advantages of learning in a

model-free fashion becomes evident. RL is able to adapt to the system in any situation and in this comparison it can then be clearly seen that it converges to a better controller gain. This controller then yields a better performance and in the validation run the state settles with less osculation.

Chapter 5

Experimental Comparison

The goal is to let the Reinforcement Learning (RL) agent learn and adapt controller settings in real time as the car drives. In the current state the RL algorithm is not robust or fast enough to do this. To verify its learning results from the simulations in the previous chapter an experiment to compare the controllers in a real world setting is set up. As the agent can not learn in the actual vehicle[1], the learned gains are set as fixed controllers and their state responses will be evaluated. First the comparison will be described whereafter the results will be discussed.

5-1 Experimental Preliminaries

In the experiment the objective is to compare the learned RL controllers from the simulations, against its OC counterparts and a controller set by BMW. At BMW there are many different models which are being used, for the given case as benchmark "BMW-controller" a gain of $K_p = -0.7$ was advised for low speeds. The corresponding controller gains are again shown in Table 5-1 and Table 5-2. The following scenarios are tested.

– Driving at 20 km/h on a flat road in second gear.

- Using RL controller trained for driving at 20 km/h in second gear.
- Using OC controller derived for driving at 20 km/h in second gear.
- Using OC controller derived for driving at 20 km/h in first gear.
- Using advised BMW controller.

[1] A BMW 7 series, as of which also the nonlinear model is derived.

– Driving at 40 km/h on a flat road.
  - Using RL controller trained for driving at 40 km/h in third gear.
  - Using OC controller derived for driving at 40 km/h in third gear.
  - Using OC controller derived for driving at 40 km/h in second gear.
  - Using advised BMW controller.

Table 5-1: Corresponding gains for driving 20 km/h in second gear.

	K_p**-gain**
BMW	-0.7
RL	-1.1013
OC	-1.1181
$\mathbf{OC}_{altered}$	-1.2235

Table 5-2: Corresponding gains for driving 40 km/h in third gear.

	K_p**-gain**
BMW	-0.7
RL	-0.8773
OC	-1.1556
$\mathbf{OC}_{altered}$	-1.2445

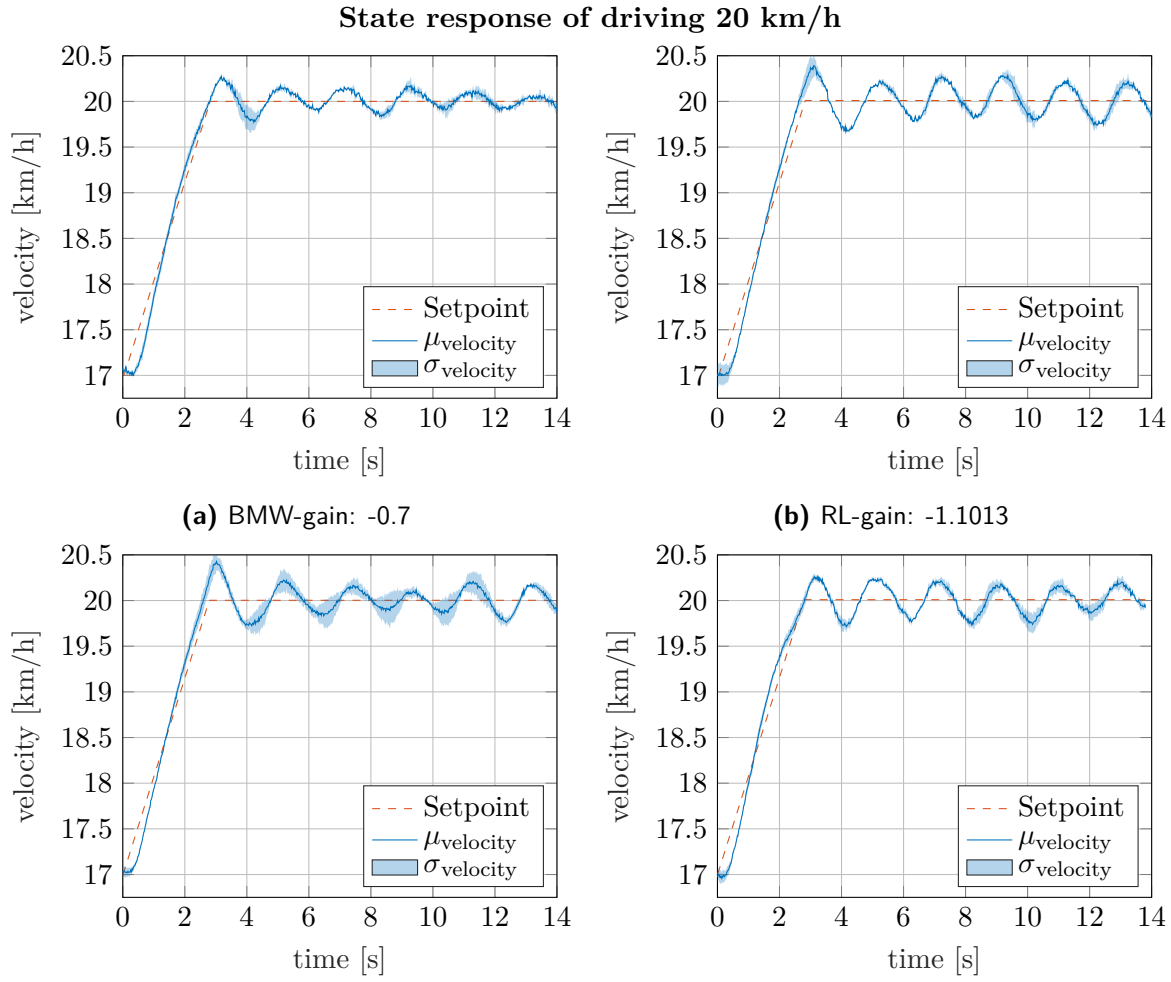
For the test run in the simulation study an offset of -3 km/h is used to show the state response's rate of stabilisation. To mimic this scenario in the experiment, the car was driven at a constant speed of 17 km/h and 37 km/h whereafter the speed input is increased to 20 km/h and 40 km/h respectively.

5-2 Results

In the simulation study the offset of 3 km/h can be seen as a step input on time step t_0. Because of how the software is configured in the car due to safety this can not be recreated. Therefore when the car is driven at 17 km/h and the input to the controller is changed to 20 km/h, a ramp-input is passed to the controller.

5-2-1 Driving at 20 km/h

Taking into consideration that the gains for RL and OC in the same setting are very close to one another it is to be expected that their behaviours match (Figure 5-1b and 5-1d). What is interesting to see when comparing the two optimal controllers (Figure 5-1c and 5-1d), the controller determined for the altered system does have more overshoot than the response of the controller for the correct system. Still, comparing these responses to the response of the BMW-gain in Figure 5-1a it can inferred that the BMW-gain is the only one able to dampen out the oscillations. A gain in the region of 1.1-1.2 is apparently too high and keeps exciting the system, only a more conservative gain is able to decrease the oscillation.



(a) BMW-gain: -0.7

(b) RL-gain: -1.1013

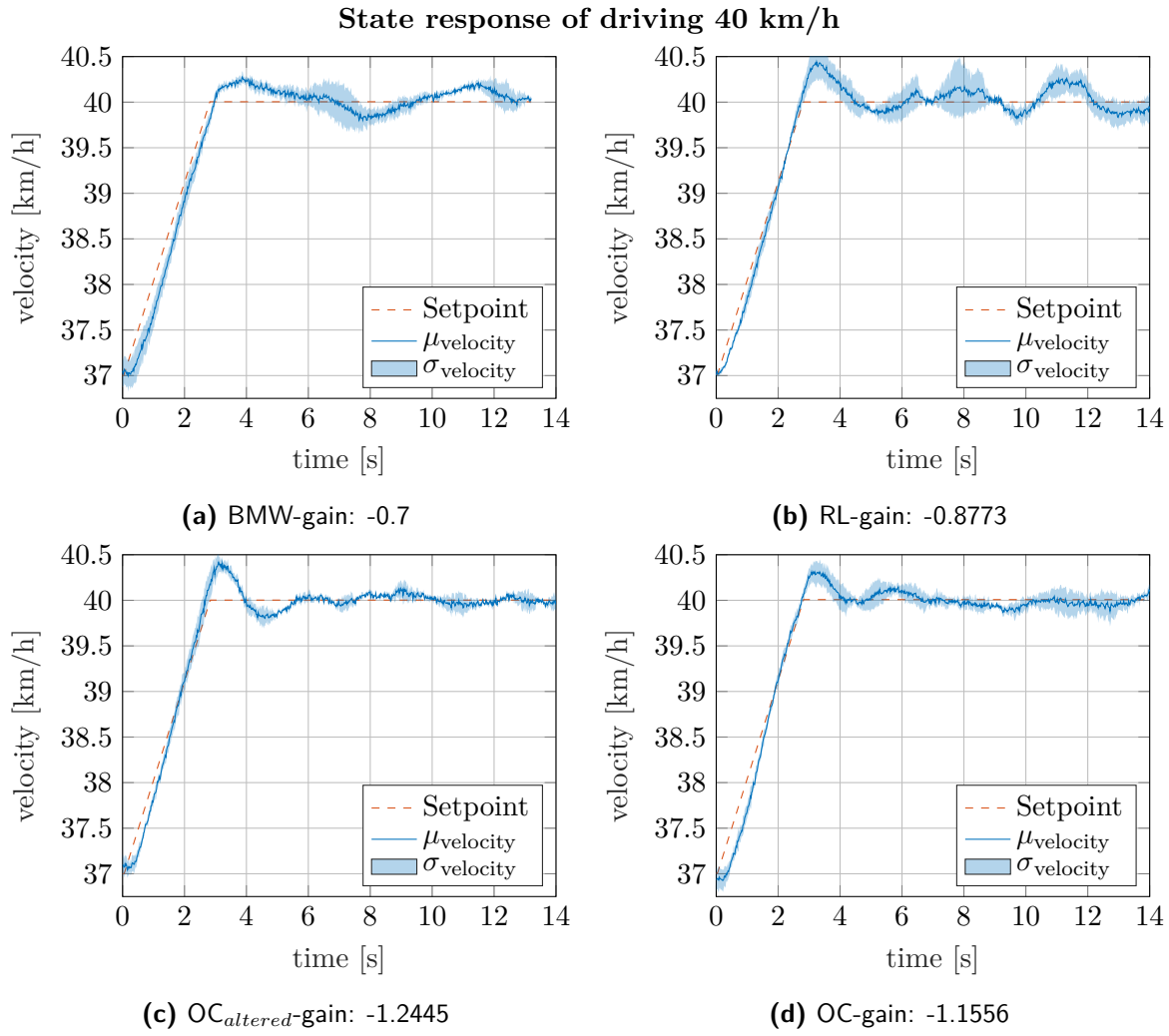
(c) $\text{OC}_{altered}$-gain: -1.2235

(d) OC-gain: -1.1181

Figure 5-1: The various state responses plotted for driving from 17 km/h to 20 km/h, each experiment was performed three times.

5-2-2 Driving at 40 km/h

Driving at 40 km/h with the proposed gains yields a opposite response to what could be observed in the 20 km/h experiment. The more conservative BMW and RL controllers are apparently too weak to decrease the oscillations, as can be inferred from Figure 5-2a and Figure 5-2b. The controllers derived via optimal control have higher gains and seem therefore be able to damp the oscillations. Also here, as in the 20 km/h case, the overshoot the OC controller for the proper gear (third) yields is less than the overshoot the controller for the altered system, i.e. second gear. The altered OC gain that is proposed is higher than the gain for OC in the proper system, therefore it stabilises the velocity better. It seems that a higher controller gain is desired at higher speed, this is at odds with the derived conclusion from the simulation study.



(a) BMW-gain: -0.7

(b) RL-gain: -0.8773

(c) $\text{OC}_{altered}$-gain: -1.2445

(d) OC-gain: -1.1556

Figure 5-2: The various state responses plotted for driving from 37 km/h to 40 km/h, each experiment was performed three times.

5-3 Discussion

The experimental data propose interesting behaviour of the controllers when used in a real world setting. When driving at 20 km/h the RL controller as well as the OC controllers are not able to decrease the velocity's oscillation where as the BMW controller is. The RL and OC gains are in a region of 1.1-1.2, this compared to the BMW controller of 0.7 is too high. On the contrary when evaluating the BMW controller at 40 km/h, it is oscillating. In the given experiment for both speeds a gain of 0.7 was used. Given the fact that for various system settings different controllers are optimal, proper gain scheduling is required for optimal performance. In hindsight for driving 40 km/h a different (higher) gain would have been more appropriate. This can be inferred from the OC gains in both situations, as they are able to decrease the oscillation. The importance of gain scheduling is also emphasized by the difference that can be observed between the optimal controllers. At both speeds the optimal controller derived for an altered system yields higher overshoot and thus performs worse. As RL is able to adjust it is more flexible towards these changes.

The difficulty in comparison lies also in the difference between the model used for training and the actual vehicle. The nonlinear model used for training is not an accurate model of the vehicle that is used for the experiment. Also, both model and vehicle have a low-level controller. This controller tries to linearise certain effects exercised on the system. These effects can vary from drive-train, like gears or throttle demand, to external settings such as changes in road surface or drag. Different settings of this low-level controller can drive apart the output of the whole model drastic. Besides, in this experiment a learned and optimal controller are compared with a manually tuned controller. Performance criterion for both controller design methods may not even include penalising oscillations as a control engineer would.

Chapter 6

Conclusion

The bigger picture behind this work is to make a comparison for controller design in autonomous vehicles. A comparison between a classical control theory method and a novel machine learning method is proposed. The idea is to emphasize the advantage of learning over theory. For this the focus lies with longitudinal control for autonomous vehicles. More specific, with speed control using Optimal Control (OC), i.e. classic control theory, and Reinforcement Learning (RL), i.e. machine learning.

The goal for final application is an autonomous vehicle. The design process started with a linearised model of a vehicle as described in chapter 3. With this linearised vehicle model the initial structure of the RL agent was derived. Also the linearised model is used to determine the OC gains further on. As function approximator in the critic a nonlinear neural network structure was compared with a quadratic function approximator. The quadratic approximation yields much lower error rate of the approximated function and is therefore chosen to be used henceforth. Further also an approach to reconstruct unobservable system dynamics is applied. Using past actions with a fully connected layer as finite impulse response filter it is possible to reconstruct an unobserved state, the acceleration.

In chapter 4 simulations using a nonlinear high fidelity vehicle model are presented. These simulations underline the difference between classic controller design and learning a controller. The simulations were conducted at two different speeds, 20 km/h and 40 km/h. For initial comparison the performance of both the OC controller and the RL agent was on par. This equal performance was to be expected as the OC controller is designed following a linearised version of the the model in the same exact system settings. In a situation where the system settings in the model change, learning makes the difference. This difference can be explained by the fact that classical control algorithms, as OC, rely on observable model information. When proper model information is available an optimal controller can be designed, but as in many situations this information is not completely or properly available and therefore the controller can perform suboptimal. Learning methods like RL can operate without any model information. To compare this change of system with respect to the performance of the controllers various scenarios were presented: Driving at 20 km/h and at 40 km/h whereafter the OC parameters were implemented for a different gear than in which the system was

actually operating. A gear change is chosen as the comparable modification in the model. This is done because the change in gear can be represented through an adjustment of the time constant that is part of the system matrices of the linearised system. The time constant representing the behaviour of the drive-train. Also a case was presented in which the system was running in an altered setting, i.e. driving on a road with 10% inclination. From these simulations it can be clearly observed performance of RL and OC is comparable when used in the same system settings. But, in all cases where system changes occur RL outperforms its OC counterpart.

To validate the obtained results from the simulation study an experimental comparison is conducted in chapter 5. Although it is difficult to draw a proper conclusion from data between a learning metric and any other form of control, when the learning algorithm is not able to train on that specific system, some conclusions can be drawn. From the simulation study and from the 20 km/h experiment it can be concluded that the performance of RL and OC derived with proper model information are comparable. When comparing the optimal controllers derived for different system settings in the experiment, it is clear that proper gain scheduling is important for optimal performance. OC performs worse, i.e. has more overshoot when derived for an altered system. As RL is able to account for system changes it can always adapt its controller settings to perform on par with an optimal controller derived for the proper system settings.

If then the question that was formulated in the beginning of this work is considered again:

"Can Reinforcement Learning be beneficial for longitudinal control of autonomous vehicles?"

It can be concluded that RL can be beneficial for longitudinal control of autonomous vehicles. It benefits from the fact that methods like OC rely on proper model information, while learning can be done without system knowledge. As gain schedules that are depending on a lot of model parameters can become complicated in higher orders RL has the potential to learn controller settings in every situation independent of how many model parameters are involved.

6-1 Future Work

The application of learning methods such as RL in longitudinal control have the potential to improve on many levels, some remarks on future work are:

As stated above and in chapter 5 learning in the system where the algorithm should be operational is key to a proper learning process. Therefore to be able to apply RL methods in actual vehicles the agent has to be able to learn in an actual vehicle. Of course performing this with random initialisation is not possible as it is prone to breaking parts of the system or potentially being hazardous. But, applying a form of agent that is pre-trained in a simulation model and letting that agent learn in the car could yield promising results.

One of the main difficulties one faces trying to properly implement RL in any system is tuning the hyperparameters of the agent. Looking at a way to determine, or learn, the most optimal hyperparameter settings and letting the actor and critic be operational to their full potential is something would benefit the complete optimisation scheme.

An important part of the optimisation process of the RL agent is the reward function. The reward function determines if what the agent does is "good". Therefore, a proper reward function that is specifically designed for a given problem could be a major contribution.

Additionally to the reward function, the determination of the value function is also key to the total operation of the RL algorithm. In this work a quadratic function approximation was used as it was already known that the value function would take on a quadratic form. Of course as problems become more difficult it would be best to use a more general function approximation for example in the form of a neural network with non linear activation and even deep neural networks. As this work also focussed on obtaining a linear controller from the RL algorithm the "network" in the actor did only consist of one weight, i.e. the linear controller gain. But as the functions to be solved become more complicated, more extensive networks or deep networks could also be implemented in the critic and actor. The latter would allow for design of a nonlinear controller.

Appendix A

Optimal Control

A-1 Regular Optimal Control

On basis of full-state feedback one would minimise the cost function J [3, 4, 24], given by

$$J = \sum_{k=0}^{\infty} x_k^T Q x_k + u_k^T R u_k. \tag{A-1}$$

By solving the Algebraic Riccati Equation

$$P = A^T P A - (A^T P B)(R + B^T P B)^{-1}(B^T P A) + Q, \tag{A-2}$$

and from this P can be determined and consecutively the optimal controller could be calculated following

$$K = (B^T P B + R)^{-1} B^T P A. \tag{A-3}$$

Obtaining control law

$$u_k = -K x_k \tag{A-4}$$

A-2 Output Feedback Control

For output feedback the control law will take on a different form

$$u_k = -K y_k \tag{A-5}$$

or

$$u_k = -KCx_k. \tag{A-6}$$

On this basis Equation 2-1 can be rewritten as

$$x_{k+1} = (A - BKC)x_k, \tag{A-7}$$

where

$$x_k = (A - BKC)^k x_0. \tag{A-8}$$

From this we infer a cost function in the following form

$$\begin{aligned} J &= x_0^T \Big[\sum_{k=0}^{\infty} (A - BKC)^{k^T} (Q + C^T K^T RKC)(A - BKC)^k \Big] x_0 \\ J &= x_0^T P x_0. \end{aligned} \tag{A-9}$$

This can be rewritten into

$$x_0^T P x_0 = trace(P x_0 x_0^T), \tag{A-10}$$

and thus

$$J = trace(P x_0 x_0^T). \tag{A-11}$$

Because the initial states are evenly distributed on the surface of the n-dimensional unit sphere the following holds

$$\mathbb{E}\{x_0 x_0^T\} = \tfrac{1}{n} I. \tag{A-12}$$

And the average performance becomes

$$\begin{aligned} \hat{J} &= trace(P) \\ \hat{J} &= trace\Big(\sum_{k=0}^{\infty} (A - BKC)^{k^T} (Q + C^T K^T RKC)(A - BKC)^k \Big). \end{aligned} \tag{A-13}$$

The optimal output controller can then be determined via an iterative solution, e.g. by following Algorithm 1.[25, 26]

Algorithm 1 Iterating optimal output feedback controller (Matlab)
Given:

System matrices $\rightarrow (A, B, C)$
Weighing matrices $\rightarrow Q$ and R
Initial parameter, K_y^0, that stabilises system

1: $\bar{A} = A + BFC$
2: $\bar{Q} = Q + C^T F^T RFC$
3: $P = dlyap(\bar{A}, \bar{Q})$
4: $\hat{J} = @(F)(trace(P))$
5: $K_y = fminsearch(@(F)(\hat{J}(F)), K_y^0)$

Appendix B

Reinforcement Learning Solving Methods

Before discussing various learning methods in depth some fundamental RL principles have to be explained.

Exploration & Exploitation RL problems usually do not feature complete model information (Model-free RL) and therefore, as stated before, makes use "trial-and-error" to interact with the environment to be able to learn optimal value functions or policies. At first, the agent will perform random actions to explore the environment. After some time it will have obtained some policy with a particular performance. A dilemma presents itself, the agent can follow this policy, being certain of obtaining a specific amount of reward from the policy, i.e. exploiting the information that is has gathered. Or, in order to find out if there is an improvement to make, the agent will have to try different actions. These different actions may lead to a deterioration of the performance, but could also lead to improvement i.e. it has to explore to learn. Finding a proper balance is key in the RL exploration-exploitation dilemma.[8, 27]

On-policy & Off-policy When the agent learns the optimal policy and simultaneously executes the actions it describes it is called on-policy learning. In RL a distinction can be made between on-policy and off-policy learning. Off-policy methods follow a certain policy without exploration and learn the optimal policy independently of the agent's actions.[8, 27]

Online & Offline Learning Letting an agent learn and control a real-world system through, initially, random interaction with the environment is not a safe approach. This is why a distinction can be made between online and offline reinforcement learning. In an online learning situation the agent directly learns by interaction with the environment.[8, 27] For example, a car needs to find an optimal path to a parking spot, it is not desirable to let it drive and explore the environment because there will be a fair chance of harming others or itself. This is where offline learning comes in, the environment is simulated and here the

agent will learn how to control the car. Online and offline learning are not necessarily two different ways of letting the agent learn, a combination could work very well. A simulation is never an exact copy of the real-world and therefore after the agent has done a lot of training in simulation it can be fined-tuned operating in the real-world.

B-1 Value Function Methods

Value function based methods rest on the idea of valuing individual state or state-action pairs. Maximising the accumulated values of these states or state-action pairs is then the goal, i.e. finding the value function for optimal policy π^*, see Equation 2-15 for the state-action-value function. Indirectly through the value function, by looking at which action yields the highest value, the optimal policy is derived.[50, 51] To estimate an optimal value function or state-action-value function a variety of algorithms have been developed, the three main types will be further discussed.

B-1-1 Monte Carlo

Monte Carlo (MC) methods are model-free online methods that uses samples to estimate average sample returns. At the end of each episode the agent evaluates all returned values for state $s_t \in \mathcal{S}$ and averages them. Long-term rewards are treated like random variables by the MC algorithms, therefore it uses the sampled means as an estimate of the values.[8, 27, 51] A representation of this is given by Algorithm 2.

Algorithm 2 Monte Carlo Value Function Algorithm
Initialisation:
$T \leftarrow$ terminal state
$\pi \leftarrow$ policy to be evaluated
$V \leftarrow$ arbitrary value function
$\gamma \leftarrow$ discount factor
$\alpha \leftarrow$ step size parameter

1: **for** $t \leftarrow 0 : (T-1)$ **do**
2: Generate episode using policy π
3: **for** each s_t in the episode **do**
4: $r_t \leftarrow$ reward following s_t
5: $R_t \leftarrow \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$
6: $V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)]$
7: **end for**
8: **end for**

B-1-2 Dynamic Programming

Dynamic Programming (DP) is a model-based class and refers to a set of algorithms that is able to compute the value function and optimal policy.[8, 27, 30, 50] These methods are called value and policy iteration respectively. The transition probability matrix $\mathcal{P}^a_{ss'}$ and reward

function $\mathcal{R}^a_{ss'}$ are known and because of this, the model does not have to be predetermined as it can be learnt from the data.[51] Value iteration is a form of DP that focuses on the direct estimation of the value function, the policy evaluation is broken off after one iteration and blended into the value function update. This update rule consists of a combination of the policy improvement and truncated evaluation steps

$$\begin{aligned} V_{k+1}(s) &= \max_a \mathbb{E}\{r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s, a_t = a\} \\ &= \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_k(s')]. \end{aligned} \tag{B-1}$$

B-1-3 Temporal Difference

Temporal Difference (TD) Learning is a solving method that combines properties of both the Monte Carlo and Dynamic Programming algorithms. TD methods are like MC methods model-free and like DP do not have to wait until the end of an episode to be able to update the value function, they can do it after every time step.[51, 52] TD methods make use of the temporal error, i.e. the difference between the estimated value of the old and new state. In this difference the received reward for the current state has also be taken into account, Equation B-3. This error is used to learn the value estimates, in other words, TD methods learn new values based on previous approximated values. This is called bootstrapping.[27]

TD(0) & TD(λ)

TD learning bares similarities with MC methods, but instead of having to wait a full episode, it is able to update after each time step. When the TD algorithm updates, observing the information after one time step, this is called TD(0), the most simple TD method.[52, 53] The TD(0) value function update rule is given by

$$V(s_t) = V(s_t) + \alpha[r_t + \gamma V(s_{t+1}) - V(s_t)], \tag{B-2}$$

where $r_{t+1} + \gamma V(s_{t+1})$ is called the TD target. When looking at line 6 of Algorithm 2 the similarity can be seen, in effect the MC update is nothing more than TD(0) with return R_t as its target, i.e. the full episode is used for approximation. Furthermore, from Equation B-2 the TD error can be derived as

$$\delta_{TD} = r_t + \gamma V(s_{t+1}) - V(s_t). \tag{B-3}$$

A way of to unifying the MC and TD(0) methods is by implementing TD(λ) where $\lambda \in [0, 1]$.[53] Here, n-step prediction and parameter λ allow interpolation between TD(0) and MC. The n-step prediction states how many steps are taken to perform evaluation, $n = 1$ equals $\lambda(0)$ where as $n = \infty$ represents MC. Performing a n-step prediction λ is used in a way to average the n-step backups following, λ^{n-1}. The resulting averaged backup towards the return is called the λ-return, R^λ_t. How this return is calculated and interacts with TD(λ) algorithm can be interpreted as in Algorithm 3.

Algorithm 3 TD(λ) Algorithm
Initialisation:
$\pi \leftarrow$ policy to be evaluated
$V \leftarrow$ arbitrary value function
$\gamma \leftarrow$ discount factor
$n \leftarrow$ n-step prediction
$\lambda \leftarrow$ mixing coefficient weight
$\alpha \leftarrow$ step size parameter

1: **for** each time step t of episode **do**
2: $a_t \leftarrow$ action given by policy π for s_t
3: $r_t \leftarrow$ reward following (s_t, a_t)
4: $R_t^n \leftarrow \sum_{k=1}^{n} \gamma^{k-1} r_{t+k} + \gamma^k V(s_{t+k})$
5: $R_t^\lambda \leftarrow (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^n$
6: $V(s_t) \leftarrow V(s_t) + \alpha[R_t^\lambda - V(s_t)]$
7: **end for**

Q-Learning

Maybe the most widely used algorithm for model-free value function learning is Q-learning.[54] It is an off-policy method that uses experienced rewards for an estimation of the agent's Q-value function to incrementally estimates Q-values, i.e. the state-action pairs. The update rule of Q-learning

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha[r_t + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)] \tag{B-4}$$

bears a resemblance to Equation B-2 as the part between brackets is again the TD error. Therefore, like in DP and TD(λ) the Q-learning algorithm is able to locally update a Q-value on the basis of the next Q-value.

SARSA

SARSA (for State-Action-Reward-State-Action, i.e. $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$) is the on-policy version of Q-learning. Meaning that SARSA, while following an initially given exploration policy π, will try to estimate a policy π.[55] The update rule for SARSA is given by

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha[r_t + \gamma Q_t(s_{t+1}, \pi(s_{t+1})) - Q_t(s_t, a_t)]. \tag{B-5}$$

B-2 Policy Methods

Value function methods look for a way of approximate an optimal value function given any policy. Policy methods look at the RL problem from a control aspect as they focus on approximating an optimal policy.[56] Both MC and DP methods also have a policy based approach counterpart which, as well as the most known policy gradient methods, will be discussed.

B-2-1 Monte Carlo

In MC policy methods, evaluation of an approximated policy and value function is important. As the policy is steered to a more optimal solution with respect to the current value function, the value function is changed in the direction to be a better fit to the value function of the new policy.[8, 56] This way the value function and the policy are like one another's moving target. Because the value function and policy shift, optimality is reached for both. Policy improvement relies on making the policy greedy towards the state-action-value function. Algorithm 4 will give a clearer view of the steps to be taken.

Algorithm 4 Monte Carlo Policy Algorithm
Initialisation:
$T \leftarrow$ terminal state
$\pi \leftarrow$ arbitrary starting policy
$Q \leftarrow$ arbitrary state-action-value function
$\gamma \leftarrow$ discount factor
$\alpha \leftarrow$ step size parameter

for $t \leftarrow 0 : (T-1)$ **do**
  Generate episode using policy π
  for each (s_t, a_t) in the episode **do**
    $r_t \leftarrow$ reward following (s_t, a_t)
    $R_t \leftarrow \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_t - Q(s_t, a_t)]$
  end for
  for each s_t in the episode **do**
    $\pi(s_t) \leftarrow \arg\max_a Q(s_t, a_t)$
  end for
end for

B-2-2 Dynamic Programming

Very similar to policy evaluation MC method there is a method developed within DP, policy iteration. Policy iteration also makes use of a two stage evaluation and improvement cycle. The difference is that in contrast to MC it can update after each time step and does not have to wait until the end of the episode.[8, 30] The value function update rule stays the same as Equation B-1 and therefore the new greedy policy, π', will become

$$\begin{aligned} \pi'(s) &= \arg\max_a Q^{\pi}(s,a) \\ &= \arg\max_a \mathbb{E}\{r_{t+1} + \gamma V^{\pi}(s')\} \\ &= \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')]. \end{aligned} \tag{B-6}$$

B-2-3 Policy Gradient

Policy Gradient (PG) methods rest on the idea of using gradient descent or ascent to update its policy and therefore maximising the expected return $J(\pi_\theta)$ directly.[56, 57] Here the policy is defined as $\pi_\theta : \mathcal{S} \times \mathcal{A} \times \theta$ and denotes the probability of selecting an action a for state s given policy parameter vector θ. The algorithm will keep updating the vector θ as these parameters represents the the final structure of the policy and thus the value of the controller gain. The parameter update direction is given by the gradient of the expected return $\nabla_\theta J(\pi_\theta)$, i.e. takes the direction of steepest descent or ascent of the return that is to be expected. As PG is an algorithm that operates in continuous time the gradient is defined as

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta \int_\tau P_\theta(\tau) R_t(\tau) d\tau, \tag{B-7}$$

where τ is the trajectory, i.e. $\tau = (s_0, a_0, s_1, a_1, ..., s_T, a_T)$, and $P_\theta(\tau)$ is the probability for the agent selecting an action a in a state s according to policy $\pi_\theta(a|s)$, i.e. $P_\theta(\tau) = \mathbb{P}(\tau|\theta)$. The update parameter θ is then defined as

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta), \tag{B-8}$$

here $\alpha \in [0, 1]$ is a learning rate.

Bibliography

[1] M. Maurer, J. C. Gerdes, B. Lenz, and H. Winner, *Autonomous Driving*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016.

[2] T. A. S. Nielsen and S. Haustein, "On sceptics and enthusiasts: What are the expectations towards self-driving cars?," *Transport Policy*, vol. 66, pp. 49–55, 2018.

[3] D. E. Kirk, *Optimal control theory: An introduction / Donald E. Kirk*. Mineola, N.Y.: Dover Publications, 2004.

[4] H. Kwakernaak and R. Sivan, *Linear optimal control systems*. New York and Chichester: Wiley-Interscience, 1972.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning."

[6] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354 EP –, 2017.

[7] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm."

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction / Richard S. Sutton and Andrew G. Barto*. Adaptive computation and machine learning, Cambridge, Mass. and London: MIT Press, 1998.

[9] D. Zhao, Z. Hu, Z. Xia, C. Alippi, Y. Zhu, and D. Wang, "Full-range adaptive cruise control based on supervised adaptive dynamic programming," *Neurocomputing*, vol. 125, pp. 57–67, 2014.

[10] E. F. Camacho, D. R. Ramírez, D. Limón, D. M. de La Peña, and T. Álamo, "Model predictive control techniques for hybrid systems," *IFAC Proceedings Volumes*, vol. 42, no. 17, pp. 1–13, 2009.

[11] S. E. Li, Z. Jia, K. Li, and B. Cheng, "Fast online computation of a model predictive controller and its application to fuel economy–oriented adaptive cruise control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 3, pp. 1199–1209, 2015.

[12] R. Schmied, H. Waschl, R. Quirynen, M. Diehl, and L. del Re, "Nonlinear mpc for emission efficient cooperative adaptive cruise control," *IFAC-PapersOnLine*, vol. 48, no. 23, pp. 160–165, 2015.

[13] Krzysztof Czarnecki, "Operational design domain for automated driving systems - taxonomy of basic terms."

[14] P. Polack, B. d'Andréa Novel, M. Fliess, A. d. La Fortelle, and L. Menhour, "Finite-time stabilization of longitudinal control for autonomous vehicles via a model-free approach."

[15] D. Görges, "Relations between model predictive control and reinforcement learning," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 4920–4928, 2017.

[16] Q. Zhu, B. Dai, Z. Huang, Z. Sun, and D. Liu, "An adaptive longitudinal control method for autonomous follow driving based on neural dynamic programming and internal model structure," *International Journal of Advanced Robotic Systems*, vol. 14, no. 6, p. 172988141774071, 2017.

[17] C. Desjardins and B. Chaib-draa, "Cooperative adaptive cruise control: A reinforcement learning approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 4, pp. 1248–1260, 2011.

[18] X. Xu, D. Hu, and X. Lu, "Kernel-based least squares policy iteration for reinforcement learning," *IEEE transactions on neural networks*, vol. 18, no. 4, pp. 973–992, 2007.

[19] B. Bischoff, D. Nguyen-Tuong, T. Koller, H. Markert, and A. Knoll, "Learning throttle valve control using policy search," in *Advanced Information Systems Engineering* (D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, C. Salinesi, M. C. Norrie, and Ó. Pastor, eds.), vol. 7908 of *Lecture Notes in Computer Science*, pp. 49–64, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[20] X.-s. WANG, Y.-h. CHENG, and W. SUN, "A proposal of adaptive pid controller based on reinforcement learning," *Journal of China University of Mining and Technology*, vol. 17, no. 1, pp. 40–44, 2007.

[21] M. Sedighizadeh and A. Rezazadeh, "Adaptive pid controller based on reinforcement learning for wind turbine control," *World Academy of Science, Engineering and Technology, International Journal of Electrical and Information Engineering*, vol. 2, no. 1, 2008.

[22] M. Kashki, Y. L. Abdel-Magid, and M. A. Abido, "A reinforcement learning automata optimization approach for optimum tuning of pid controller in avr system," in *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence* (D.-S. Huang, D. C. Wunsch, D. S. Levine, and K.-H. Jo, eds.), vol. 5227 of *Lecture Notes in Computer Science*, pp. 684–692, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[23] A. el Hakim, H. Hindersah, and E. Rijanto, "Application of reinforcement learning on self-tuning pid controller for soccer robot multi-agent system," in *2013 Joint International Conference on Rural Information & Communication Technology and Electric-Vehicle Technology (rICT & ICeV-T)*, pp. 1–6, IEEE, 26-Nov-13 - 28-Nov-13.

[24] D. P. Bertsekas, *Dynamic programming and optimal control*. Belmont, Mass.: Athena Scientific, 2nd ed. ed., 2001.

[25] W. Levine and M. Athans, "On the determination of the optimal constant output feedback gains for linear multivariable systems," *IEEE Transactions on Automatic Control*, vol. 15, no. 1, pp. 44–48, 1970.

[26] J. Lunze, *Regelungstechnik 2*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016.

[27] M. van Otterlo and M. Wiering, "Reinforcement learning and markov decision processes," in *Reinforcement Learning* (M. Wiering and M. van Otterlo, eds.), vol. 12 of *Adaptation, Learning, and Optimization*, pp. 3–42, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[28] F. S. Hillier, E. A. Feinberg, and A. Shwartz, *Handbook of Markov Decision Processes*, vol. 40. Boston, MA: Springer US, 2002.

[29] C. Boutilier, T. Dean, and S. Hanks, "Decision-theoretic planning: Structural assumptions and computational leverage," *Journal of Artificial Intelligence Research*, vol. 11, pp. 1–94, 1999.

[30] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1972.

[31] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[32] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

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

[34] S. Liang and R. Srikant, "Why deep neural networks for function approximation?."

[35] B. C. Csáji, "Approximation with artificial neural networks," Master's thesis, Faculty of Sciences Eötvös Loránd University Hungary.

[36] M. van Gerven, "Computational foundations of natural intelligence," *Frontiers in computational neuroscience*, vol. 11, p. 112, 2017.

[37] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[38] S. Krig, "Feature learning and deep learning architecture survey," in *Computer Vision Metrics* (S. Krig, ed.), vol. 29, pp. 375–514, Cham: Springer International Publishing, 2016.

[39] S. Krig, "Feature learning architecture taxonomy and neuroscience background," in *Computer Vision Metrics* (S. Krig, ed.), vol. 29, pp. 319–374, Cham: Springer International Publishing, 2016.

[40] F. X. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanism*. Washington: Spartan Books, 1961.

[41] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," in *SIAM Journal on Control and Optimization*, pp. 1008–1014, MIT Press, 2000.

[42] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning* (E. P. Xing and T. Jebara, eds.), vol. 32 of *Proceedings of Machine Learning Research*, (Bejing, China), pp. 387–395, PMLR, 22–24 Jun 2014.

[43] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning."

[44] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (G. Gordon, D. Dunson, and M. Dudík, eds.), vol. 15 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 315–323, PMLR, 11–13 Apr 2011.

[45] G. Lin and W. Shen, "Research on convolutional neural network based on improved relu piecewise activation function," *Procedia Computer Science*, vol. 131, pp. 977–984, 2018.

[46] P. Petersen and F. Voigtlaender, "Optimal approximation of piecewise smooth functions using deep relu neural networks," *Neural networks : the official journal of the International Neural Network Society*, vol. 108, pp. 296–330, 2018.

[47] K. Ogata, *Modern control engineering*. Upper Saddle River, NJ and London: Prentice Hall, 3rd ed. ed., 1997.

[48] L. Puccetti, C. Rathgeber, and S. Hohmann, "Reinforcement learning for output control of a linear serial plant." Unpublished manuscript, Munich, 2019.

[49] A. Schwartz, "A reinforcement learning method for maximizing undiscounted rewards," in *Proceedings of the Tenth International Conference on International Conference on Machine Learning*, ICML'93, (San Francisco, CA, USA), pp. 298–305, Morgan Kaufmann Publishers Inc., 1993.

[50] P. L. Bartlett, "An introduction to reinforcement learning theory: Value function methods," in *Advanced Lectures on Machine Learning* (G. Goos, J. Hartmanis, J. van Leeuwen, S. Mendelson, and A. J. Smola, eds.), vol. 2600 of *Lecture Notes in Computer Science*, pp. 184–202, Berlin, Heidelberg: Springer Berlin Heidelberg, 2003.

[51] J. Kober and J. Peters, *Reinforcement Learning in Robotics: A Survey*, pp. 9–67. Cham: Springer International Publishing, 2014.

[52] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, Aug 1988.

[53] C. Szepesvári, "Algorithms for reinforcement learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 4, no. 1, pp. 1–103, 2010.

[54] C. J. C. H. Watkins, *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, May 1989.

[55] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Tech. Rep. TR 166, Cambridge University Engineering Department, Cambridge, England, 1994.

[56] M. P. Deisenroth, "A survey on policy search for robotics," *Foundations and Trends in Robotics*, vol. 2, no. 1-2, pp. 1–142, 2011.

[57] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems 12* (S. A. Solla, T. K. Leen, and K. Müller, eds.), pp. 1057–1063, MIT Press, 2000.

Glossary

List of Acronyms

ACC	Adaptive Cruise Control
AI	Artificial Intelligence
ANNs	Artificial Neural Networks
CACC	Cooperative Adaptive Cruise Control
CC	Cruise Control
DP	Dynamic Programming
DPG	Deterministic Policy Gradient
FC	Fully Connected
FIR	Finite Impulse Response
LM	Levenberg-Marquardt
MC	Monte Carlo
MDP	Markov Decision Process
ML	Machine Learning
OC	Optimal Control
OOC	Optimal Output Control
PG	Policy Gradient
QFL	Quadratic Feature Layer
R2V	Road-to-Vehicle
RBF	Radial Basis Function

RL	Reinforcement Learning
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
TD	Temporal Difference
V2V	Vehicle-to-Vehicle