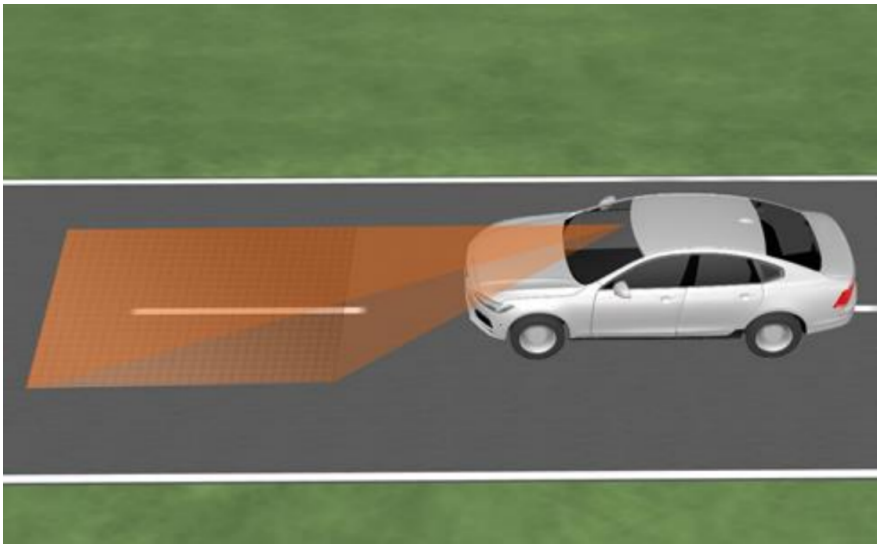
CHALMERS
UNIVERSITY OF TECHNOLOGY



Neural network controller for semi-active suspension systems with road preview

A Reinforcement learning approach using deep deterministic policy gradient for semi-active wheel suspension with road preview

Master's thesis in Master Programme Systems, Control and Mechatronics

ANTON GUSTAFSSON
ALEXANDER SJÖGREN

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019

Neural network controller for semi-active suspension systems with road preview

A Reinforcement learning approach using deep deterministic policy gradient for semi-active wheel suspension with road preview

ANTON GUSTAFSSON
ALEXANDER SJÖGREN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Automatic Control Group
Chalmers University of Technology
Gothenburg, Sweden 2019

Neural network controller for semi-active suspension systems with road preview
A Reinforcement learning approach using deep deterministic policy gradient for semi-active wheel suspension with road preview
ANTON GUSTAFSSON
ALEXANDER SJÖGREN

Supervisor: Anton Albinsson, Volvo Car Corporation
Examiner: Balázs Adam Kulcsár, Electrical Engineering

Master's Thesis 2019:NN
Department of Electrical Engineering
Division of Systems and Control
Automatic Control Group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization of a car equipped with a vision system that is gathering preview information of the road.

Typeset in LaTeX
Gothenburg, Sweden 2019

Neural network controller for semi-active suspension systems with road preview
A Reinforcement learning approach using deep deterministic policy gradient for semi-active wheel suspension with road preview
ANTON GUSTAFSSON
ALEXANDER SJÖGREN
Department of Electrical Engineering
Chalmers University of Technology

Abstract

By use of semi-active wheel suspension, ride comfort can be significantly improved compared to conventional passive suspension. Semi-active wheel suspension systems are able to change the stiffness of the dampers, and hence adapt to the road profile. If preview data of the road profile is utilized of the suspension controller, the ride comfort can be further improved. Lately, machine learning and neural networks have got a lot of attention, and not the least neural network controllers trained by reinforcement learning. In the last couple of years, learning techniques for continuous environments have shown promising results, e.g. the deep deterministic policy gradient. Making it interesting for many realistic control tasks. This thesis presents a neural network control approach, trained by the deep deterministic policy gradient. A control approach for a simple quarter-car model is proposed, as well as a control approach for a more detailed full-car model. Both control approaches are evaluated on the same models that they are trained on, and the full-car controller is also evaluated on an even more detailed model in IPG CarMaker. The simulation results show that a neural network controller with road preview can reach significantly improved ride comfort for certain roads, compared to a passive suspended system and a conventional reactive controller for semi-active suspension.

Keywords: Semi-active suspension, neural network, reinforcement learning, deep deterministic policy gradient, road preview, machine learning, full-car model, quarter-car model, model predictive control, ride comfort, actor-critic.

Acknowledgements

We would like to thank Anton Albinsson, our supervisor at Volvo Cars, for his excellent guidance and support throughout the thesis. We would also like to thank Wolgan Dovland Herrera, Stavros Angelis, Goran Vasilevski and especially Pontus Carlsson for their support regarding technical matters. Finally, we would like to express our greatest gratitude to Balázs Adam Kulcsár, our supervisor and examiner at Chalmers University of Technology, for his support regarding technical matters and the thesis, as well as for his dedication to this project.

Anton Gustafsson and Alexander Sjögren, Gothenburg, May 2019

Nomenclature

Abbreviations

ANN	Artificial Neural Network
COG	Center Of Gravity
CCD	Continuously Controlled Damping
DDPG	Deep Deterministic Policy Gradient
DOF	Degree Of Freedom
DQL	Deep Q-Learning
DQN	Deep Q-Network
LPV	Linear Parameter Varying
MDP	Markov Decision Process
MPC	Model Predictive Control
NN	Neural Network
RL	Reinforcement Learning
RMS	Root Mean Square
VCC	Volvo Cars Corporation

Contents

List of Figures

List of Tables

1
Introduction

Wheel suspension systems have for a long time been an important feature for improving ride comfort and handling performance in vehicles. Improvements in handling performance is achieved by keeping the vehicle body and wheels as close to the road surface as possible. Ride comfort improvements are usually achieved by decreasing the vehicle's body accelerations. For a passive suspension system, this is typically achieved by using soft suspension. A reduction of body accelerations also increases safety aspects, e.g. reduces driver fatigue [1]. Better handling is typically achieved by using stiff suspension. Thus, increasing the grip between the wheels and the road. Thereof, for passive suspension, handling performance and ride comfort are to some extent in conflict with each other [2].

In step with the technical evolution, wheel suspension systems have gone from basic spring systems and passive damping systems, to nowadays active and semi-active damping systems. By using these controllable suspension systems, the conflict between handling performance and ride comfort can be reduced. For these systems, many different control approaches have been used. Recently, machine learning has got a lot of attention due to its rising performance, especially reinforcement learning control algorithms. In this thesis a control approach for semi-active wheel suspension using reinforcement learning will be presented. Furthermore, the presented approach is making use of road profile measurements, given from a camera mounted on the vehicle. Most of today's active and semi-active suspension systems are reactive in the sense that the controller output is based on the acceleration of the chassis that is measured by a sensor. By using estimations of the road profile, the suspension system controllers can be designed to adapt to the road profile. Hence, better comfort and grip between the wheels and the road may be achieved.

1.1 Background

Recently, Volvo Cars Corporation (VCC) and Chalmers University of Technology investigated the possibility to use model predictive control (MPC) and road preview from camera, in order to control a semi-active wheel suspension system. The thesis showed that MPC can offer good performance but to a high computational cost [3]. The high computational cost comes from solving a nonlinear optimization problem. A promising method for reducing computational complexity in nonlinear control problems is to use artificial neural networks. It has previously been shown that using neural networks in nonlinear control applications can be beneficial in terms of

computational cost. Artificial neural networks have been used as nonlinear black-box models of dynamic behaviours [4], [5], [6], but also as controller, outputting the control signal [7].

An artificial neural network needs to be trained in some way. One interesting approach is to use reinforcement learning (RL), because of the advantage of not needing a model. Furthermore, a model free RL approach can possibly learn aspects of the environment that a conventional model-based controller can not, because it is not limited by the accuracy of a model. Additionally, RL controllers have recently shown good results when applied to various control tasks, both in simulation [8], [9], [10] and in physical environments [11]. One of the proposed learning algorithms is called deep deterministic policy gradient (DDPG) [12]. The DDPG brings the benefits of the successful deep Q-learning algorithm [13] to continuous action spaces.

1.2 Purpose

The purpose of this thesis is to investigate whether neural network controller, trained by the DDPG algorithm, is a viable option for controlling a semi-active wheel suspension system, with access to road preview. This includes an investigation of the performance and the computational complexity of the controller. The purpose is also to investigate the effect of using road preview in combination with the neural network controller. These investigations are interesting in a scientific point of view, since there are very few similar studies.

1.3 Objective

The objective of the thesis is to to train a neural network, using the RL algorithm DDPG, to control a semi-active wheel suspension system with access to road preview. More specifically, this includes designing a model/environment of a semi-active suspension system and the associated vehicle, implement the RL algorithm DDPG, customize the algorithm to fit the current environment, train a neural network on the designed environment, and finally, evaluate the trained controller, in terms of performance and computational complexity, on the designed environment as well as in the model-based testing and design software IPG CarMaker. Furthermore the effect of using road preview data will be evaluated by comparing the performance of a controller with access to the data, to a controller without access to the data.

1.4 Delimitations

All inputs to the controller, i.e. road preview data and a linear combination of the states, will be given as noise free. The controller will be designed for one specific semi-active wheel suspension system and vehicle. The neural network will be trained on a designed environment with a few different road profiles. The controller will be

implemented and evaluated in simulation environments, i.e. the designed environment used for training and IPG CarMaker. All training sessions and evaluations will be made with a constant vehicle velocity. Various velocities between the sessions will be considered though. The reward function for the learning algorithm will not be optimized in terms of ride comfort and handling performance. Generally, the reward function will just consist of the body accelerations. The road preview data will consist of the height of the road profile along a fixed longitudinal axis. Thus, the road preview data will not take the angle of the car into consideration, which would be needed if the data was given from a camera on top of the car.

1.5 Related work

Techniques that make use of road preview for controlling wheel suspension systems has been proposed for many years. Early proposals of feedforward controllers were made by [14], [15] and [16]. An early proposal of an MPC was made in 1997 by [17]. In the paper, an MPC for an active suspension system with access to road preview shows significantly improved ride comfort, compared to a passive system. The study is carried out on a half-car model in simulation. The proposed controller is tolerant to significant amount of noise in the preview information. Furthermore, a real-time implementation of the MPC is shown to be feasible. In [18] and [19], two different control approaches for active suspension systems, using sensor data of the road profile, have been proposed; an MPC with road preview and a feedforward control approach with preview. Both controllers showed satisfactory results in simulation. The second controller was also evaluated on a real car. The achieved results, using road preview cameras and the feedforward controller, looks promising for usage in vehicles. In [20], another MPC for semi-active suspension on a full-car was proposed. No road preview was used, instead an observer was designed for estimating the road disturbances. The simulation results showed that the proposed MPC with observer performed close to as good as an MPC with road preview. Recently, VCC and Chalmers University of Technology proposed a nonlinear MPC with access to road preview, for a semi-active suspension system [3]. The thesis showed that good performance in terms of ride comfort could be achieved by the nonlinear MPC, in simulation on a full-car model. However, when implemented on a physical vehicle, it was shown that the computational complexity of the proposed controller was too large to run on a real-time system. Hence, the semi-active system with MPC did not perform better than a passive system.

In terms of reinforcement learning approaches for wheel suspension control, some different methods have been investigated. Already in 1996, a reinforcement learning algorithm was proposed to control a semi-active suspension system for a four wheeled passenger vehicle. In [21], the control objective is to minimize the vehicle's body accelerations by the use of online reinforcement learning. No preview of the road is available, hence the input to the controllers mainly is the road input disturbance for each wheel. A continuous action reinforcement learning automata algorithm is designed and an improvement in terms of body accelerations, compared to a passive reference, can be observed after just a couple of hours training. Further, despite

the large sensor noises affecting the controllers, a promising result was achieved. In [22], batch reinforcement learning, a technique to approximate solutions of optimal control problems, was proposed for training a semi-active suspension controller. The proposed controller is reactive, i.e. it does not use any preview data of the road, to determine its control signal. Also, the controller is model-free. Hence it can learn the aspects of the environment and does not need a model to determine the control signal. The report shows that a well tuned batch reinforcement learned controller can guarantee the overall best performance, compared to some of today's most common control strategies, such as *Skyhook control* and the quite similar *Acceleration driven damping control*. In [23], an online reinforcement learning method for controlling a quarter-car active suspension model is proposed. The controller is learning online, i.e. it adapts to the present road. The simulation results showed improvement of the body acceleration and displacement, as well as that the controller could adapt to the road rather quick. In summary, reinforcement learning controllers for wheel suspension systems have shown promising results both in simulation and using real hardware. However, reinforcement learning controllers that make use of road preview have not been particularly explored.

Further, reinforcement learning has recently shown impressing results in other applications. In [10] and [11], reinforcement learning, with neural networks, has been used to control quadrotors. Quadrotors are often hard to control, since they are sensitive and unstable systems. Despite this, classic and model-based control techniques are still often used to stabilize the flight. In the papers, they prove that model-free reinforcement learning algorithms successfully can be used for controlling quadrocopters. In [11], the machine learned controller is combined with some basic regular controller, and together an outstanding performance and, at the same time, computationally cheap controller is designed. Further, many of today's reinforcement learning algorithms are accelerating the training process to a fast convergence, which also increases the usability.

In [8], reinforcement learning is used to control urban traffic lights. The issues with traffic light optimization is that a large number of input information is available for controlling the system. Hence, traffic light systems often are controlled on localized parts of the traffic light network, and then the localized parts of the network is coordinated by a multi-agent setup. In this paper, reinforcement learning, in form of the deep deterministic policy gradient algorithm, is used in order to control the whole network as one part. This method is used to overcome the large scale of available state information. DDPG also enables multiple rewards, so each individual traffic light can be given an individual reward based on its chosen action. The results achieved, using this method, was that the algorithm worked good on a smaller system, but the higher complexity of the network, the more training was needed to achieve a good result. However, the results indicates that with a larger scale hardware infrastructure, the algorithm should work for larger traffic light networks. The same algorithm is used in [9], where it is used to control a planar bipedal walking robot in simulation. The goal is to make the robot walk by itself without any prior knowledge of itself or the world dynamics. The robot learns via trial and error

by giving it rewards based on how well it is performing a walking behaviour. The results showed yet again that the DDPG algorithm, for relatively complex systems, performs great within reasonable training times on standard hardware.

1.6 Ethics

Since the proposed technology is about improving comfort and performance/safety, the work is not particularly controversial in an ethical point of view. If taken to the edge, one argument could be that it is more important to focus on safety systems than to put effort in achieving a better ride comfort. However, since active and semi-active suspension systems also implies better grip between the wheels and the road, it also improves the safety of the vehicle.

2
Theory

This chapter contains theory about subjects that are of interest for this thesis. The chapter includes theory about vehicle suspension systems, and reinforcement learning. The section about vehicle suspension systems considers passive, semi-active and active systems, as well as describing damper and bump stop characteristics. It also contains a model of a quarter-car and full-car, as well as a brief description of ride comfort in terms of primary and secondary ride. Further, reinforcement learning learning is described, including the the basics of reinforcement learning and the elements that are used in the learning algorithm that is used in this thesis.

2.1 Vehicle suspension systems

Today, there are three main types of vehicle suspension systems, passive suspension, semi-active suspension and active suspension. In this section, a description of the three systems are presented. Furthermore, a quarter-car and a full-car model with semi-active suspension system are derived. Finally there is a brief section about ride comfort in terms of primary and secondary ride.

2.1.1 Different suspension systems

In this section, passive, semi-active and active suspension systems are described and their characteristics are compared.

2.1.1.1 Passive suspension

Today passive suspension is widely used in common cars. Passive systems simply reduce vertical, spinning and tilting (heave, roll and pitch) movements by passive springs and dampers. Passive indicates that no energy can be added to the system and that the characteristics of the springs and the dampers are fixed. Hence, passive systems have limited capability of completely controlling the vehicle dynamics. For a comfortable ride, it is desirable to limit accelerations of the body. While body accelerations can be reduced by a soft suspension system, tire-road contact and thereof handling benefits from stiff suspension [24]. Thus, a compromise between comfort and handling is needed when the characteristics of the springs and dampers are selected. It should also be noted that vehicles typically operate over different road profiles at different velocities, with different loads. These factors also need to be considered when selecting damper and spring characteristics [25].

2.1.1.2 Semi-active suspension

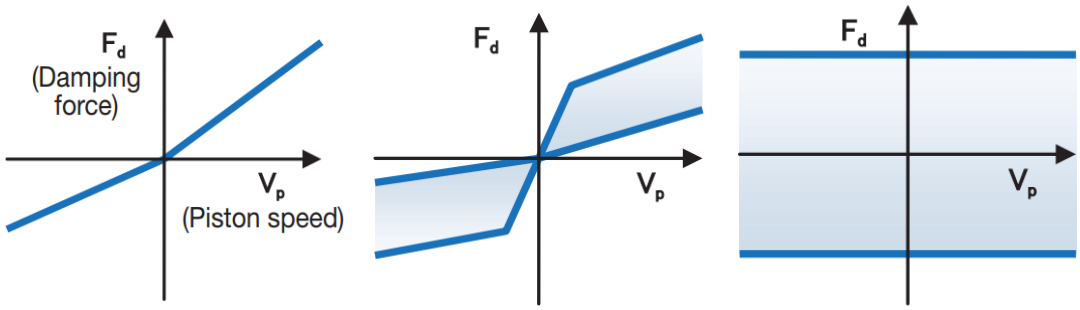
Semi-active systems are able to vary the viscous damping coefficient of the damper, following some control method. Hence, they are able to adapt to different road surfaces or driving modes, to a low energy cost [24]. Because of the good compromise between cost in terms of energy consumption, hardware and performance, semi-active dampers have got a lot of attention lately [22]. A drawback with semi-active systems is that they become nonlinear when changing the viscous damping coefficient [25], which is not really the case with passive suspension.

2.1.1.3 Active suspension

Active suspension systems are able to to raise and lower the chassis of the vehicle in order to suppress vibrations due to road irregularities. The chassis are controlled by independent actuators at each wheel. The actuators are however limited in the sense that while increasing ride comfort, the suspension working space must be preserved [26]. Still, using a proper control method, a good compromise between ride comfort and handling can be achieved. Many researchers recognize these systems to be among the best in order to improve the overall performance, due to their ability to manage contradictory parameters. The main drawback is that the systems are complex and expensive (in a manufacturing and energy consuming way), which has resulted in that only a small number of high-end cars and trucks use the system [27].

2.1.1.4 Comparison of damping characteristics

The characteristics of passive, semi-active and active dampers are illustrated in Figure 2.1. In particular, the characteristics describes how the damping force varies given the damper piston speed. In the figure it can be seen that the potential (the marked area) of semi-active and active damping is much larger than for the passive damper. As mentioned earlier, active suspension includes independent actuators. Hence they can deliver a bounded arbitrary force independent of the damper piston speed. As a consequence, active suspension has controllability potential in all four quadrants. Semi-active suspension is more limited and only has controllability in the first and the third quadrant. This makes active suspension more viable to effectively remove vibrations, but due to some drawbacks with active suspension systems, for example very large equipment sizes, semi-active suspension is used more often [28].



(a) Passive damping **(b)** Semi-active damping **(c)** Active damping

Figure 2.1: Illustration of damping characteristics for different damping systems [28].

2.1.2 Passive and semi-active dampers characteristics

A standard passive damper system consists of a piston rod, cylinder, piston valve, base valve and an accumulator (see Figure 2.2). The piston rod consists of a piston valve that can be calibrated to achieve the wanted stiffness. The damper is designed such that a piston rod can move vertically within the cylinder that is filled with oil. When the piston is moving in and out of the cylinder it causes a change in volume. This is compensated for by oil that is flowing, through the base valve, in or out of the accumulator. The change in pressure within the base valve and the piston valve causes a damping force on the piston.

The design of a semi-active damper is somewhat like the passive damper system, except that the piston- and base valves are replaced by check valves and a current-controlled valve is added (see Figure 2.2). The current-controlled valve is connected to the upper and the lower damper chamber and controls the flow of oil between them. The current opens and closes the valve to increase or decrease the restriction of flow. Hence, damping ratios can be changed by alternate the current given to the valve [29]. The following consequence is that the damping force can be changed given an arbitrary damper piston speed.

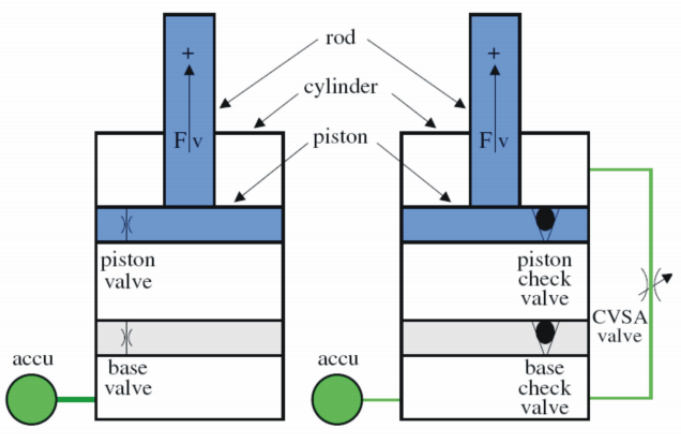


Figure 2.2: Illustration of a passive (left) and a semi-active (right) damper [29].

2.1.3 Modelling of dampers

Dampers are often very nonlinear, but they are often simplified and described as linear. Then, the damping force F_d is be described as

$$F_d = v_p \cdot c_d \tag{2.1}$$

where v_p is the piston speed and c_d is the damping coefficient. Considering a passive damper, the damping coefficient is constant. Consequently, the force acting on the damper is defined as a linear function over the piston speed. The difference in describing the damping force for a semi-active damper is that the damping coefficient is not necessarily constant. The damping coefficient can vary over the time as $c_d(t)$, which makes the equation nonlinear.

2.1.4 Bump stops/buffers

Bump stops, or sometimes called buffers, are designed to protect suspension components and increase ride comfort by limiting the bump travel without generating noise [30]. If the suspension system does not use bump stops while the vehicle runs over a bump, the dampers runs the risk of bottoming out. This means that the damper piston gets fully expanded or compressed, which can damage the suspension components. Furthermore, bump stops are often made of rubber and their main function is absorbing impacts and soften the force given to the chassis.

2.1.5 Primary and secondary ride

In vehicles, ride comfort is often referred to as primary and secondary ride. This means that ride comfort can be divided into different sections depending on the frequency of the vibrations affecting the vehicle. Primary ride is the section which involves low frequency vibrations and secondary rides are the section which involves high frequency vibrations. Primary ride describes events like large bumps, which generally can be described as a high amplitude with a low frequency that ranges from $0 - 4$ Hz [31]. Hence, movements in the rigid body (i.e. the chassis) are often

associated with primary ride. Furthermore, vibrations exposed to the driver with a frequency above 4 Hz are often associated with secondary ride. These vibrations can be associated with low amplitude and high frequency. They are often occurring because of imperfections in the road surface, for example very small bumps or cracks in the asphalt. Hence, secondary ride is often what passengers defines as ride comfort. In general, issues with primary ride will occur by a vehicle at high speed, whilst issues with secondary ride will occur by a vehicle at low speed. Moreover, primary ride is mostly controlled by anti-roll bars, shock absorbers and springs, whilst secondary ride mostly depend on tire properties, suspension and bush isolation [32].

2.1.6 Quarter-car model

A quarter-car model is a two degrees of freedom (vertical displacements) model and is shown in figure 2.3. It consists of a sprung mass m_s (a quarter of a car's mass), an unsprung mass m_u (mass of the wheel), two springs, and a damper [33], [34]. The sprung mass is illustrating a quarter of the car's chassis and weight. The sprung mass position is here called z_s. The unsprung mass is the mass of the wheel and it's position is called z_u. At the bottom of the figure a road surface is drawn and z_r is the height of the road surface's bumps. Between the wheel mass and the road surface, a spring k_u is placed, which purpose is to illustrate the characteristics and dynamics of the tire. Between the sprung and unsprung mass a damper c and a spring k_s is placed. These are supposed to reduce the vibrations from the road surface, which also may improve the comfort for the passengers of the car.

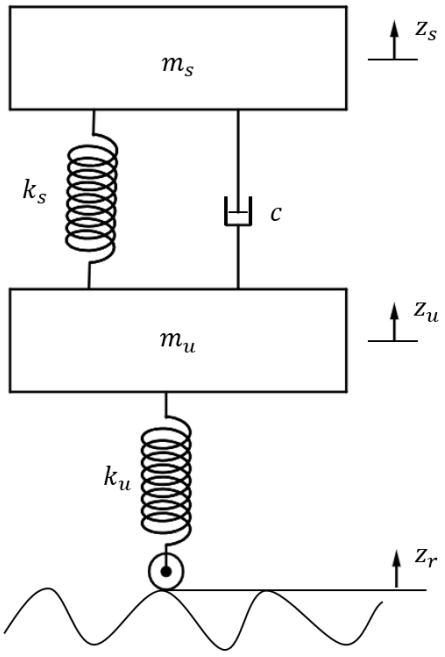


Figure 2.3: Illustration of the dynamics of a quarter-car model.

In order to model the quarter-car model, Newton's second law is used in order to derive the following equations.

$$m_s \cdot \ddot{z}_s = -k_s(z_s - z_u) - c(\dot{z}_s - \dot{z}_u) \tag{2.2}$$
$$m_u \cdot \ddot{z}_u = k_s(z_s - z_u) + c(\dot{z}_s - \dot{z}_u) - k_u(z_u - z_r) \tag{2.3}$$

2.1.7 Full-car model

The full-car model is shown in Figure 2.4. It is basically four quarter-car models that are all connected to the same sprung mass [33]. This introduces some additional degrees and movements to take into consideration. The full-car model uses seven degrees of freedom (heave, roll, pitch and vertical displacements of the four unsprung masses) [34]. In the figure, vertical displacements of each unsprung mass are noted as z_{u_i}, where $i \in \{1, 2, 3, 4\}$ represents the four wheels of the car according to Table 2.1. Furthermore, z_{s_i} corresponds to the vertical displacement of the sprung mass at each corner of the car. Road profiles affecting each wheel are noted as z_{r_i} Roll and pitch corresponds to the car's rotations around x- and y-axis respectively. The vertical displacement of the car's center of gravity (COG) is called heave. Distances a_1, a_2, b_1, b_2 are describing the lever between each wheel and the car's COG. In a linear representation of the full-car model, c_i corresponds to the damping coefficient used for each wheel's damper. Furthermore, the constants k_{s_i} and k_{u_i} represent the wheel suspension's spring constants, which are more detailed described in section 2.1.6.

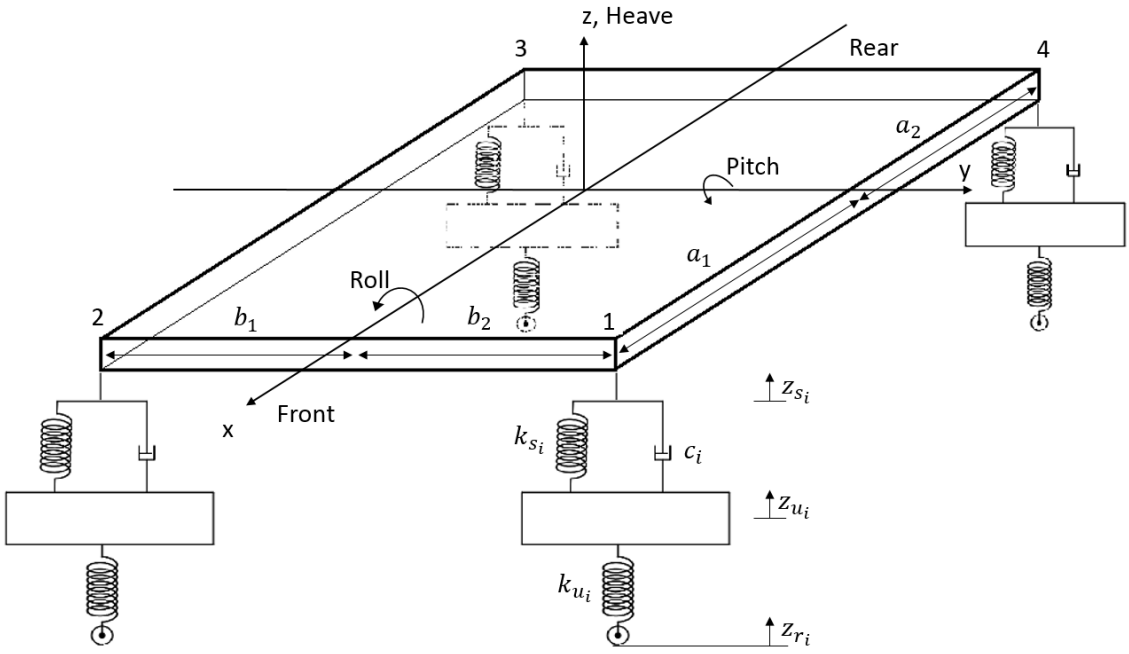


Figure 2.4: Illustration of the dynamics of a full-car model, $i \in \{1, 2, 3, 4\}$.

The parameters used for modelling the dynamics of the full-car model are shown in Table 2.2.

Table 2.1: Description of how to relate $i \in \{1, 2, 3, 4\}$ to the correct wheel.

i	Wheel
1	Front, Left
2	Front, Right
3	Rear, Right
4	Rear, Left

Table 2.2: Parameters for full-car model, $i \in \{1, 2, 3, 4\}$.

Parameter	Description	Unit
φ	Roll	rad
θ	Pitch	rad
z_s	Heave	m
z_{s_i}	Vertical displacement of sprung mass at each corner	m
z_{u_i}	Vertical displacement of unsprung mass at each corner	m
I_x	Roll inertia	kgm^2
I_y	Pitch inertia	kgm^2
m_s	Sprung mass	kg
m_{u_i}	Unsprung mass	kg
z_{r_i}	Vertical displacement of road	m
k_{s_i}	Spring constant, sprung mass	N/m
k_{u_i}	Spring constant, sprung tire	N/m
b_1	COG-distance right	m
b_2	COG-distance left	m
a_1	COG-distance front	m
a_2	COG-distance rear	m
c_i	Damping coefficient	Ns/m

By using Newton's second law, the linearized dynamics of the seven degree of freedom (DOF) full-car model can be described as in the following equations of motion [35]. Worth mentioning is that the model is using small angle approximation, i.e. $sin(\theta) \approx \theta$ and $cos(\theta) \approx 1$, which is a good approximation, since the angles of the car is relatively small in normal conditions of the road.

$$\begin{aligned} m_s\ddot{z}_s = & -c_1\Big(\dot{z}_s - \dot{z}_{u_1} + b_1\dot{\varphi} - a_1\dot{\theta}\Big) - c_2\Big(\dot{z}_s - \dot{z}_{u_2} - b_2\dot{\varphi} - a_1\dot{\theta}\Big) \\ & - c_3\Big(\dot{z}_s - \dot{z}_{u_3} - b_1\dot{\varphi} + a_2\dot{\theta}\Big) - c_4\Big(\dot{z}_s - \dot{z}_{u_4} + b_2\dot{\varphi} + a_2\dot{\theta}\Big) \\ & - k_{s_1}\Big(z_s - z_{u_1} + b_1\varphi - a_1\theta\Big) - k_{s_2}\Big(z_s - z_{u_2} - b_2\varphi - a_1\theta\Big) \\ & - k_{s_3}\Big(z_s - z_{u_3} - b_1\varphi + a_2\theta\Big) - k_{s_4}\Big(z_s - z_{u_4} + b_2\varphi + a_2\theta\Big) \end{aligned} \tag{2.4}$$

$$\begin{aligned} I_x\ddot{\varphi} = & -b_1c_1\Big(\dot{z}_s - \dot{z}_{u_1} + b_1\dot{\varphi} - a_1\dot{\theta}\Big) + b_2c_2\Big(\dot{z}_s - \dot{z}_{u_2} - b_2\dot{\varphi} - a_1\dot{\theta}\Big) \\ & + b_1c_3\Big(\dot{z}_s - \dot{z}_{u_3} - b_1\dot{\varphi} + a_2\dot{\theta}\Big) - b_2c_4\Big(\dot{z}_s - \dot{z}_{u_4} + b_2\dot{\varphi} + a_2\dot{\theta}\Big) \\ & - b_1k_{s_1}\Big(z_s - z_{u_1} + b_1\varphi - a_1\theta\Big) + b_2k_{s_2}\Big(z_s - z_{u_2} - b_2\varphi - a_1\theta\Big) \\ & + b_1k_{s_3}\Big(z_s - z_{u_3} - b_1\varphi + a_2\theta\Big) - b_2k_{s_4}\Big(z_s - z_{u_4} + b_2\varphi + a_2\theta\Big) \end{aligned} \tag{2.5}$$

$$\begin{aligned} I_y\ddot{\theta} = & \, a_1c_1\Big(\dot{z}_s - \dot{z}_{u_1} + b_1\dot{\varphi} - a_1\dot{\theta}\Big) + a_1c_2\Big(\dot{z}_s - \dot{z}_{u_2} - b_2\dot{\varphi} - a_1\dot{\theta}\Big) \\ & - a_2c_3\Big(\dot{z}_s - \dot{z}_{u_3} - b_1\dot{\varphi} + a_2\dot{\theta}\Big) - a_2c_4\Big(\dot{z}_s - \dot{z}_{u_4} + b_2\dot{\varphi} + a_2\dot{\theta}\Big) \\ & + a_1k_{s_1}\Big(z_s - z_{u_1} + b_1\varphi - a_1\theta\Big) + a_1k_{s_2}\Big(z_s - z_{u_2} - b_2\varphi - a_1\theta\Big) \\ & - a_2k_{s_3}\Big(z_s - z_{u_3} - b_1\varphi + a_2\theta\Big) - a_2k_{s_4}\Big(z_s - z_{u_4} + b_2\varphi + a_2\theta\Big) \end{aligned} \tag{2.6}$$

$$\begin{aligned} m_{u_1}\ddot{z}_{u_1} = & \, c_1\Big(\dot{z}_s - \dot{z}_{u_1} + b_1\dot{\varphi} - a_1\dot{\theta}\Big) + k_{s_1}\Big(z_s - z_{u_1} + b_1\varphi - a_1\theta\Big) \\ & - k_{u_1}\Big(z_{u_1} - z_{r_1}\Big) \end{aligned} \tag{2.7}$$

$$\begin{aligned} m_{u_2}\ddot{z}_{u_2} = & \, c_2\Big(\dot{z}_s - \dot{z}_{u_2} - b_2\dot{\varphi} - a_1\dot{\theta}\Big) + k_{s_2}\Big(z_s - z_{u_2} - b_2\varphi - a_1\theta\Big) \\ & - k_{u_2}\Big(z_{u_2} - z_{r_2}\Big) \end{aligned} \tag{2.8}$$

$$\begin{aligned} m_{u_3}\ddot{z}_{u_3} = & \, c_3\Big(\dot{z}_s - \dot{z}_{u_3} - b_1\dot{\varphi} + a_2\dot{\theta}\Big) + k_{s_3}\Big(z_s - z_{u_3} - b_1\varphi + a_2\theta\Big) \\ & - k_{u_3}\Big(z_{u_3} - z_{r_3}\Big) \end{aligned} \tag{2.9}$$

$$\begin{aligned} m_{u_4}\ddot{z}_{u_4} = & \, c_4\Big(\dot{z}_s - \dot{z}_{u_4} + b_2\dot{\varphi} + a_2\dot{\theta}\Big) + k_{s_4}\Big(z_s - z_{u_4} + b_2\varphi + a_2\theta\Big) \\ & - k_{u_4}\Big(z_{u_4} - z_{r_4}\Big) \end{aligned} \tag{2.10}$$

2.2 Reinforcement learning

Reinforcement learning is an area with machine learning. Reinforcement learning is a technique that uses the fundamental ideas of "learning by doing", and "trial and error". A common way of describing the learning process is the events of an *agent* acting in an influenceable *environment*. In the beginning, the agent has no information about which is the best *action* to take in each environment *state*. However, by taking an action in the environment, the agent will get a *reward* in return, telling the agent whether the action was good or not. When taking an action, the state of the environment will also be updated, and the new state is given to the agent. The loop is now closed and the agent is ready to take a new action. By repeating the process over and over again, the agent will learn how to act from its experience.

An example that often is referred to when describing the idea of reinforcement learning (e.g. in [36] and [37]), is how a baby learns to walk. A simple illustration is shown in Figure 2.5. Every time the baby improves by getting closer to walk, it will get a lot of positive rewards from its parents in terms of appreciation. If the baby on the other hand makes a bad move and falls down, it might get hurt and in turn it will get negative rewards. With time, the baby will use its experience in order to develop a walking technique (a good *policy*).

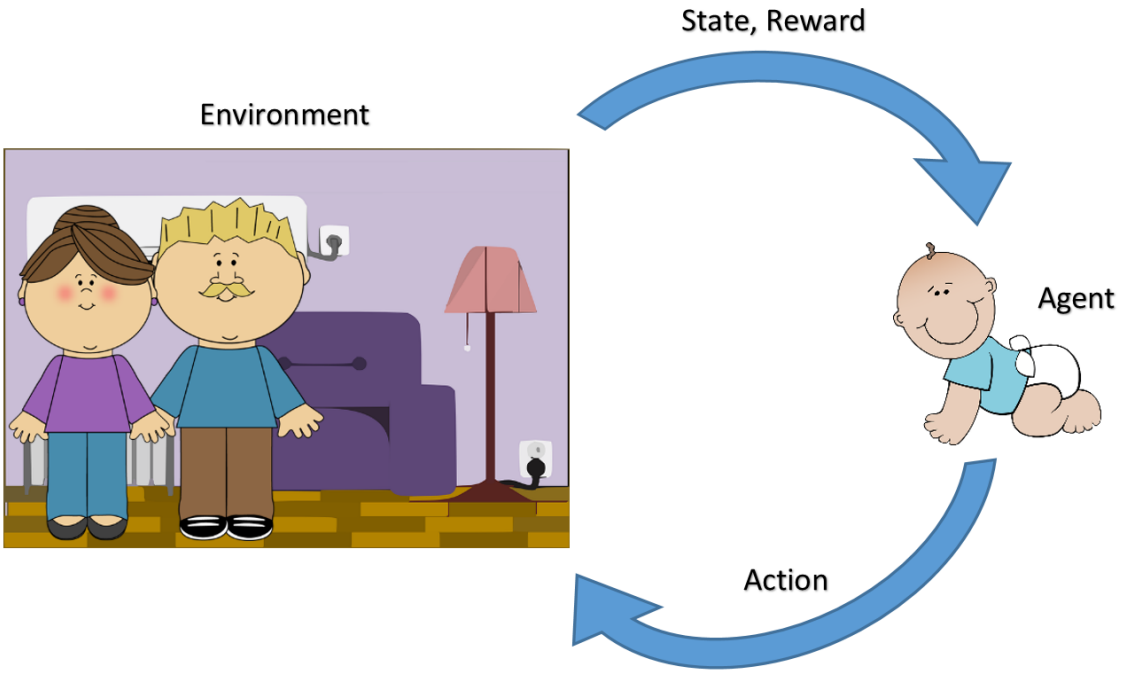


Figure 2.5: Illustration of the reinforcement learning concept.

I some cases, including the walking baby example, it might be hard to separate the environment and the agent. The agent is not necessary the entire robot or organism, and the environment is not necessary only what is outside of the robot or organism [38]. In the walking baby example, the position of the baby's body is definitely a part of the environment, since the state of the body has a very large impact on how the next action (body movement) will be made.

One aspect with reinforcement learning that in some cases is a bit problematic, is the trade-off between *exploration* and *exploitation*. Because in order for the agent to obtain rewards, it needs to exploit its experience and choose an action based on what it has done before [38]. Then the agent will never learn something new, it will never explore. An example could be how people act when deciding what they want to eat when visiting a restaurant. People that always choose their favourite dish, will most probably always be quite satisfied. However, they might miss out on something that is very good. People that often try new dishes, have the chance of experience new fantastic dishes, but they might also get something that they do not like. As mentioned, it is a trade-off, and it is often very hard to decide on how much

one wants to explore and exploit in order to achieve the best result in the end.

2.2.1 Elements of reinforcement learning

In reinforcement learning, one usually identify the agent and the environment as the two main elements. These have been introduced sufficiently in the previous section. Furthermore, there are three or four sub-elements that are necessary in reinforcement learning; a reward, a policy, a value function and in some cases a model [38].

Policy - A policy is what decides on which action to take in each state. The policy is what one wants learn or improve during learning. A policy can be very simple, such as a very simple function or as a look-up- table that matches states with actions. It can also be complex, e.g. using deep neural networks. The policy can be stochastic or deterministic. Deterministic policies are used in deterministic environments, i.e. environments where you are given state, a particular action affects the environment in a specific way. Deterministic policies maps a specific state to a specific action. Stochastic policies on the other hand, includes an uncertainty. In a specific state the chosen action may vary. Similarly, stochastic policies are used in stochastic environments. In stochastic environments, taking a particular action in a specific state may have several different outcomes.

Reward - A reward is a feedback signal directed to the agent, which tells how good or bad the current action was. The reward is necessary for learning and the design of it can be very important in order to achieve a desired result.

Value function - A value function is a function that expresses how good it is to be in a particular state. The value function is basically a computation or estimation of the cumulative rewards that an agent can expect to achieve in the future, given the current state. While the reward is telling what is good to do right now, the value tells what is good in the long run.

Model - In some cases, a model of the environment can be used in order for the agent to predict consequences, i.e. the new state and reward, of taking a certain action. Methods using models are called model-based whereas other methods are called model-free. Model-based methods are able to make use of planning, i.e. they can decide on which action is the best, based on predictions of the future. Hence, model-based methods can often learn a good policy requiring fewer samples of training than a model-free approach [39]. However it can be very hard to come up with a model that represents the environment good enough to make useful predictions, and there is always a risk of limiting the learning when using a model.

2.2.2 Markov decision process

A Markov decision process (MDP) is a discrete stochastic model, describing a control process. It includes a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a real valued reward function $r(s, a)$ and a state-transition probability function $p(s'|s, a)$. The state-transition

probability function describes the probability of ending up in state s' given state s and action a. Hence, the state-transition function describes the dynamics of the model, given the current action and state. It is necessary that the state s includes all information about the past that affects the future. If it does, the state is said to have the *Markov property*. Mathematically, the Markov property can be described as

$$Pr[S_{t+1} = s' \mid S_t] = Pr[S_{t+1} = s' \mid S_0, S_1, ..., S_t] \tag{2.11}$$

where S_t is the state at time instance $t = 1, 2, 3, ...$

The relationship between reinforcement learning and the MDP can be described as if the MDP is the framework of classic reinforcement learning [40], or to reconnect to previous sections; the MDP is used to describe the environment and the interaction properties with the agent. Given an agent that decides on which actions to take, the MDP will cause a trajectory with the look of

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ... \tag{2.12}$$

where A_t and R_t are the state, action and reward at time instance $t = 1, 2, 3, ...$

2.2.3 Rewards and returns

The reward signal is the feedback to the agent that tells whether the performed action was good or bad. The choice of reward signal can be hard to decide and may have a large impact on the result of the learning. It is very important that the reward signal represents the actual goal of the task, and not any subgoals. A used example considers the game of chess, the correct reward is to give $+1$ for winning and -1 for losing, not giving rewards for subgoals as taking out the opponent's pieces and similar [38]. The goal in reinforcement learning is to maximize the cumulative reward over time. In most cases, this can also be expressed as maximizing the expected *return*, where the return, G_t, is a function of the future rewards. In the most basic case with a finite episode length $t \in [0,\ T]$ (e.g. when playing a game), it is just the cumulative sum of the future rewards

$$G_t = R_{t+1} + R_{t+2} + .. + R_T \tag{2.13}$$

However, in many cases and especially control tasks, the tasks are continuous rather than having a finite episode length. Thus, the final time step would be $T = \infty$, which likely leads to an infinite return. To solve this problem, a discount factor, $\gamma \in [0,\ 1]$, is often used. The discount factor decides the importance of future rewards. The discounted return is written as

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{i=0}^{\infty} \gamma^i R_{t+i+1} \tag{2.14}$$

and by choosing $\gamma < 1$, the return becomes finite.

2.2.4 Value functions and policies

Almost every reinforcement learning method makes use of value functions. Value functions express the value of being in a particular state. The value of being in a particular state is a computation or estimation of the expected rewards in the future, the expected return. The value function can be seen as complement to the reward, since the value function can not be calculated or estimated without rewards, and its only purpose is to achieve more rewards [38]. Furthermore, the value function is highly dependent on the policy. The policy represents the behaviour of the agent, i.e. it decides on which actions to take. Future rewards depend on what actions that are performed, i.e. which policy that is used, thereby the dependency of the value function. For an MDP, given a policy π which is mapping states $s \in \mathcal{S}$ to certain actions $a \in \mathcal{A}$ through the probability function $\pi(a|s)$, the dependent value function is defined as

$$\begin{aligned} v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] &= \mathbb{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i R_{t+i+1} \mid S_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big] \end{aligned} \tag{2.15}$$

In many reinforcement learning methods a similar function called the *action-value function* is used. The action-value function returns the expected returns just as the value function, but it also takes a chosen action into consideration. The action-value function is defined as

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i R_{t+i+1} \mid S_t = s, A_t = a\right] \tag{2.16}$$

Value functions can be computed or estimated from experience. If an agent follows a policy π and stores an estimate of the actual return for each state, the stored estimate will converge to the actual return as the number of visitations of the current state approaches infinity. The same goes for the action-value function, with the difference of that an estimate of each state-action pair need to be stored. However in many cases, the number of states (or state-action pairs) is too large to allow storage of separate values for each state. If so, a value function approximation can be used as

$$\hat{v}_\pi(s, \mathbf{w}) \approx v_\pi(s) \tag{2.17}$$

where $\mathbf{w}$ is vector of parameters. There are several different methods for function approximation, for instance linear combinations of features and neural networks.

2.2.5 Basic algorithms for solving MDPs

In this section, the two basic RL algorithms *value iteration* and *policy iteration* are presented. In the following section, the algorithm *Q-learning* is introduced. Given a finite MDP, the three algorithms converge to an optimal policy π^*.

2.2.5.1 Policy iteration

Policy iteration is an algorithm for solving MDPs. The policy iteration algorithm (Algorithm 1) combines the use of policy evaluation and policy improvement. In the evaluation, the value $V(s)$ of each state is updated following the policy π, until it converges. The value function has converged once the difference between the updated value function and the previous one, Δ, is less than a chosen tolerance ε. In the improvement, the policy is updated to act *greedy* with regards to $V(s)$ in each state, i.e. choose the action that maximizes the expected value in the next state [38].

Algorithm 1: Policy iteration

Initialize $V(s) \in \mathbb{R}$, $\pi(s) \in \mathcal{A} \quad \forall \quad s \in \mathcal{S}$
while $\pi_{old} \neq \pi$ **do**
  while $\Delta > \varepsilon$ **do** `// Policy evaluation`
    $\Delta \leftarrow 0$
    for $s \in \mathcal{S}$ **do**
      $V_{old} \leftarrow V(s)$
      $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\Big[r + \gamma V(s')\Big]$
      $\Delta \leftarrow \max(\Delta, |V_{old} - V(s)|)$
  for $s \in \mathcal{S}$ **do** `// Policy improvement`
    $\pi_{old}(s) \leftarrow \pi(s)$
    $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma V(s')\Big]$
return V, π

2.2.5.2 Value iteration

The value iteration algorithm (Algorithm 2) can be seen as working backwards in relation to policy iteration. Instead of evaluate a policy and update it step by step, value iteration starts with computing the optimal value function for each state. Hence, the computation of the value function becomes slightly different. In value iteration, the value for each state is computed as the maximum possible value over all actions, instead of as the value with regards to a current policy. Once the value function has converged, a deterministic policy can be computed.

Algorithm 2: Value iteration

Initialize $V(s) \in \mathbb{R}$, $\pi(s) \in \mathcal{A} \quad \forall \quad s \in \mathcal{S}$
while $\Delta > \varepsilon$ **do**
  $\Delta \leftarrow 0$
  for $s \in \mathcal{S}$ **do**
    $V_{old} \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma V(s')\Big]$
    $\Delta \leftarrow \max(\Delta, |V_{old} - V(s)|)$
$\pi(s) = \text{argmax}_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma V(s')\Big]$
return V, π

2.2.6 Value-based, policy-based and actor-critic methods

Reinforcement algorithms can often be categorized as either *value-based* or *policy-based*. In value-based methods, the policy is computed based on learned value- or action-value functions. Policy-based methods does not learn a value function for each state. Instead, a policy is learned directly, and it is updated continuously during training. Value functions can still be used in order to update the policy, but they are not used to select any actions. The difference can also be demonstrated by comparing value iteration (value-based) and policy iteration (policy-based) which are presented in section 2.2.5. While policy iteration starts with a policy and updates it continuously until convergence, value iteration does not compute any policy until the value functions have converged. Generally, value-based methods are able to perform very good, given a discrete action space that is small enough. Policy-based methods can be beneficial in cases with continuous or stochastic action spaces, but they often end up in local optimums due to their difficulty of evaluate the current policy. Aside from pure value-based and policy-based methods, there are hybrid methods called *actor-critic* methods. Actor-critic methods include a policy-based actor that controls the behaviour of the agent, and a value-based critic that evaluates the behaviour of the agent.

2.2.7 Q-learning

Q-learning is an value-based algorithm that was introduced in 1989 and makes use of action-value functions, $Q(s, a)$ [41]. The algorithm basically returns a lookup table, Q-table, of the action-value for each state-action pair. It is an iterative process, where the Q-table is updated by exploring the environment. The environment is explored for multiple episodes (e.g. game sessions or trials), and for each episode, the approximation of the action-values is updated. The algorithm makes use of the *Bellman equation*, which expresses the action value function recursively

$$Q(s_i, a) = \mathbb{E}[r_i + \gamma \max_{a'} Q(s_{i+1}, a')] \tag{2.18}$$

The Bellman equation basically states that the value of the current state is equal to the reward of moving to the next state, added with the value of the next state taking the best action.

The algorithm (Algorithm 3) starts with initializing the Q-table. Thereafter, for each new episode, an initial state is defined. Further, for each step in episode, an action is chosen based on current state and a policy derived from Q. The policy can be any arbitrary policy, often a strategy called ε-greedy is used. An ε-greedy policy chooses a random action to a probability $\varepsilon \in [0, 1)$, and acts greedy (chooses the action with the largest action-value) with a probability of $1 - \varepsilon$. Once the action has been chosen, it is executed in the environment. In response, a reward and a new state is given back. The last step is to update the action-value with a learning rate α. The update is made with regards to minimizing the *temporal difference error* (TD-error)

$$\delta_i = Q(s_i, a) - \Big(r_i + \gamma \max_{a'} Q(s_{i+1}, a')\Big) \tag{2.19}$$

The TD-error is identical to the difference of the two sides in the Bellman equation, without the expected value. The sum of $r + \gamma \max_a Q(s', a)$ form what is called the *target*. The target is possibly more accurate than the current estimate of the action-value, since it includes information of the latest reward r. Given a finite MDP, the targets and the action-values will eventually become equal.

Algorithm 3: Q-learning

Initialize $Q(s, a) \in \mathbb{R}, \quad \forall \quad s \in \mathcal{S},\ a \in \mathcal{A}$
for *each episode* **do**
  Initialize s
  for *each step* **do**
    Choose a from s, using a policy derived from Q
    Execute action a and observe the next state s' and reward r
    $Q(s, a) \leftarrow Q(s, a) + \alpha\Big[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\Big]$
    $s \leftarrow s'$
  until $s = s_t$

2.2.8 Deep Q-learning

The theory about reinforcement learning that has been presented in this thesis has been assuming finite and discrete state and action spaces. Value-based reinforcement learning typically works very good in these environments, since as long as the state and action spaces are small enough, an optimal policy can be found. However, if the combination of state and action spaces is very large, the presented algorithms will require too much memory in order to work. In 2013, DeepMind Technologies presented an algorithm called *deep Q-learning* (DQL) [13]. The algorithm combines Q-learning with a deep neural network (called *deep Q-network* or DQN) for Q-function approximation. Hence, the algorithm is capable of generalizing over very large state spaces, as in e.g. chess, certain video games etc. A drawback is though, that by including non-linear function approximators, as neural networks, convergence is not longer guaranteed. The algorithm has shown great results especially for certain Atari video games, achieving superhuman performance with only the raw pixels and the score as input.

Aside from Q-learning and the deep neural network, DQL includes two fundamental techniques in order to work, *experience replay* and a *target network*. Experience replay stores the recent transitions (can be a very large amount) into a buffer. Then, for every step, a minibatch of transitions is randomly sampled from the buffer to update the network. Since the minibatch consists of independently sampled transitions, rather than correlated samples, the algorithm becomes more stable. The target network is a second network, a duplicate of the original one. The difference is that only the original network is updated during every training step. The target network is then synchronized with the original network, by every so often copy the weights from the original network. Just as its name indicates, the target network is used for generating targets. The benefit of using a network that does not update that often, is that with a temporarily fixed target, the learning becomes more stable.

If the original network would be used for generating targets, the targets would be constantly shifting and the training easily becomes unstable [42].

DQL is an *off-policy* algorithm. Off-policy means that the algorithm trains the agent using experience retrieved from policies other than the current one. In contrast to off-policy, *on-policy* algorithms trains the agent just using experience retrieved with the current policy. The DQL algorithm is off-policy since it uses experience replay. Hence it uses experience (stored transitions) generated from policies other than the current one. Generally, on-policy algorithms are faster, but they also risk ending up with a policy that is just locally optimal. Off-policy algorithms may be slower, but are more flexible for finding the optimal policy.

2.2.9 Policy gradients

Policy gradients are policy-based and improve the parameterized policy $\pi_\theta(a|s) = Pr[a|s]$ by repeating two steps. First, a score function $J(\theta)$ is used to evaluate how good the current policy is. (A loss function $L(\theta)$ can also be used if the goal is to minimize something rather than maximize a score.) Depending on the task and the environment, the score function can be different. For episodic environments, the score function can be computed as the expected return for the entire episode.

$$J(\theta) = \mathbb{E}_{\pi_\theta}[G_t] = \mathbb{E}_{\pi_\theta}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...] \tag{2.20}$$

The second step is to perform gradient ascent on the policy parameters θ, in order to maximize the score function. The update of the parameters is then made as

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{2.21}$$

where α is the learning rate. Once the score function is maximized, the optimal policy is found.

Compared to the value-based deep Q-learning, policy gradient methods generally converges smoother. In Q-learning, a relatively small change of the value functions may result in a considerable change of the policy. This is not really the case using policy gradients because of that the policy is updated stepwise. However, quite often policy gradients end up in local optimum instead of global. Furthermore, policy gradients have the ability to handle continuous or high dimensional action spaces, and the ability to learn stochastic policies[43].

2.2.10 Deep deterministic policy gradient

While DQL is a solution to problems with large state spaces, it can only handle relatively small and discrete action spaces. DQL can not be applied to continuous action spaces since it computes the optimal action in the sense of maximizing the action-value function. With a continuous action space, the number of actions are infinite, and hence, some kind of iterative optimization needs to be done. In 2016, Google DeepMind presented algorithm called DDPG [12]. The algorithm uses a couple of elements that are also used in DQL (e.g. deep neural networks, target

networks and experience replay), together with policy gradients to work with continuous action spaces.

The algorithm is based on the actor-critic off-policy *deterministic policy gradient* (DPG) algorithm that was introduced in 2014 [44]. The algorithm aims to learn a deterministic policy that maximizes the score function, i.e. the expected return in equation (2.20). The algorithm makes use of a parameterized critic $Q(s, a|\theta^Q)$ defining the action value of state-action pairs, and a parameterized deterministic actor function $\mu(s|\theta^\mu)$ which maps states to specific actions. Thus, the actor function specifies the current policy. The critic is updated as in Q-learning, using the Bellman equation. The actor is updated by following the gradient of the policy's performance, i.e. the gradient of the expected return

$$\begin{aligned}\nabla_{\theta^\mu} J &\approx \mathbb{E}[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)}] \\ &= \mathbb{E}[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s=s_t}]\end{aligned} \tag{2.22}$$

The target network approach in the DDPG is similar to what is used in DQL. The method has been adapted to the actor-critic architecture with two networks. Further, the main difference is the "soft" update of the target networks. The weights of the target networks are updated with a particular update rate for every step, τ, rather than just copy the weights from the original network every so often. Hence, the target networks will slowly track the original networks, improving stability of the learning.

As previously mentioned, reinforcement learning often needs to deal with the trade-off between exploration and exploitation. However, since the DDPG is off-policy, it can deal with the problem of exploration independently from learning. While learning with the DDPG, the exploration is made by adding noise to the actions generated from the actor policy.

$$\mu'(s_t) = \mu(s_t|\theta_t^\mu) + \mathcal{N} \tag{2.23}$$

Here, $\mu(s_t|\theta_t^\mu)$ is the actor policy, $\mathcal{N}$ is the noise and $\mu'(s_t)$ is the exploration policy. In the original paper, *Ornstein-Uhlenbeck noise* is used. The correlated noise process satisfies the stochastic differential equation

$$dx_t = \theta(\mu - x_t)dt + \sigma dW_t \tag{2.24}$$

where μ is a drift constant, θ and σ are positive parameters and W_t is the Wiener process [45]. However, the action noise can be chosen arbitrary to fit the environment [12].

The DDPG is presented in its completeness as Algorithm 4. It is noticeable that all operations are made within the inner loop. Thus, the algorithm is able to operate in both episodic and fully continuous environments.

Algorithm 4: Deep deterministic policy gradient

Initialize critic network $Q(s, a|\theta^Q)$ and actor network $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ
Initialize target networks Q' and μ' with target weights $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $\mathcal{R}$
for *each episode* **do**
  Receive initial observation state s_1
  Initialize exploration noise process $\mathcal{N}$
  for *each step* $t \in [1, T]$ **do**
    Choose $a_t = \mu(s_t|\theta_t^\mu) + \mathcal{N}_t$, using the current policy and exploration noise
    Execute action a_t and observe the new state s_{t+1} and reward r_t
    Store the transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{R}$
    Sample random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from $\mathcal{R}$
    Set the targets $y_i = r_i + \gamma Q'\Big(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}\Big)$
    Update the critic by minimizing the loss $L = \frac{1}{N}\sum_i \Big(y_i - Q(s_i, a_i|\theta^Q)\Big)^2$
    Update the policy of the actor using the sampled policy gradient

$$\nabla_{\theta^\mu} J \approx \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s=s_i}$$

    Update the target networks

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

2.3 Artificial neural networks

In many reinforcement learning methods, especially those made for larger state spaces, artificial neural networks (ANNs) are used. They are computing systems inspired by the biological neural networks that exists in, for instance, human brains. ANNs can be described as massive parallel computing systems that consists of extremely many simple processors with a large number of interconnections [46]. Thus, ANNs are a good tool for finding patterns in very complex systems. These patterns can often be way too complex for humans to find, and even harder to explicitly teach machines to recognize [47].

ANNs often consists of multiple layers, which are divided into input layers, hidden layers and output layers. These layers can be seen in Figure 2.6, where each layer consists of artificial neurons (the white circles), which often also are called nodes or units. Each neuron receives inputs (visualized as the arrows in the figure) from a specified number of neurons from other layers, or from an external source, and output a single signal. When a neuron receives a signal, it can process it and send

the processed signal further to other layer's neurons connected to it. The processing each neuron does to calculate a single output from multiple inputs, is a weighted sum as is shown in equation (2.25). In the equation the output y is calculated as the sum over all inputs where x_n is the input and w_n is its corresponding weight which expresses the importance for each input. In some cases a bias might also be added to the output.

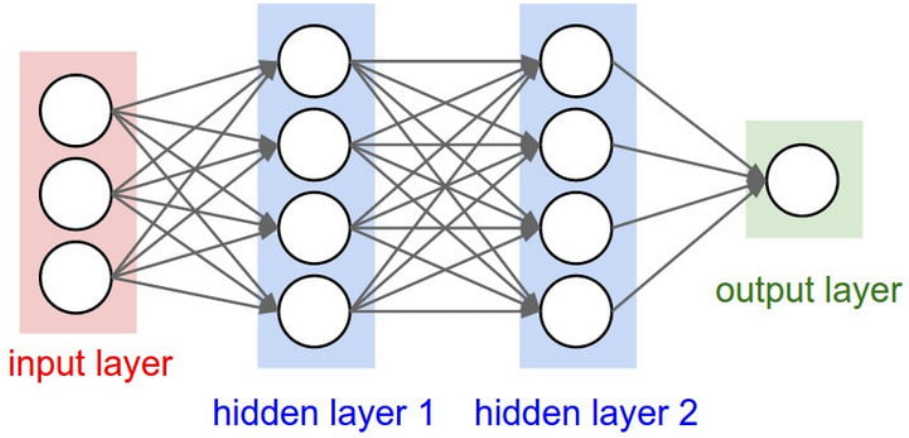


Figure 2.6: Visualization of the design of a simple artificial neural network [47].

$$y = \sum_n w_n x_n \tag{2.25}$$

Activation functions then determines how to process the output. There are multiple different activation functions that process the output in different ways. The most basic one is using a step function as activation function, which basically means that the activation function determines if the output is above a certain value or not [48]. If it is above the chosen value, the neuron can be considered activated. Neurons that are considered activated will output a value otherwise the neuron will output 0. This method is not commonly used, since it does not handle multiple activated inputs very well. Hence, more commonly used activation functions are tanh, sigmoid and ReLu, since they can use partly activated neurons. In the case of multiple activated neurons, they can then find the most activated neuron.

In many cases the distribution of the input values to a learning algorithm can change drastically between different datasets. This makes the machine learning algorithms sensitive to changes in distribution of the input values. Batch normalization is often used between layers to get rid of this problem. It normalizes the activations of each layer, which reduces the effect of different input distributions. Furthermore, using batch normalization might also speed up the learning for the network.

There are multiple different types of ANNs. Each type are made for specific use cases and different levels of complexity. Two of the most common neural networks are feedforward and recurrent neural networks. Feedforward neural network is one

of the most basic types of networks. It uses a technique that only can send information in one direction between input and output. A recurrent neural network can send information in multiple directions, which makes the technique suitable for more complex tasks as it possesses greater learning abilities.

3
Methods

In this chapter, the process and the used methods are presented. In the first two sections, the modelling of a quarter-car and a full-car are described. In the section that follows, the control approach using reinforcement learning is presented.

3.1 Quarter-car modelling

This section describes the modelling of a quarter-car. The reason for modelling a quarter-car model, was to try the reinforcement learning control approach. A quarter-car model is a simple model and includes less states than a full-car model, but it is still similar. Because of the smaller state space, the controller does not need as much training to find a good control policy. Hence, the result of applying the control approach on a quarter-car model could indicate useful aspects within a reasonable time.

In order to model the quarter-car dynamics, equations (2.2) and (2.3) are used. By using a state vector $\mathbf{x}$ and a control signal u the system can be described by a state space representation as

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}u, \quad \mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}u \tag{3.1}$$

with

$$\mathbf{x} = \begin{bmatrix} z_s & z_u & \dot{z_s} & \dot{z_u} \end{bmatrix}^T, \quad u = z_r \tag{3.2}$$

Here, the states are the positions and velocities for the sprung and unsprung mass. The input to the plant is just the height of the road. Since the damping coefficient c is the controllable parameter, the $\mathbf{A}$ and $\mathbf{C}$ matrices are not constant. Hence, the system is a linear parameter varying (LPV) control system.

$$\mathbf{A}(c(t)) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{-k_s}{m_s} & \frac{k_s}{m_s} & \frac{-c(t)}{m_s} & \frac{c(t)}{m_s} \\ \frac{k_s}{m_u} & \frac{-k_s-k_u}{m_u} & \frac{c(t)}{m_u} & \frac{-c(t)}{m_u} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{k_u}{m_u} \end{bmatrix} \tag{3.3}$$

The output of the plant is defined as

$$\mathbf{y} = \begin{bmatrix} \Delta z_s & \Delta z_u & \dot{z}_s & \dot{z}_u & \ddot{z}_s & \ddot{z}_u \end{bmatrix}^T \tag{3.4}$$

where Δz_s and Δz_u is the distance between the sprung and unsprung mass, respectively unsprung mass and road surface (see figure 3.1). Further, the choice to use

Δz_s and Δz_u, instead of the total height, was mainly based on the reality aspects, where Δz_s actually can be measured while z_s can not. Consequently, the matrices $\mathbf{C}$ and $\mathbf{D}$ are then defined as

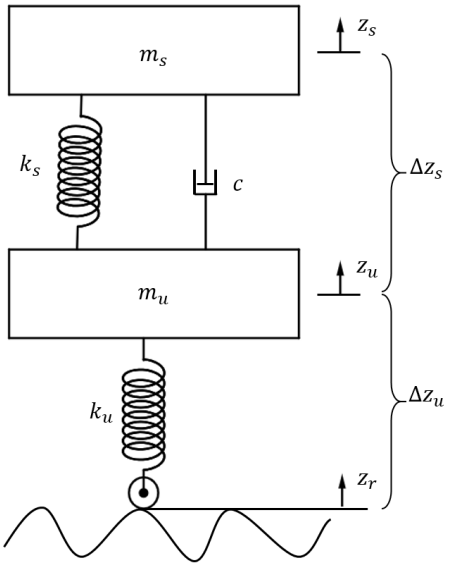


Figure 3.1: Illustration of the dynamics of a quarter-car model.

$$\mathbf{C} = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{-k_s}{m_s} & \frac{k_s}{m_s} & \frac{-c}{m_s} & \frac{c}{m_s} \\ \frac{k_s}{m_u} & \frac{-k_s-k_u}{m_u} & \frac{c}{m_u} & \frac{-c}{m_u} \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ \frac{k_u}{m_u} \end{bmatrix} \tag{3.5}$$

The system is continuous and in order to design a controller, the system needs to be discretized. The discretized system is then defined as

$$\mathbf{x}_{k+1} = \hat{\mathbf{A}}\mathbf{x}_k + \hat{\mathbf{B}}u_k, \quad \mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{D}u_k \tag{3.6}$$

where

$$\hat{\mathbf{A}} = I - \Delta t\mathbf{A}, \quad \hat{\mathbf{B}} = \Delta t\mathbf{B} \tag{3.7}$$

Δt is here defined as the period time, and I is the identity matrix.

To be able to understand the performance of the trained controller, controlling a semi-active damper, it was compared to a passive damper running over the exact same road profile. The passive damper was easily designed such that, instead of using a controller to choose different damping coefficients, the damper coefficient was set to a constant. Hence, the performance for the trained controller, relative to the passive damper, could be achieved.

3.2 Full-car modelling

This section describes the full-car model that was created and used for training and evaluation. The model is based on the linearized dynamics from section 2.1.7 with some added nonlinearities.

3.2.1 Dynamics

To model the full-car dynamics, equations (2.4)-(2.10) were used with some modifications. To get a more precise model, nonlinear dampers were used instead of linear representations. Also, nonlinear bump stops / buffers were added to the model (implying forces when the dampers reach their compression or expansion limit). With the modifications, the nonlinear dynamics are described as

$$\begin{aligned} m_s\ddot{z}_s = &\sum_{i=1}^{4}\Big(F_{d_i}(c_{d_i}, v_{d_i}) + F_{b_i}(p_{d_i})\Big) - k_{s_1}\Big(z_s - z_{u_1} + b_1\varphi - a_1\theta\Big) \\ &- k_{s_2}\Big(z_s - z_{u_2} - b_2\varphi - a_1\theta\Big) - k_{s_3}\Big(z_s - z_{u_3} - b_1\varphi + a_2\theta\Big) \\ &- k_{s_4}\Big(z_s - z_{u_4} + b_2\varphi + a_2\theta\Big) \end{aligned} \tag{3.8}$$

$$\begin{aligned} I_x\ddot{\varphi} = &\, b_1\Big(F_{d_1}(c_{d_1}, v_{d_1}) + F_{b_1}(p_{d_1})\Big) - b_2\Big(F_{d_2}(c_{d_2}, v_{d_2}) + F_{b_2}(p_{d_2})\Big) \\ &- b_1\Big(F_{d_3}(c_{d_3}, v_{d_3}) + F_{b_3}(p_{d_3})\Big) + b_2\Big(F_{d_4}(c_{d_4}, v_{d_4}) + F_{b_4}(p_{d_4})\Big) \\ &- b_1k_{s_1}\Big(z_s - z_{u_1} + b_1\varphi - a_1\theta\Big) + b_2k_{s_2}\Big(z_s - z_{u_2} - b_2\varphi - a_1\theta\Big) \\ &+ b_1k_{s_3}\Big(z_s - z_{u_3} - b_1\varphi + a_2\theta\Big) - b_2k_{s_4}\Big(z_s - z_{u_4} + b_2\varphi + a_2\theta\Big) \end{aligned} \tag{3.9}$$

$$\begin{aligned} I_y\ddot{\theta} = &- a_1\Big(F_{d_1}(c_{d_1}, v_{d_1}) + F_{b_1}(p_{d_1})\Big) - a_1\Big(F_{d_2}(c_{d_2}, v_{d_2}) + F_{b_2}(p_{d_2})\Big) \\ &+ a_2\Big(F_{d_3}(c_{d_3}, v_{d_3}) + F_{b_3}(p_{d_3})\Big) + a_2\Big(F_{d_4}(c_{d_4}, v_{d_4}) + F_{b_4}(p_{d_4})\Big) \\ &+ a_1k_{s_1}\Big(z_s - z_{u_1} + b_1\varphi - a_1\theta\Big) + a_1k_{s_2}\Big(z_s - z_{u_2} - b_2\varphi - a_1\theta\Big) \\ &- a_2k_{s_3}\Big(z_s - z_{u_3} - b_1\varphi + a_2\theta\Big) - a_2k_{s_4}\Big(z_s - z_{u_4} + b_2\varphi + a_2\theta\Big) \end{aligned} \tag{3.10}$$

$$\begin{aligned} m_{u_1}\ddot{x}_1 = &- c_1\Big(F_{d_1}(c_{d_1}, v_{d_1}) + F_{b_1}(p_{d_1})\Big) + k_{s_1}\Big(z_s - z_{u_1} + b_1\varphi - a_1\theta\Big) \\ &- k_{u_1}\Big(z_{u_1} - z_{r_1}\Big) \end{aligned} \tag{3.11}$$

$$\begin{aligned} m_{u_2}\ddot{x}_2 = &- c_2\Big(F_{d_2}(c_{d_2}, v_{d_2}) + F_{b_2}(p_{d_2})\Big) + k_{s_2}\Big(z_s - z_{u_2} - b_2\varphi - a_1\theta\Big) \\ &- k_{u_2}\Big(z_{u_2} - z_{r_2}\Big) \end{aligned} \tag{3.12}$$

$$\begin{aligned} m_{u_3}\ddot{x}_3 = &- c_3\Big(F_{d_3}(c_{d_3}, v_{d_3}) + F_{b_3}(p_{d_3})\Big) + k_{s_3}\Big(z_s - z_{u_3} - b_1\varphi + a_2\theta\Big) \\ &- k_{u_3}\Big(z_{u_3} - z_{r_3}\Big) \end{aligned} \tag{3.13}$$

$$\begin{aligned} m_{u_4}\ddot{x}_4 = &- c_4\Big(F_{d_4}(c_{d_4}, v_{d_4}) + F_{b_4}(p_{d_4})\Big) + k_{s_4}\Big(z_s - z_{u_4} + b_2\varphi + a_2\theta\Big) \\ &- k_{u_4}\Big(z_{u_4} - z_{r_4}\Big) \end{aligned} \tag{3.14}$$

Most notations are defined just as in Table 2.2, with a few exceptions. $F_{d_i}(c_{d_i}, v_{d_i})$ is the damping force at wheel i, depending on the control current to the damper c_{d_i} and the damper velocity v_{d_i}. A positive damper velocity corresponds to an extension and a negative damper velocity corresponds to a compression of the damper. The buffer forces are denoted $F_{b_i}(p_{d_i})$, and depends on the position of the damper p_{d_i}. The positions and velocities of the dampers are defined as

$$\begin{aligned} p_{d_1} &= z_s - z_{u_1} + b_1\varphi - a_1\theta, & v_{d_1} &= \dot{z}_s - \dot{z}_{u_1} + b_1\dot{\varphi} - a_1\dot{\theta} \\ p_{d_2} &= z_s - z_{u_2} - b_2\varphi - a_1\theta, & v_{d_2} &= \dot{z}_s - \dot{z}_{u_2} - b_2\dot{\varphi} - a_1\dot{\theta} \\ p_{d_3} &= z_s - z_{u_3} - b_1\varphi + a_2\theta, & v_{d_3} &= \dot{z}_s - \dot{z}_{u_3} - b_1\dot{\varphi} + a_2\dot{\theta} \\ p_{d_4} &= z_s - z_{u_4} + b_2\varphi + a_2\theta, & v_{d_4} &= \dot{z}_s - \dot{z}_{u_4} + b_2\dot{\varphi} + a_2\dot{\theta} \end{aligned} \tag{3.15}$$

The characteristics of the buffer forces are shown in Figure 3.2. The magnitude of the buffer forces quickly get much larger than the maximum controllable damping force. Hence, to achieve a smooth behaviour of the car body, the controller should act such that buffer forces are avoided if possible.

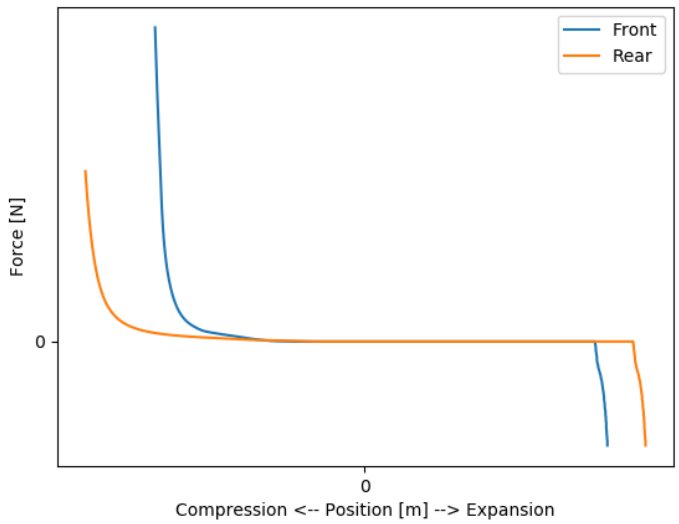


Figure 3.2: Illustration of the buffer forces for the front and rear wheels, depending on the damper piston position.

In Figure 3.3, the characteristic of a semi-active damper is shown. The area between the upper and lower limits, marked with bright grey, illustrates the possible force outputs from the damper. The stiffness of the damper, and thereby also the force, can be changed for an arbitrary piston speed by varying the current to the damper. The more current, the stiffer the damper becomes. The characteristics of the damper is linear between the lines 'Max' and 'Mid' stiffness and between 'Min' and 'Mid' stiffness. The stiffness of the damper, and the following force, is controlled by the current to the damper. Given a specific damper velocity, the current decides where, between the upper and the lower limit, the force output can be found.

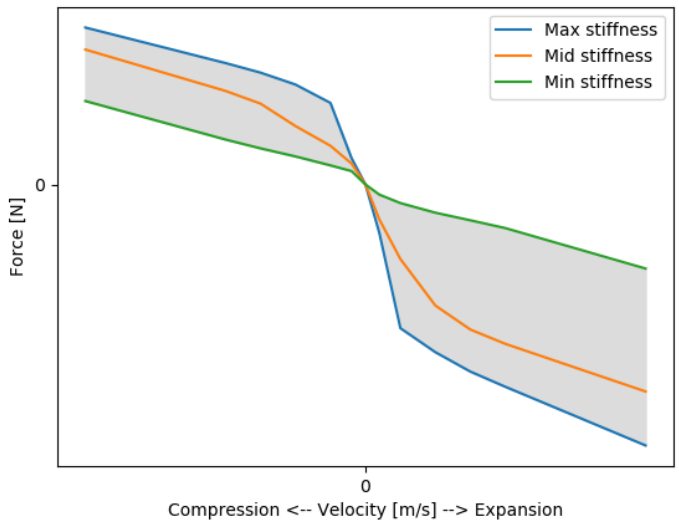


Figure 3.3: Characteristic of semi-active damper, depending on the damper piston velocity and the damper control signal.

To increase the precision of the model, a response delay representing the behaviour of the dampers when the input current is changed, was included. The response delay was modelled as a pure time delay and a change of rate limitation for changing the current to the damper. The pure delay is the time between sending a signal to the damper, to when the damper characteristics actually starts changing, based on the new signal. The change of rate limitation limits how fast the damper stiffness can change. This property is quite complex and nonlinear, but in this thesis it is approximated with a linear function.

The coefficient limiting the maximum change of rate of the damper, was determined using a step response of the real damper. The coefficient k_{c_d} was calculated as

$$k_{c_d} = \frac{\Delta F}{\Delta T} \tag{3.16}$$

where ΔF is the change in force, from minimum to 90% of maximum, and ΔT is the corresponding time.

The equations achieved for updating the force according to the delay are then formulated as

$$F_{new} = \begin{cases} F_{set}, & \text{if } |F_{new} - F_{set}| \leq k_{c_d}\, dt \\ F_{set} + k_{c_d}\, dt, & \text{if } F_{new} - F_{set} > k_{c_d}\, dt \\ F_{set} - k_{c_d}\, dt, & \text{if } F_{new} - F_{set} < -k_{c_d}\, dt \end{cases}$$

where F_{set} is the setting value (the chosen force from the controller), dt is the sampling time between each new setting value and F_{new} is the force affected by the delay and hence, the force used to update the plant.

To be able to understand the performance of the trained controller, controlling four semi-active dampers, it was compared to a passive damped reference. The characteristics of the passive dampers are shown in Figure 3.4. It can be observed that the passive dampers are nonlinear and also more realistic than the passive damper used for comparison in the quarter-car simulations.

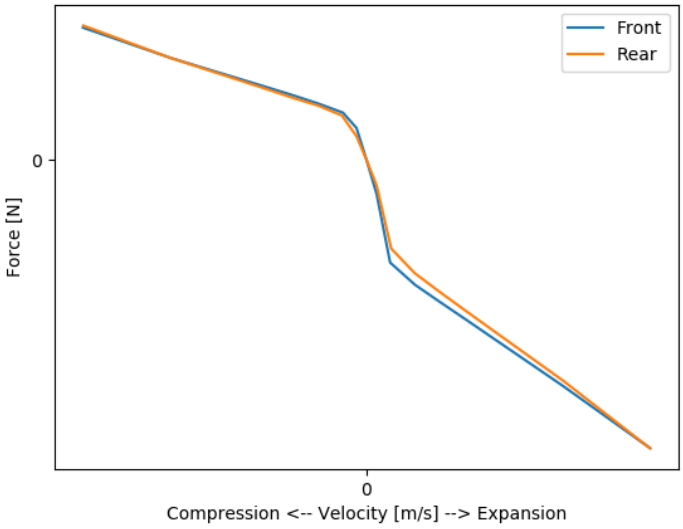


Figure 3.4: Illustration of the passive damper characteristics.

3.2.2 State space model

The full-car was modelled as a state space model with the state vector $\mathbf{x}$ and input vector $\mathbf{u}$. To maintain a linear state space model, the damper and buffer forces are calculated outside of the state space, and then included in the input vector.

$$\mathbf{x} = \begin{bmatrix} z_s & \dot{z}_s & \theta & \dot{\theta} & \varphi & \dot{\varphi} & z_{u_1} & \dot{z_{u_1}} & z_{u_2} & \dot{z_{u_2}} & z_{u_3} & \dot{z_{u_3}} & z_{u_4} & \dot{z_{u_4}} \end{bmatrix}^T \tag{3.17}$$

$$\mathbf{u} = \begin{bmatrix} F_1 & F_2 & F_3 & F_4 & z_{r_1} & z_{r_2} & z_{r_3} & z_{r_4} \end{bmatrix}^T \tag{3.18}$$

Here, F_i is the sum of the damping force and potential buffer force for each wheel, i.e. $F_i = F_{d_i} + F_{b_i}$ for $i = 1, 2, 3, 4$. z_{r_i} is the height of the road affecting each wheel. The output vector of the system is defined as

$$\mathbf{y} = \begin{bmatrix} \dot{z}_s & \ddot{z}_s & \theta & \dot{\theta} & \ddot{\theta} & \varphi & \dot{\varphi} & \ddot{\varphi} & \Delta z_1 & \cdots & \Delta z_4 & \Delta \dot{z}_1 & \cdots & \Delta \dot{z}_4 \end{bmatrix}^T \tag{3.19}$$

where Δz_i is the expansion or the compression distance of the damper at wheel i, and $\Delta \dot{z}_i$ is the corresponding velocity.

The dynamics can be described with the approximate discrete state space model

$$\begin{aligned} \mathbf{x}_{k+1} &= (\mathbf{I} + \mathbf{A}\Delta t)\mathbf{x}_k + \mathbf{B}\Delta t \mathbf{u}_k \\ \mathbf{y}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k \end{aligned} \tag{3.20}$$

where $\mathbf{x}_k$, $\mathbf{u}_k$ and $\mathbf{y}_k$ is the state vector, the input vector and the output vector at step k, respectively. $\mathbf{I}$ is the identity matrix and Δt is the step size. The matrices $\mathbf{A}$ and $\mathbf{B}$ are derived from the set of dynamic equations $\mathbf{f}$ (equation (3.8) - (3.14)), as $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ and $\frac{\partial \mathbf{f}}{\partial \mathbf{u}}$ respectively. The matrices $\mathbf{C}$ and $\mathbf{D}$ are appropriate matrices for achieving the output specified in (3.19).

3.3 Reinforcement learning control

In this section, the control problem is defined and a reinforcement learning approach for controlling semi-active suspension is presented. The approach is applied to both a quarter-car environment and a full-car environment, which previously have been presented in section 3.1 and 3.2.

3.3.1 Control problem identification

Recalling from section 2.2, reinforcement learning algorithms learn the agent how to act from experience. The agent basically gets to try different actions in the environment of interest, and with time it learns how it should act. In this thesis, reinforcement learning is used to train a controller for semi-active dampers. Thus, the agent is identified as the controller. The environment is a vertical vehicle model (quarter-car or full-car) that is running over a specified road. In order to control semi-active suspension systems, as in the quarter-car and full-car environments, a learning algorithm for continuous action spaces is needed. Furthermore, aside from the observed outputs from the two models, the road profile is said to be known for a specific distance in front of the car. This data is referred to as road preview data. When adding road preview data to the observation space (i.e. the data that is observed by the agent), the state space quickly becomes very large. Hence, the algorithm need to be able to handle large state spaces as well.

3.3.2 Deep deterministic policy gradient approach

Based on the specified requirements in section 3.3.1, the DDPG (Algorithm 4) is a suitable learning algorithm. It is designed for continuous action spaces and can

handle large state spaces. Also, it is an off-policy algorithm that strives to reach the global optimum, and does not get stuck in a local optimum. Another advantage is that the algorithm is versatile in the sense of that it is compatible with both episodic and fully continuous environments. This property allows for learning both by running over one specific road for several times, and by running over a never ending road for a specific time.

The DDPG algorithm was implemented in Python using the deep learning library TFLearn. The implementation of the algorithm was made based on [49], and strictly follows the structure of Algorithm 4. Just as in the original paper [12], Ornstein-Uhlenbeck noise was used for exploration.

The training of the agent was made by simulating the vehicle model running over a road profile, with the agent controlling the semi-active dampers. By trying out different actions given different observation states (i.e. trying out different stiffnesses of the semi-active dampers in different situations), and get a corresponding reward, the agent is able to learn how to act given different observation states. The observation state vector that is observed by the agent consists of the output vector of the vehicle model $\mathbf{y}$, and the road preview vector $\mathbf{z}_r$, as

$$\mathbf{y}_{obs} = \begin{bmatrix} \mathbf{y} \\ \mathbf{z}_r \end{bmatrix} \tag{3.21}$$

The road preview vector describes how the road is changing in front of the wheels. Every state in the vector describes the road height difference between the current longitudinal position and a future longitudinal position. The number of preview states for each wheel is specified as N and the longitudinal distance between the states are defined as Δ_{prev} (the density of the preview states). These parameters are selected and heavily impact the size of the road preview vector. Furthermore, the training were made with a constant velocity of the car. Different velocities were tested for different roads, but within each training session the velocity was constant.

The reward is the feedback to the agent, and hence fundamental for learning a desired behaviour. The desired behaviour of the controller is to maximize ride comfort and handling of the car. Two factors that in many ways are subjective. Since this thesis does not investigate which variables that contributes to a optimal ride comfort and handling, the shaping of the reward signal has been very simple.

When training a neural network, a the hyper-parameters need to be specified. Hyper-parameters are parameters that affect the learning and are set before the learning process begins. The choice of hyper-parameters were inspired by [12] and are shown in Table 3.1. A couple of different setups were investigated, but did not show satisfactory results.

Table 3.1: Hyper-parameters for the DDPG.

Description	Value
Actor network learning rate	0.0001
Critic network learning rate	0.001
Discount factor	0.99
Update rate of target networks	0.001
Buffer size	1000000
Minibatch size	64

3.3.3 Quarter-car control

The purpose of training a neural network controller for a quarter-car environment was to get an indication for how the DDPG algorithm would work in a wheel suspension environment. Furthermore, since a quarter-car model implies a much simpler environment, customization of the learning algorithm and the controller is easier and more time efficient. Customization settings and results from the quarter-car model could then be used in the full-car model.

A quarter-car controller was trained with the DDPG algorithm on the state space model presented in section 3.1. Thus, the control signal decided by the controller was the bounded damping coefficient of the semi-active damper. The observation state vector for the controller consisted of the output from the plant (3.4), and the road preview data

$$\mathbf{y}_{obs} = \begin{bmatrix} \Delta z_s & \Delta z_u & \dot{z}_s & \dot{z}_u & \ddot{z}_s & \ddot{z}_u & \Delta z_{r_1} & \dots & \Delta z_{r_N} \end{bmatrix}^T \tag{3.22}$$

Here, every preview state, $\Delta z_{r_j}\ \forall\ j \in [1,\ N]$, represents the height difference between the current height of the road and a future height of the road.

The architecture of the actor and the critic network that was used for the quarter-car is similar to what was used in [12], both including two hidden layers with rectified linear unit (ReLU) functions, and batch normalization. Also, the output layer of the actor network is a tanh() layer. The critic network takes both observation states and actions as input. However, the actions are not included until the second hidden layer. This is a design that experimentally has shown to work well [49].

The controller was trained on a short road, running over it for several times. Different road preview setups were used, as well as different velocities of the car and different setups of neural networks. The reward function, used for learning, was set to depend on the acceleration and jerk of the body, $\ddot{z}_s$ and $\dddot{z}_s$ respectively, as

$$R = -|\ddot{z}_s| - 0.003 \cdot |\dddot{z}_s| \tag{3.23}$$

The small jerk term was included in the reward function to compensate for the fact that the simple model allows to change the damping coefficient to any value within the bounds instantly. Hence, small oscillations of the body acceleration could be

prevented. The reward is mainly focusing on reducing the body acceleration, which is a common measurement for wheel suspension.

3.3.4 Full-car control

The DDPG algorithm was also applied to the full-car model presented in section 3.2. Since the model includes four semi-active dampers, the controller had four actions, the damper control signals $c_{d_i},\ i = 1, 2, 3, 4$. These signals have been described previously in section 3.2.1. The states observed by the controller consist of the output vector from the full-car plant (3.19) and the road preview data as in (3.21). The preview vector is defined as

$$\mathbf{z}_r = \begin{bmatrix} \Delta z_{r_{1,1}} & \Delta z_{r_{1,2}} & \cdots & \Delta z_{r_{i,j}} & \cdots & \Delta z_{r_{4,N-1}} & \Delta z_{r_{4,N}} \end{bmatrix}^T \tag{3.24}$$

where $i \in [1,\ 4]$ defines the wheel number and $j \in [1,\ N]$ defines the preview step number. Similarly as to the quarter-car, $z_{r_{i,j}}$ represents the height difference between the current height of the road for the current wheel and a future height of the road for the current wheel. The preview length and density was experimentally decided. During experiments, a preview of 15 steps for each wheel, that together covered just over 0.2 s of the road in front of the wheels showed good results. A reasonable time considering the response time of the dampers. However, for implementation simplicity, the preview was defined in distance rather than time ahead of the vehicle. Due to this, the preview setup in terms of preview distance and density was needed to be changed for different vehicle velocities.

Just as for the quarter-car, both the actor and the critic network was set to include two fully connected layers with ReLU activation functions, batch normalization and tanh() layers as output layers. The only difference was the size of the networks. Because of the much larger state and action space, deeper networks were used.

Full-car controllers were trained both episodically and continuously, i.e. running over a short road for several times and running over a very long road for just one time. Different road preview setups were used, as well as different hyper-parameters, different velocities of the car and different neural network settings. The reward was defined as the negative sum of the accelerations (roll, pitch and heave).

$$R = -|\ddot{\varphi}| - |\ddot{\theta}| - |\ddot{z}_s| \tag{3.25}$$

The evaluations of the trained controllers were made both using the training environment and IPG CarMaker. When evaluated in the training environment, the controller is compared to a passive reference. IPG CarMaker includes a more complex model of the full-car, as well as the opportunity to compare the trained controllers to a couple of other controllers. The controllers used for comparison is a *Continuously Controlled Damping controller* (CCD) and an improved version of the nonlinear MPC proposed by [3]. CCD is a control method for semi-active suspension without preview that is used in some of today's cars.

4
Results

This chapter presents training results using the DDPG. Furthermore, simulated performance results for controllers trained with the DDPG, are presented. First, results using the quarter-car model are presented, and then the results using the full-car model are presented. The quarter-car controller is just evaluated on the same model that it is trained on, and is compared with a quarter-car with passive suspension. The full-car controller is evaluated both on the model used for training, but also in IPG CarMaker, with a more complex vehicle model. The performance in IPG CarMaker is also compared with an MPC and a CCD controller.

4.1 Quarter-car control

The quarter-car model used for training and evaluation is described in section 3.1. The control approach for the quarter car is described in 3.3.4. Furthermore, the actor network and critic network used for training the controller are shown in Table 4.1 and 4.2, respectively.

Table 4.1: Specification of the actor network used for the quarter-car model.

Description	Number of neurons	Activation function
Fully connected hidden layer	300	ReLU
Batch normalization	-	-
Fully connected hidden layer	200	ReLU
Batch normalization	-	-
Output layer	1	tanh

The neural network controller was trained by letting the quarter-car model run over a road profile for several times (several episodes). In 4.1, the learning curve is shown for one training session. The figure shows how the return, i.e. the sum of the rewards for one episode, is increasing during training. For this session, the quarter-car ran over a 27 m long road profile with bumps (later shown in Figure 4.2) for 1000 times. The velocity of the quarter-car was 60 kph and the control frequency was 500 Hz. Furthermore, controller had access to 1 m of road preview, with a data density of 0.1 m, giving a total of 10 preview steps. The training time of the session was a few

Table 4.2: Specification of the critic network used for the quarter-car contrller.

Description	Number of neurons	Activation function
Fully connected hidden layer	400	ReLU
Batch normalization	-	-
Fully connected hidden layer	300	ReLU
Batch normalization	-	-
Output layer	1	linear

hours on a regular laptop.

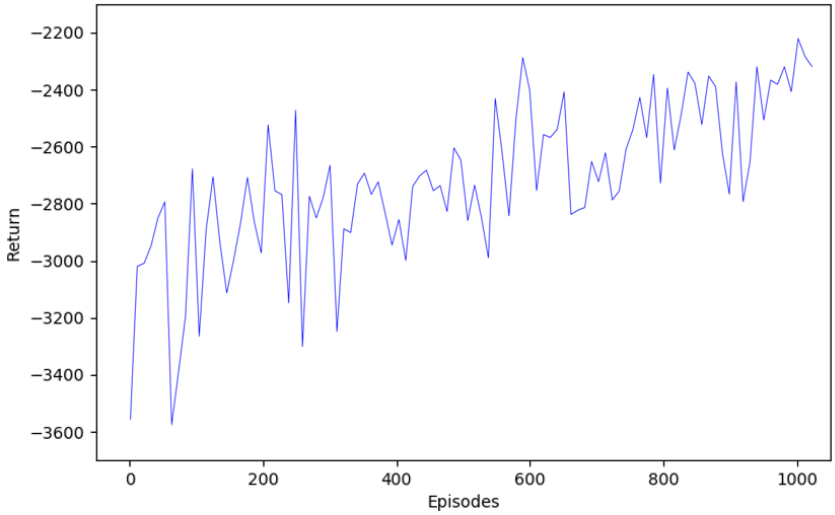


Figure 4.1: Return during training of quarter-car controller.

Evaluation results of the trained controller, compared with a passive damped reference, are shown in Figure 4.2. Here, the road profile is presented together with the body acceleration and the chosen action from the controller. The road profile is the same as the one the controller was trained on. By comparing the body accelerations of the two systems, it is clear that the semi-active system with the trained controller performs significantly better than the passive system. In Table 4.3, the root mean square (RMS) values of the reward and the body acceleration are presented for the semi-active system with the trained controller and the passive system. Generally, the RMS is around 40% better for the trained controller. It can also be observed that the controller has a clear policy, where it chooses low damping coefficients when approaching bumps, and high coefficients to damp rebounds. The passive damper uses a constant damping coefficient of 1500 Ns/m.

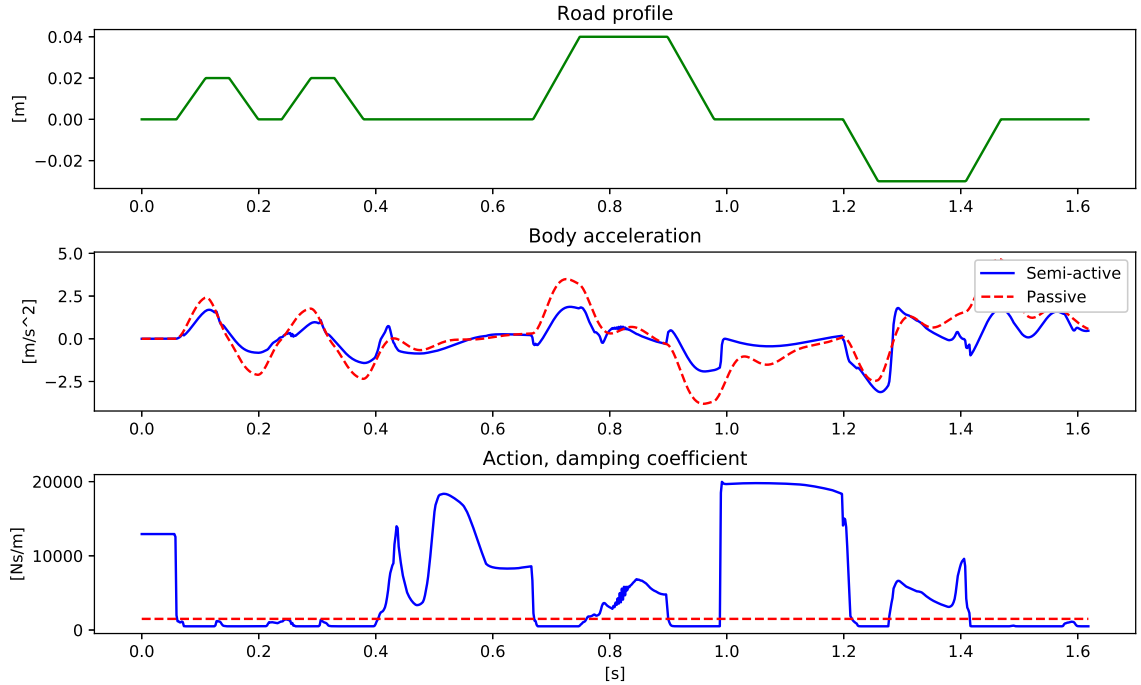


Figure 4.2: Evaluation results of the semi active system with the trained controller and the passive system, on the road used for training.

Table 4.3: RMS performance of the semi active system with the trained controller and the passive system, on the road used for training.

Description	Passive	Semi-active	Improvement in %
RMS reward	1.747	1.078	38.3
RMS body acceleration	1.657	0.958	42.2

The trained controller was also evaluated on two other road profiles which it had not been trained on. In Figure 4.3, the evaluation results of running over an inverted version of the training road with bumps is presented. Hence, the road is similar to the one used for training in terms of frequency and amplitude of the bumps and holes. As shown in the figure, and in Table 4.4, the controller performs very well on the inverted road as well. In fact, it even performs a little better than on the road used for training, compared to the passive system.

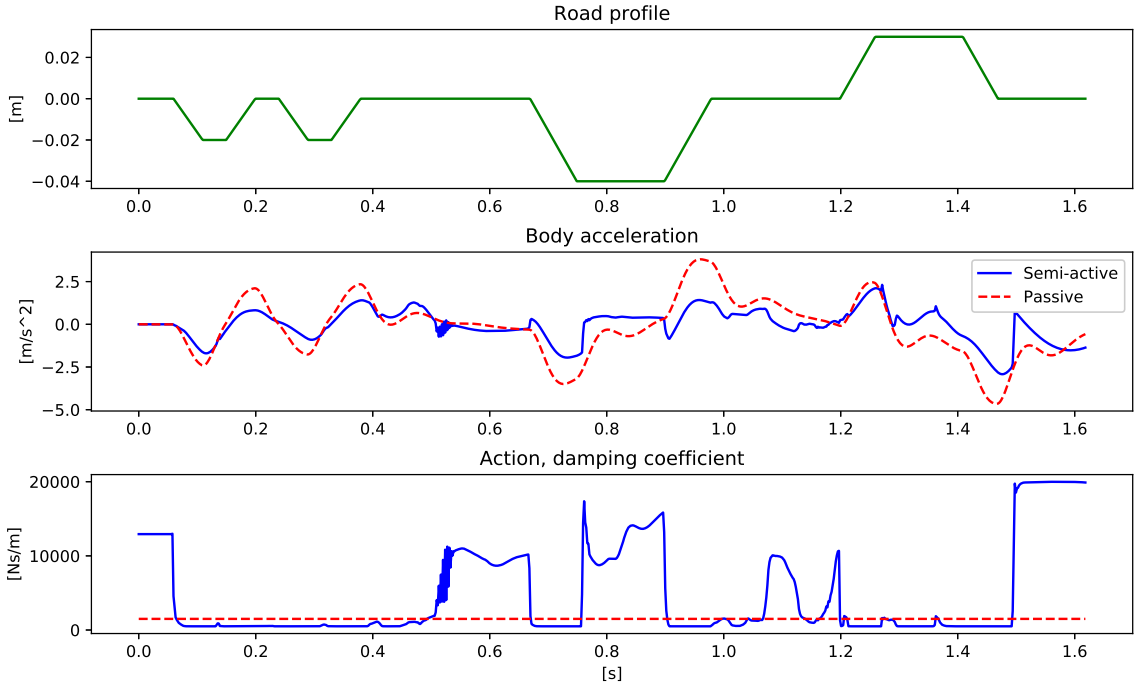


Figure 4.3: Evaluation results of the semi active system with the trained controller and the passive system, on the inverted road used for training.

Table 4.4: RMS performance of the semi active system with the trained controller and the passive system, on the inverted road used for training.

Description	Passive	Semi-active	Improvement in %
RMS reward	1.749	1.059	39.4
RMS body acceleration	1.657	0.939	43.3

The evaluation results of the third road profile are presented in Figure 4.4. The third road is completely different from the two previous road profiles, since it includes high frequency disturbances. By observing the body acceleration comparison, it is clear that the semi-active system with the trained controller performs much worse. This is also supported by Table 4.5. The bad performance is expected since the controller has not been trained on this type of road profile. The results are presented to clarify the effect of episodic training on a specific road.

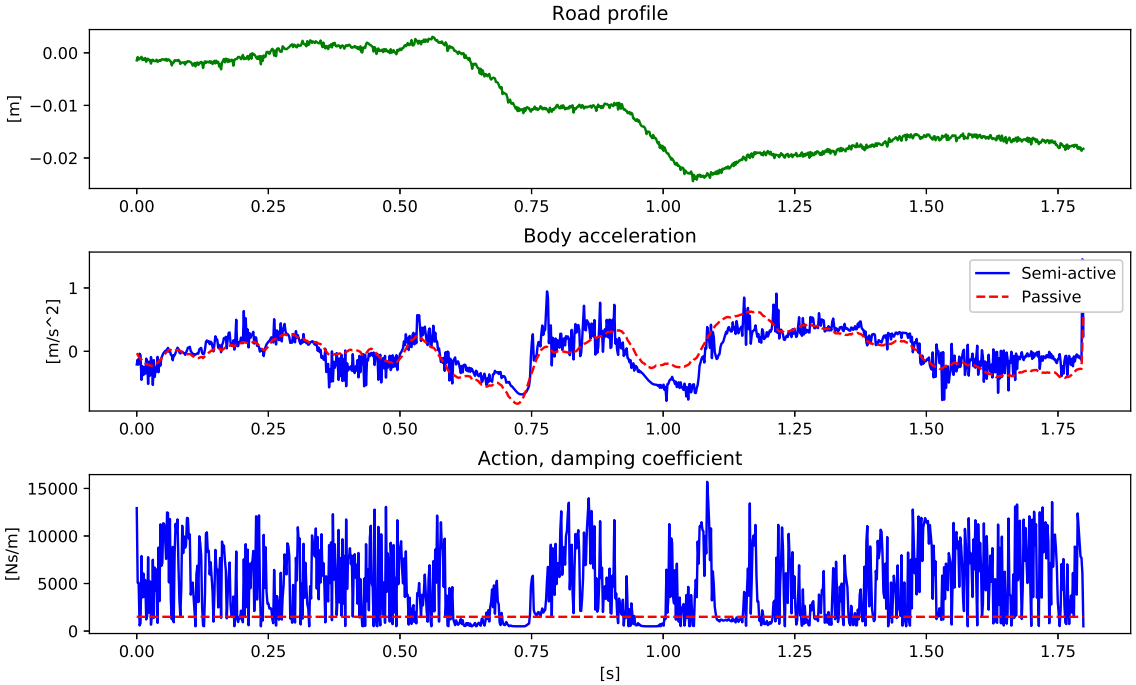


Figure 4.4: Evaluation results of the semi active system with the trained controller and the passive system, on a road with high frequency disturbances.

Table 4.5: RMS performance of the semi active system with the trained controller and the passive system, on a road with high frequency disturbances.

Description	Passive	Semi-active	Worsening in %
RMS reward	0.313	0.660	111.1
RMS body acceleration	0.295	0.311	6.4

The results of training a neural network controller, with the DDPG in the quarter-car environment, shows that the learning algorithm is capable of successful learning in the environment of interest.

4.2 Full-car control

The full-car model used for training and evaluation is described in section 3.2 and the control approach is described in section 3.3.4. Several different neural network controllers were trained for different road profiles and settings in terms of velocity, preview and control frequency. Results are presented for some of these different scenarios.

Due to the significantly larger state space of a full-car model, a deeper actor network was used than for the quarter-car model. The new actor network architecture is presented in table 4.6. As for the critic network, the design used for the quarter-car

model was maintained. Due too the larger actor network and the more complex environment, the training times increased up to many hours and even several days, for some scenarios.

Table 4.6: Specification of the actor network used for the full-car controller.

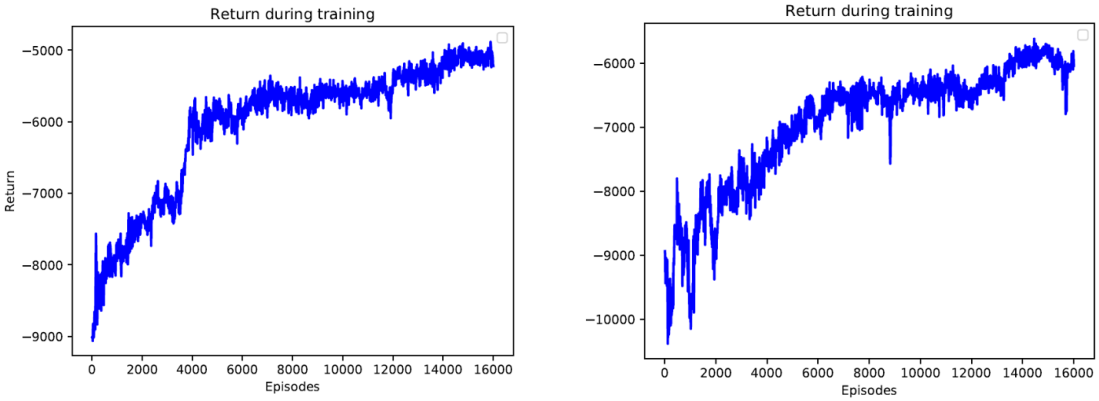
Description	Number of neurons	Activation function
Fully connected hidden layer	1000	ReLU
Batch normalization	-	-
Fully connected hidden layer	800	ReLU
Batch normalization	-	-
Output layer	1	tanh

4.2.1 Scenario 1

In this scenario the neural network specifications shown in Table 4.6 was used. This scenario's road profile was around 100 m long and consisted of a few bumps that hits the left and right wheels of the vehicle simultaneously. The training was done for 16000 episodes and the controller was trained by running over the road in 50 kph, with a control frequency of 100 Hz. Furthermore, to investigate the effect of using road preview, two controllers were trained on this road. One of the controllers had access to 3 m (15 steps per wheel with a density of 0.2 m) of road preview and the other did not have any access to road preview.

4.2.1.1 Effect of using road preview

In Figure 4.5 the achieved return over each episode, for both controllers, are visualized. The return using road preview is significantly better. The controller with preview learns faster and it also reaches a better return at the end of the training session.



(a) Controller with preview

(b) Controller without preview

Figure 4.5: Comparison of the return achieved during training between a controller with road preview and a controller without road preview.

4.2.1.2 Evaluation on training model

The trained controllers were then evaluated on the same model and road which they were trained on. The performance, compared to a full-car system with passive dampers, are presented in Figure 4.6 and in Table 4.7. It can be observed that both controllers perform better than the passive reference in terms of RMS reward, and the controller using preview performs the best. The controller with preview performs very good in terms of heave and pitch acceleration. More specifically, the improvement compared to the passive reference is 16% and 28%, for heave and pitch respectively. Regarding the roll performance, the trained controller performs much worse than the passive reference. The reason for this behavior is because the current road is identical for the left and right side of the car at all time, as can be seen in the top of Figure 4.6. Furthermore, the model of the car is symmetric around the roll-axis. For these reasons, the passive reference does not induce any roll. The trained controller does induce a roll though. This is because the controller has been trained by trying random actions for each wheel, and the learned policy is not entirely optimal. However, it should be noted that the roll acceleration is rather small compared to the heave and pitch accelerations.

Table 4.7: RMS performance comparison between a passive reference, a trained controller without road preview and a trained controller with road preview.

Description	Passive	No preview	3m preview
RMS heave acceleration	1.11	1.17	0.94
RMS pitch acceleration	1.39	1.04	1.00
RMS roll acceleration	0.00	0.22	0.19
RMS reward	2.30	2.18	1.90

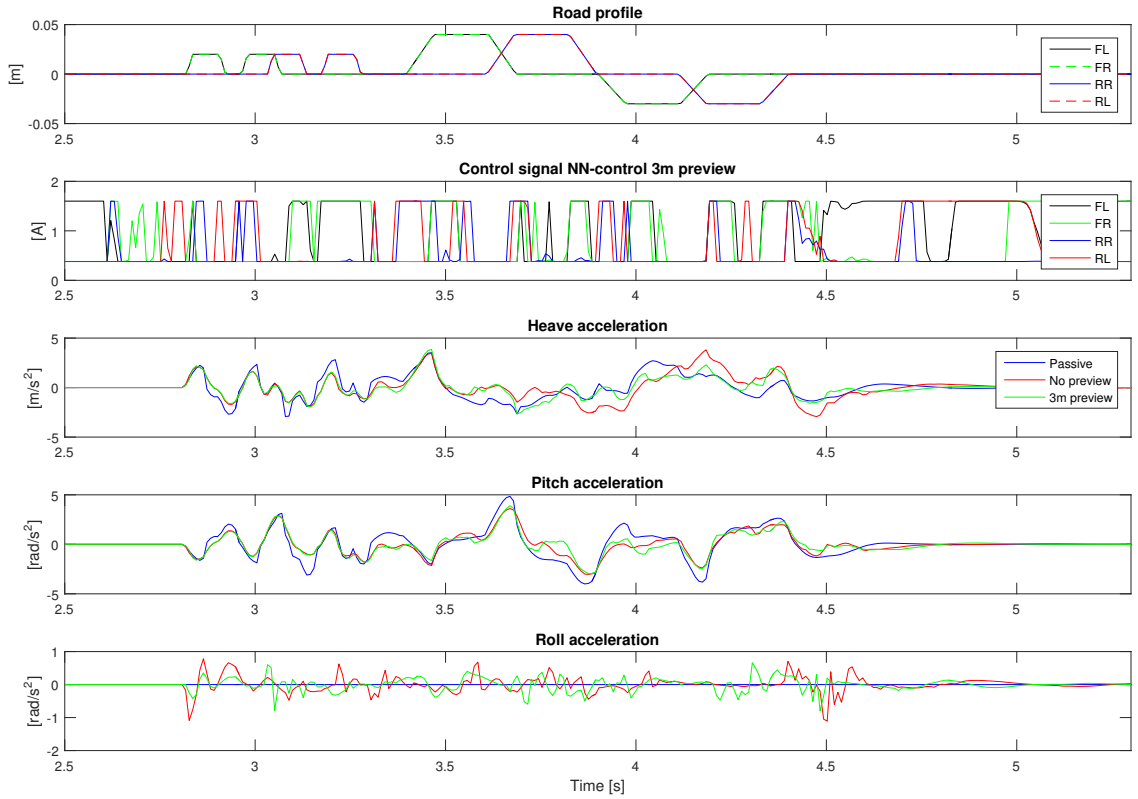


Figure 4.6: Performance comparison between a passive full-car suspension system, a trained controller without road preview data and a trained controller using 3m road preview data.

4.2.1.3 Evaluation in IPG CarMaker

The trained controller with preview was also evaluated in IPG CarMaker, which uses a more detailed model than the one used for training. The results of the evaluation is shown in Figure 4.7 and in Table 4.8. As can be seen in the figure, the road used for evaluation is the same as the NN-controller was trained on. The behaviour of the NN-controller is overall very similar to when it was evaluated on the model used for training (Figure 4.6). Generally, the NN-controller outperforms the CCD but it is not really as good as the MPC. In particular, the heave and pitch accelerations for the NN-controller are 28% and 21% lower than for the CCD. The roll performance for the NN-controller is not very good, and even a bit worse due to the different model in IPG CarMaker. Despite this, the NN-controller performs 11% better, in terms of RMS reward, than the CCD.

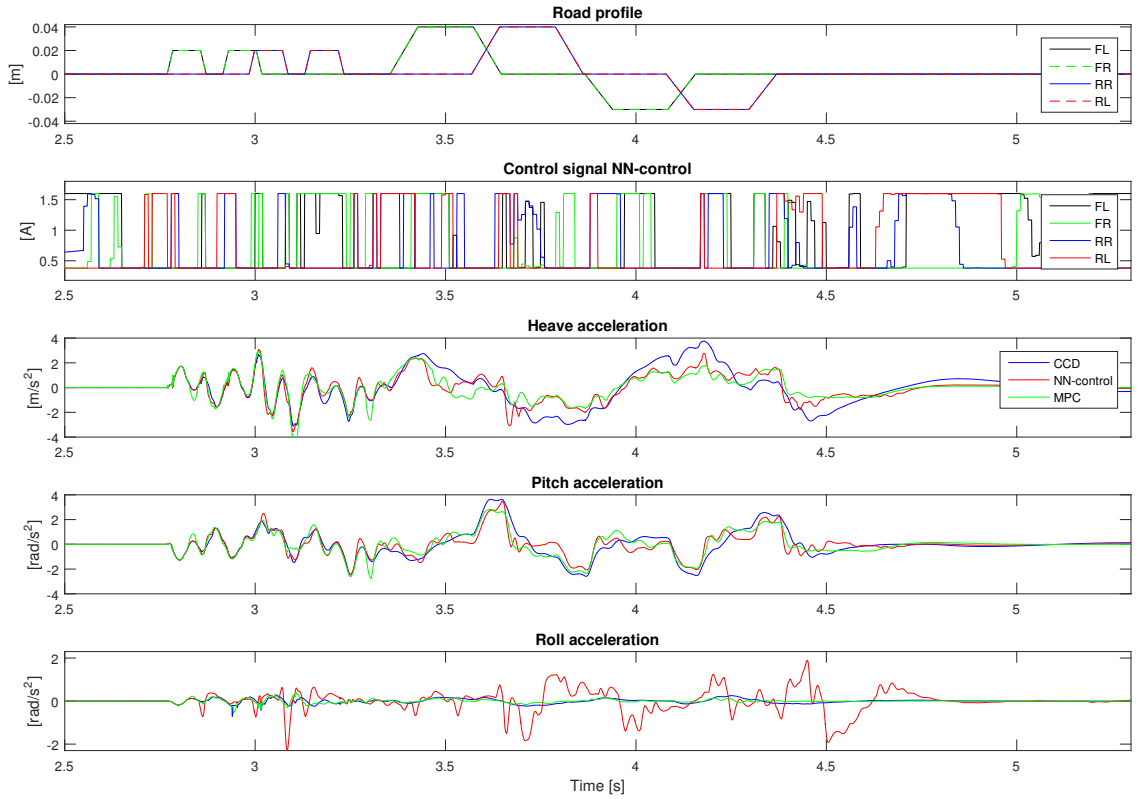


Figure 4.7: Comparison between CCD, MPC and NN-control using IPG CarMaker.

Table 4.8: RMS performance comparison between CCD, MPC and NN-control using IPG CarMaker.

Description	CCD	MPC	NN-control
RMS heave acceleration	0.805	0.543	0.582
RMS pitch acceleration	0.609	0.521	0.484
RMS roll acceleration	0.050	0.044	0.302
RMS reward	1.333	1.009	1.188

The frequency content of the vehicle's body accelerations, using CCD, NN-control and MPC, is shown in Figure 4.8. Except for the roll case, the amplitude is lower for the NN-controller, compared to the CCD, for low frequencies.

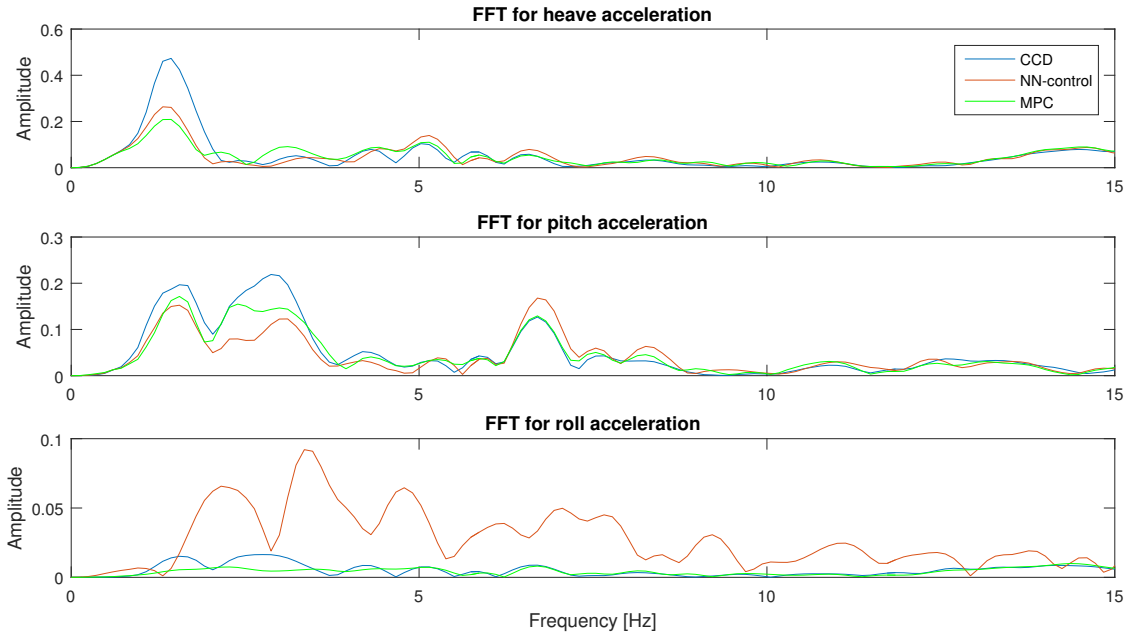


Figure 4.8: Comparison of frequency spectrum for body accelerations between CCD, MPC and NN-control using IPG CarMaker.

4.2.2 Scenario 2

This scenario's road profile was around 300 m long and the training was done for 5000 episodes. The controller was trained by running over the road in 90 kph, with a control frequency of 100 Hz. The NN-controller had access to 5.625 m (15 steps per wheel with a density of 0.375 m) of road preview.

4.2.2.1 Evaluation on training model

The trained controller was then evaluated on the same road and model as it was trained on. The performance, compared to a full-car system with passive dampers, is presented in Figure 4.9 and Table 4.9. It can be seen that the trained controller performs better than the passive for roll and pitch, but slightly worse for heave acceleration. However, the RMS reward for the trained controller show an improvement of more than 10%, compared to the passive system.

Table 4.9: RMS performance comparison between a passive reference and a trained controller using road preview data.

Description	Pasive	NN-control
RMS heave acceleration	0.690	0.709
RMS pitch acceleration	0.410	0.343
RMS roll acceleration	0.832	0.707
RMS reward	1.700	1.5274

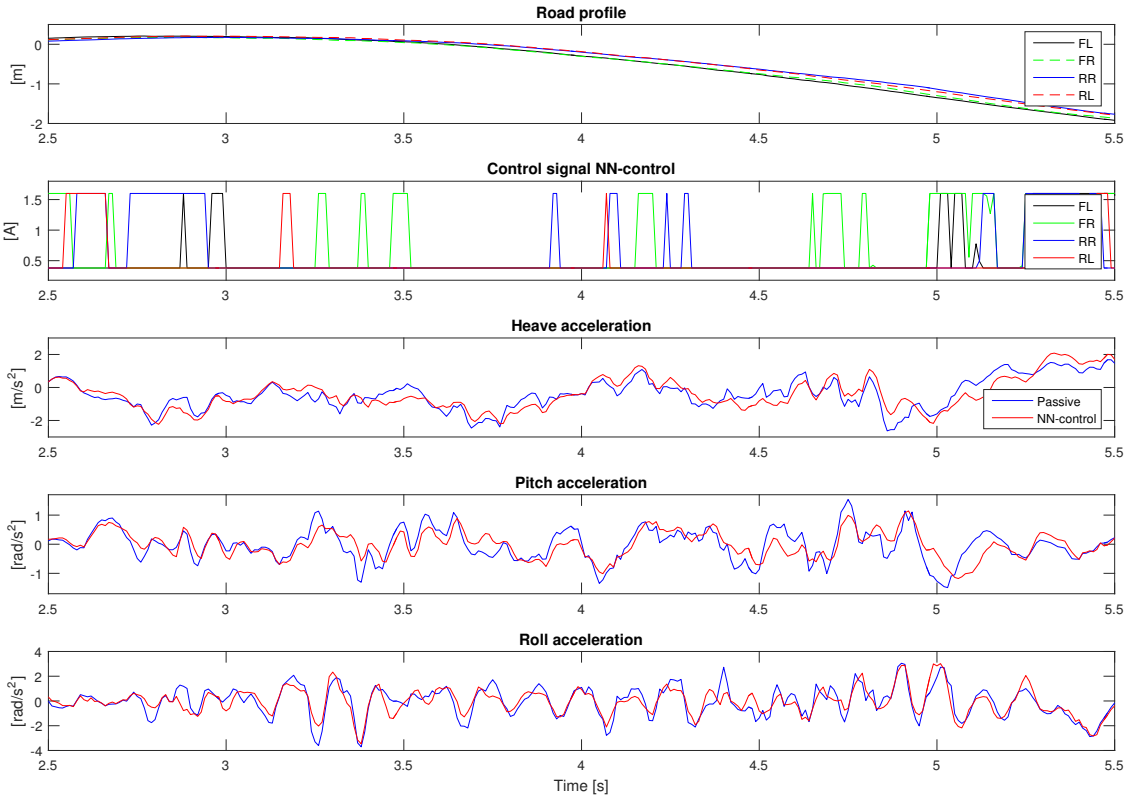


Figure 4.9: Comparison between a passive full-car suspension system and an NN-controller.

4.2.2.2 Evaluation in IPG CarMaker

The trained controller was also evaluated in IPG CarMaker, using the same road as it was trained on. The results from the evaluation are shown in Figure 4.10 and Table 4.10. Here, the controller is, yet again, compared with a CCD and the reference MPC. It can be observed that the amplitudes of the accelerations are similar to the ones in Figure 4.9. However, the acceleration graphs includes much more high frequency content than what achieved when evaluating on the training model. Consequently, the control signal from the NN-controller is varying much more in the CarMaker case. Furthermore, the differences in the environment used for training and evaluation causes difficulties for the NN-controller. Resulting in a 8% worse

performance, in terms of RMS reward, for the NN-controller compared to the CCD.

Table 4.10: RMS performance comparison between CCD, MPC and NN-control using IPG CarMaker.

Description	CCD	MPC	NN-control
RMS heave acceleration	0.864	0.783	0.848
RMS pitch acceleration	0.413	0.379	0.467
RMS roll acceleration	1.050	1.048	1.140
RMS reward	1.981	1.917	2.132

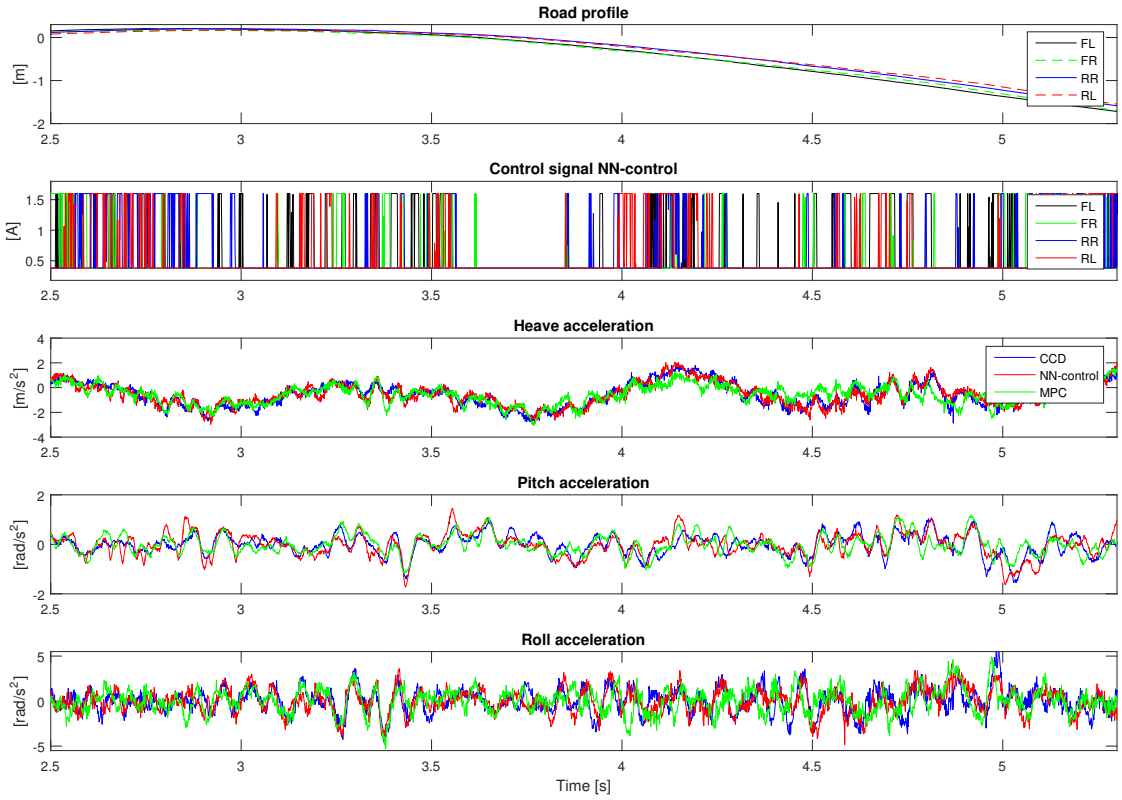


Figure 4.10: Comparison between CCD, MPC and NN-control using IPG CarMaker.

The frequency content of the vehicle's body accelerations, using CCD, NN-control and MPC, is shown in Figure 4.11. The amplitude of most frequencies are quite similar between the different control methods. Generally, the MPC performs best for low frequencies. For high frequencies, it is hard to tell a winner.

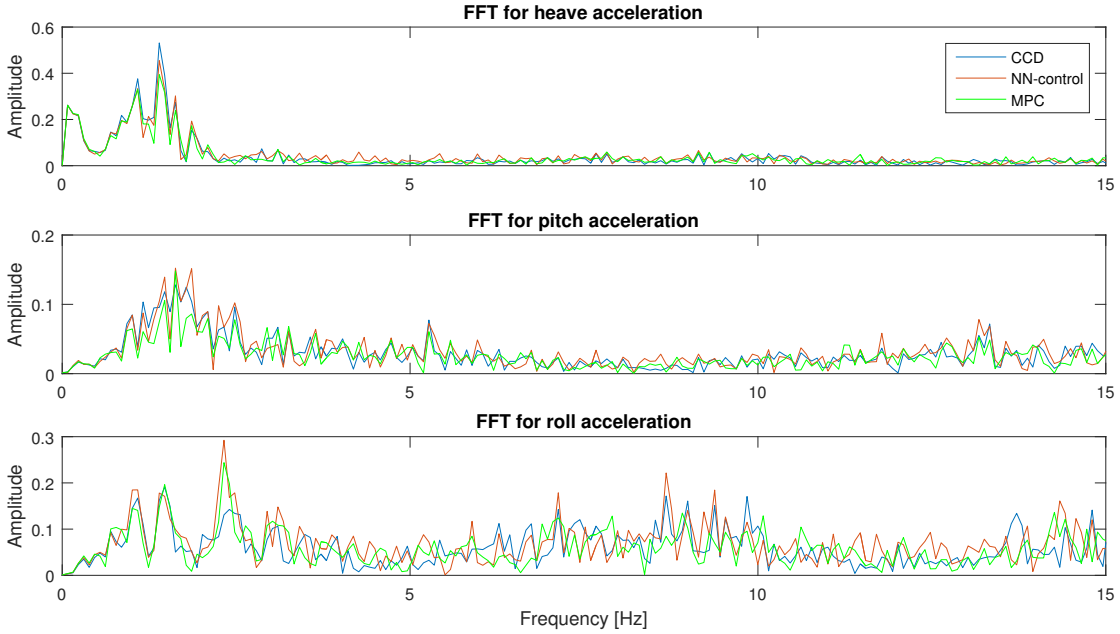


Figure 4.11: Comparison of frequency spectrum for body accelerations between CCD, MPC and NN-control using IPG CarMaker.

4.2.3 Scenario 3

In this scenario the neural network specifications shown in Table 4.6 was used. This scenario's road profile was around 150 m long and the training was done for 2300 episodes. The controller was trained by running over the road in 30 kph, with a control frequency of 100 Hz. The NN-controller had access to 1.87 m (15 steps per wheel with a density of 0.1245 m) of road preview.

4.2.3.1 Evaluation on training model

The trained controller was then evaluated on the same road and model as it was trained on. The performance, compared to a full-car system with passive dampers, is presented in Figure 4.12 and Table 4.11. The trained controller shows good results for all three accelerations. In total, the RMS reward for the trained controller show an improvement of more than 30%, compared to the passive system.

Table 4.11: RMS performance comparison between a passive reference and a trained controller using road preview data.

Description	Passive	NN-control
RMS heave acceleration	0.70	0.45
RMS pitch acceleration	0.39	0.32
RMS roll acceleration	0.13	0.09
RMS reward	1.07	0.75

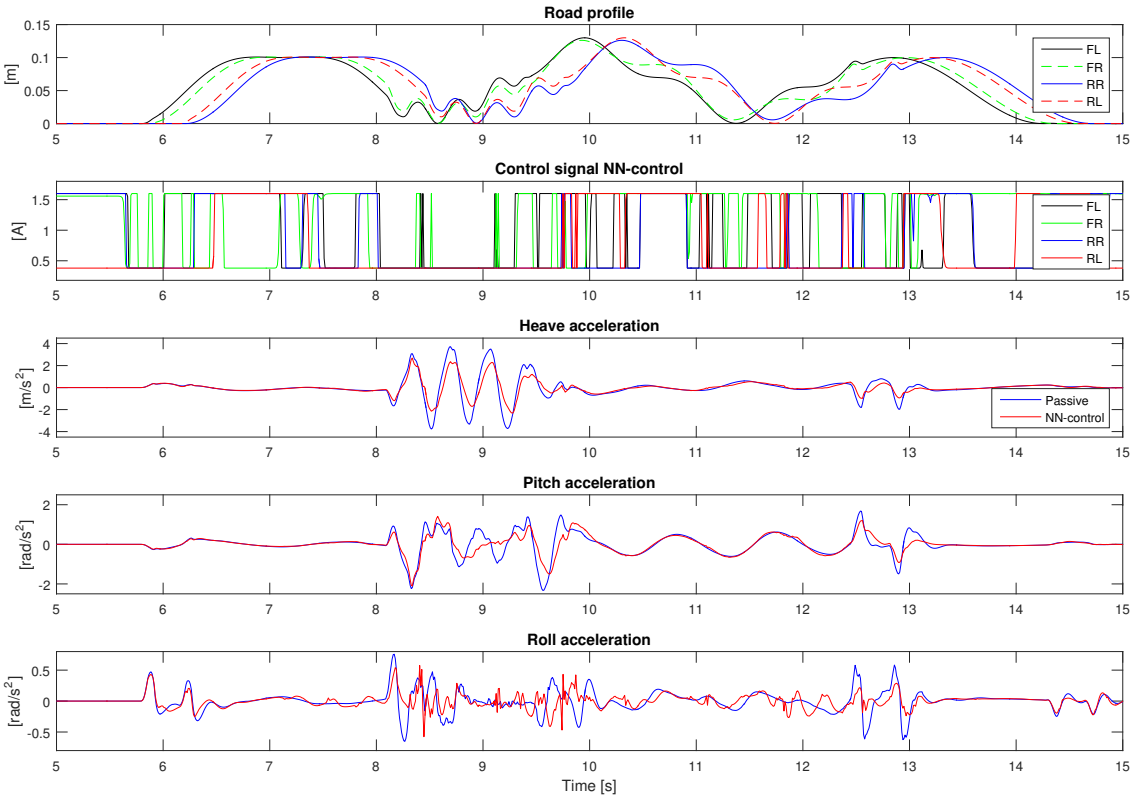


Figure 4.12: Comparison between a passive full-car suspension system and an NN-controller.

4.2.4 Scenario 4

In this scenario, continuous training was investigated and evaluated. A controller was trained on a 90 km long road, including mainly high frequency content. The training time was approximately 60 h. The controller had a control frequency of 500 Hz, in order to be able to handle the high frequency content. The controller was trained by running over the road in 50 kph and had access to 5.625 m (15 steps per wheel with a density of 0.375 m) of road preview.

4.2.4.1 Evaluation on training model

The trained controller was then evaluated on a short road with a similar shape as the training road. The performance, compared to a full-car system with passive dampers, is presented in Figure 4.13 and Table 4.12. Because of the relatively small high frequency noise, the heave and pitch accelerations are quite low for both controllers. The large accelerations occur in the roll spectrum and here the trained controller significantly outperforms the passive reference. For the particular road segment, the improvement is 30% in terms of RMS roll acceleration. Speaking of RMS reward, the trained controller show an improvement of a little more than 19%, compared to the passive system.

Table 4.12: RMS performance comparison between a passive reference and a trained controller using road preview data.

Description	Passive	NN-control
RMS heave acceleration	0.93	0.91
RMS pitch acceleration	0.95	0.92
RMS roll acceleration	2.27	1.59
RMS reward	3.70	3.01

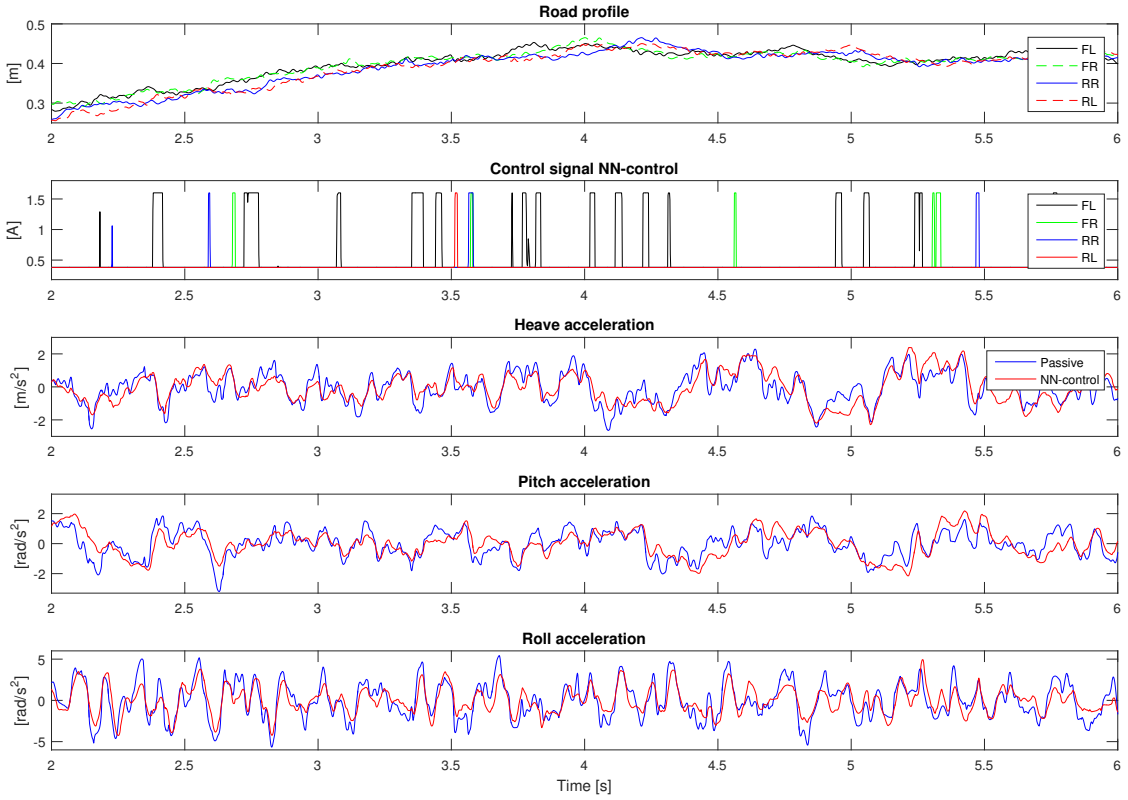


Figure 4.13: Comparison between a passive full-car suspension system and an NN-controller.

4.2.5 Execution time

Calculating the execution time for the controller is important to be able to see if it is fast enough to run in real-time. The execution time is the time it takes from sending all the states as input to the controller/neural network, until achieving a control signal. In Figure 4.14, a histogram presents the distribution of the execution times for the NN-controller, collected from four different evaluations on two different road profiles. The simulations were run on a regular laptop and the controller consisted neural network with the architecture presented in Table 4.6. In the figure, it can be observed that most of the execution times are less than 0.5 ms and almost all execution times are less than 1 ms. This is just a fraction of the time that the

reference MPC needs to compute the control signal. Furthermore, most controllers have been run in 100 Hz in simulation. Given the presented execution times, the proposed controllers would be able to run in real time without problems.

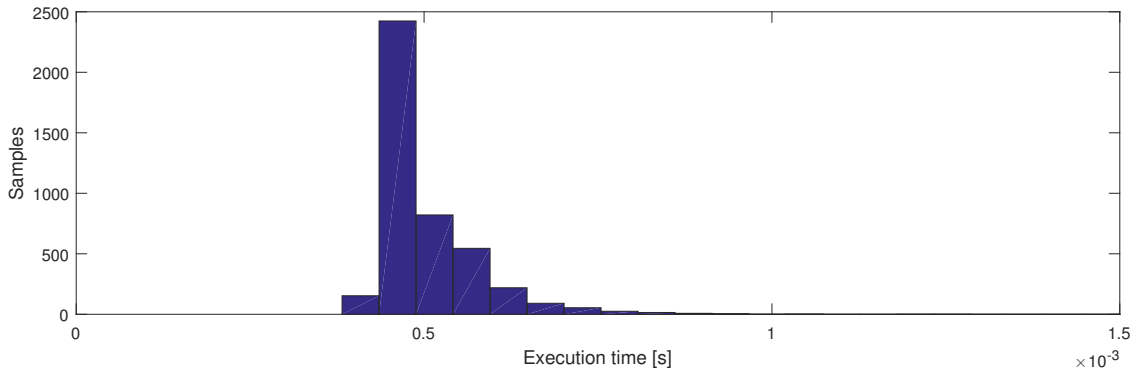


Figure 4.14: Execution time for NN-control.

5
Discussion

The purpose of the thesis was to investigate if a neural network, trained by the DDPG algorithm, could be successful in controlling a full-car semi-active wheel suspension system with access to road preview. To get a first indication of how the DDPG algorithm would perform in a suspension system environment, a simple quarter-car model was designed and used for training. The results from training a neural network, with the DDPG in the quarter-car environment, show that the DDPG successfully can train a controller to perform better than passive suspension in the environment it has been trained on. Thereof, a more complex full-car model was designed to be used for further training. The results from training in the full-car environment shows that a neural network can control a semi-active suspension system, and reach a performance that is significantly better than the performance of a passive system. Furthermore, the neural network controller has also shown better performance than the reactive CCD controller that is used in today's cars. Aside from investigating the performance of a neural network controller for semi-active wheel suspension, the computational complexity was said to be investigated. The proposed controller has a relatively low computational complexity, just a fraction of the complexity of the reference MPC. Finally, the purpose of the thesis was to investigate the effect of using road preview in combination with a neural network controller. The results showed that road preview speeds up training of the neural network, and that a higher performance can be reached.

The results show that episodic training on a road profile implies successful performance on the current road. By running over the same road for several times, the learning converges to a satisfying control policy. Good performance for this task can be reached with rather small neural networks. Hence, a small computational cost can be maintained. Evaluation of the trained neural networks controllers on other roads than the training road, has not been very successful. It was shown that the trained controllers could perform good on roads that had a similar shape, in terms of frequency and magnitude, as the training road. Given very different roads compared to the training road, the neural network controllers could not achieve satisfying results. This behaviour is however somewhat expected, since a trained neural network controller will just perform good given similar states to what it has visited during training. Thus, to attain a single neural network controller that is able to perform well on all kinds of road profiles that can occur in practice, a neural network needs to be trained on a road profile that makes the full-car model visit a much larger part on the entire state space. Further, for a neural network to be able to deliver suitable control signals for such a large state space, a deeper neural

network that is trained for a longer time is probably needed. Because of the low computational cost of the neural network used in this thesis, it is possible to make the neural network much deeper without sacrificing the possibility to run in real time.

The performance of the neural network controllers was evaluated in terms of RMS roll, pitch and heave accelerations. By comparing the results for the different road profiles, it is clear that the learning algorithm develops different policies for different road profiles. The learning algorithm finds what is the most optimal way of maximizing its reward for the current road profile, which can be very different depending on the road. Since the reward is the negative sum of the roll, pitch and heave accelerations, the algorithm will often focus on reducing one or two of the accelerations if it results in a better reward. This behaviour is probably not desired for a controller in a real car, but it can easily be changed by changing the reward function. The performance in terms of different frequencies shows that the neural network controller generally performs good for low frequencies (primary ride) and worse for high frequencies (secondary ride). Whether this behaviour can be changed by changing the reward, or if it is due to something else is unclear.

As mentioned previously, the calculated execution times for the neural network to compute the control signal is just a fraction of the time the reference MPC needs. Further, the execution times for neural networks does not vary with different input sets of data. This is not the case for the MPC, which is very dependent on the computational complexity from the given inputs. Hence, neural networks might be more suitable than MPC to be used in a real-time system, where the inputs varies substantially.

The road preview data that is sent to the controller has not been optimized in terms of preview length and density. A couple of different setups were tried during the process, but it is definitely an area that can be improved further. By increasing the size of the preview data (either by increasing the preview length or the density), the observation state space that the controller needs to map increases rapidly. Hence, even though an increased preview size could improve the performance slightly, but it may not be the optimal way of spending the computational resources that are available.

Regarding the DDPG learning algorithm, it has shown to be successful for learning in the environment of interest. It can learn a successful policy in a continuous state and action space, given an arbitrary reward to represent ride comfort.

5.1 Conclusion

In this thesis, a control approach for semi-active wheel suspension with road preview has been proposed. The control approach is based on an artificial neural network trained with reinforcement learning, more specifically the DDPG algorithm. The

learning method has shown to be successful in both continuous and episodic environments. The approach shows promising results, often able to outperform today's CCD controller for semi-active suspension in simulation. The trained controller does not really match the performance of the reference MPC, but the computational cost of the trained controller is just a fraction. This indicates that the proposed control approach can be further improved by utilizing a deeper neural network.

5.2 Future work

To achieve a controller that is suitable for all kind of road profiles, a neural network needs to be trained on a road that covers a much larger space of the entire state space. In order for the network to collect all information during the training, a deeper network should be considered. Alternatively one could train several different networks and change between the networks depending on the road. Furthermore, the road preview should be optimized in terms of preview length and density. Also, to be able to run on a real car, rotational pre-processing of the road preview data is needed to suit a camera on top of the car.

References

[1] Y. Zhou and S. Chen, "Vehicle ride comfort analysis with whole-body vibration on long-span bridges subjected to crosswind", *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 155, pp. 126 –140, 2016, ISSN: 0167-6105. DOI: https://doi.org/10.1016/j.jweia.2016.05.001.

[2] P. Els, N. Theron, P. Uys, and M. Thoresson, "The ride comfort vs. handling compromise for off-road vehicles", *Journal of Terramechanics*, vol. 44, no. 4, pp. 303 –317, 2007, ISSN: 0022-4898. DOI: https://doi.org/10.1016/j.jterra.2007.05.001.

[3] F. Kjellberg and S. Sundell, "Real-time nonlinear model predictive control for semi-active suspension with road preview", M.S. thesis, Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden, 2018. [Online]. Available: http://publications.lib.chalmers.se/records/fulltext/255751/255751.pdf.

[4] M. Willis, G. Montague, C. D. Massimo, M. Tham, and A. Morris, "Artificial neural networks in process estimation and control", *Automatica*, vol. 28, no. 6, pp. 1181 –1187, 1992, ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(92)90059-O.

[5] J. MacMurray and D. Himmelblau, "Identification of a packed distillation column for control via artificial neural networks", in *1993 American Control Conference*, 1993, pp. 1455–1459. DOI: 10.23919/ACC.1993.4793112.

[6] J. Thibault and B. Grandjean, "Neural networks in process control - a survey", in *Advanced Control of Chemical Processes 1991*, ser. IFAC Symposia Series, K. NAJIM and E. DUFOUR, Eds., Amsterdam: Pergamon, 1992, pp. 251 – 260, ISBN: 978-0-08-041267-2. DOI: https://doi.org/10.1016/B978-0-08-041267-2.50046-2.

[7] M. T. Hagan, H. B. Demuth, and O. D. Jesús, "An introduction to the use of neural networks in control systems", *International Journal of Robust and Nonlinear Control*, vol. 12, no. 11, pp. 959–985, 2002. DOI: 10.1002/rnc.727.

[8] N. Casas, "Deep deterministic policy gradient for urban traffic light control", *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1703.09035.

[9] A. Kumar, N. Paul, and S. N. Omkar, "Bipedal walking robot using deep deterministic policy gradient", *CoRR*, 2018. [Online]. Available: http://arxiv.org/abs/1807.05924.

[10] G. Cano Lopes, M. Ferreira, A. Simões, and E. Luna Colombini, "Intelligent control of a quadrotor with proximal policy optimization reinforcement learning", Nov. 2018, pp. 503–508. DOI: 10.1109/LARS/SBR/WRE.2018.00094.

[11] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning", *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017, ISSN: 2377-3766. DOI: 10.1109/LRA.2017.2720851.

[12] 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", *CoRR*, vol. abs/1509.02971, 2015. [Online]. Available: http://arxiv.org/abs/1509.02971.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning", *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: https://arxiv.org/abs/1312.5602.

[14] M. Tomizuka, ""optimum linear preview control with application to vehicle suspension"—revisited", *Journal of Dynamic Systems Measurement and Control-transactions of The Asme - J DYN SYST MEAS CONTR*, vol. 98, Sep. 1976. DOI: 10.1115/1.3427040.

[15] A G. Thompson, B R. Davis, and C E. M. Pearce, "An optimal linear active suspension with finite road preview", in *1980 Automotive Engineering Congress and Exposition*, Feb. 1980. DOI: 10.4271/800520.

[16] G. Prokop and R. Sharp, "Performance enhancement of limited-bandwidth active automotive suspensions by road preview", *Control Theory and Applications, IEE Proceedings*, vol. 142, pp. 140 –148, Apr. 1995. DOI: 10.1049/ip-cta:19951772.

[17] R. Mehra, J. Amin, K. Hedrick, C Osorio, and S Gopalasamy, "Active suspension using preview information and model predictive control", Nov. 1997, pp. 860 –865. DOI: 10.1109/CCA.1997.627769.

[18] C. Göhrle, A. Schindler, A. Wagner, and O. Sawodny, "Model predictive control of semi-active and active suspension systems with available road preview", pp. 1499–1504, 2013. DOI: 10.23919/ECC.2013.6669185.

[19] C. Göhrle, A. Schindler, A. Wagner, and O. Sawodny, "Road profile estimation and preview control for low-bandwidth active suspension systems", *IEEE/ASME Transactions on Mechatronics*, vol. 20, no. 5, pp. 2299–2310, 2015, ISSN: 1083-4435. DOI: 10.1109/TMECH.2014.2375336.

[20] M. Q. Nguyen, M. Canale, O. Sename, and L. Dugard, "A model predictive control approach for semi-active suspension control problem of a full car", in *2016 IEEE 55th Conference on Decision and Control (CDC)*, 2016, pp. 721–726. DOI: 10.1109/CDC.2016.7798353.

[21] M. Howell, G. Frost, T. Gordon, and Q. Wu, "Continuous action reinforcement learning applied to vehicle suspension control", *Mechatronics*, vol. 7, no. 3, pp. 263 –276, 1997, ISSN: 0957-4158. DOI: 10.1016/S0957-4158(97)00003-2.

[22] S. Tognetti, S. M. Savaresi, C. Spelta, and M. Restelli, "Batch reinforcement learning for semi-active suspension control", in *2009 IEEE Control Applications, (CCA) Intelligent Control, (ISIC)*, Saint Petersburg, Russia, 2009. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5281070.

[23] I. Bucak and H R. Öz, "Vibration control of a nonlinear quarter-car active suspension system by reinforcement learning", *International Journal of Systems Science - IJSySc*, vol. 43, pp. 1177–1190, Jun. 2012. DOI: 10.1080/00207721.2010.549576.

[24] A. R. Bhise, R. G. Desai, R. N. Yerrawar, A. Mitra, and R. R. Arakerimath, "Comparison between passive and semi-active suspension system using matlab/simulink and serially concatenated codes", *IOSR Journal of Mechanical and Civil Engineering*, vol. 13, no. 4, pp. 01–06, 2016. [Online]. Available: http://iosrjournals.org/iosr-jmce/papers/vol13-issue4/Version-1/A1304010106.pdf.

[25] S. A. Hassan, "Fundamental studies of passive, active and semi-active automotive suspension systems", Ph.D. dissertation, Department of Mechanical Engineering, The University of Leeds, Leeds, UK, 1986. [Online]. Available: https://core.ac.uk/download/pdf/9257269.pdf.

[26] M. A. Nekoui and P. Hadavi, "Optimal control of an active suspension system", in *Proceedings of 14th International Power Electronics and Motion Control Conference EPE-PEMC 2010*, Ohrid, Macedonia, 2010.

[27] A. F. Mohd Riduan, N. Tamaldin, A. Sudrajat, and F. Ahmad, "Review on active suspension system", *SHS Web of Conferences*, vol. 49, p. 02 008, 2018. DOI: 10.1051/shsconf/20184902008.

[28] N. Ito, "Semi-active suspension", in *KYB Technical review*, T. Tezuka *et al*, Ed., Tokyo, Japan: Shobi Printing Co., 2017, pp. 31 –32. [Online]. Available: https://www.kyb.co.jp/english/technical_report/data/no55e/all.pdf.

[29] M. Witters and J. Swevers, "Identification of noe models for a continuously variable semi-active damper", *23rd International Conference on Noise and Vibration Engineering 2008, ISMA 2008*, vol. 7, Jan. 2008.

[30] J. Reimpell, H. Stoll, and J. W. Betzler, in *The Automotive Chassis*, Second Edition, Oxford: Butterworth-Heinemann, 2001, ISBN: 978-0-7506-5054-0.

[31] K. Wolff, J. Hoppermans, and R. Kraaijeveld, "Objective evaluation of subjective driving impressions", *JSAE Annual Cong*, Yokohama, Japan, 2008. [Online]. Available: https://www.fev.com/fileadmin/user_upload/Media/TechnicalPublications/Chassis/V0815_FISITA2008_Objective_Evaluation_F20080319-Paper_01.pdf.

[32] R. Dixon, "Back to basics – ride quality and comfort", *Motorworld Media*, Oct. 26, 2014. [Online]. Available: http://www.dieselcar.com/features/doctor/back-to-basics-ride-quality-and-comfort/.

[33] A. Kruczek and A. Stribrsky, "A full-car model for active suspension - some practical aspects", in *Proceedings of the IEEE International Conference on Mechatronics, 2004. ICM '04.*, 2004, pp. 41–45. DOI: 10.1109/ICMECH.2004.1364409.

[34] S. M. Savaresi, C. Poussot-Vassal, C. Spelta, O. Sename, and L. Dugard, "Chapter 3 - suspension oriented vehicle models", in *Semi-Active Suspension Control Design for Vehicles*, Boston, USA: Butterworth-Heinemann, 2010, pp. 41 –69, ISBN: 978-0-08-096678-6. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780080966786000031.

[35] R. N. Jazar, *Vehicle Dynamics: Theory and Applications*. Riverdale, NY: Springer Science+Business Media, 2008, ISBN: 978-0-387-74243-4.

[36] J. D. Seo, "My journey to reinforcement learning — part 0: Introduction", *Towards Data Science*, Apr. 7, 2018. [Online]. Available: https://towardsdatascience.com/my-journey-to-reinforcement-learning-part-0-introduction-1e3aec1ee5bf.

[37] ADL, "A brief introduction to reinforcement learning", *Medium*, Aug. 27, 2018. [Online]. Available: https://medium.freecodecamp.org/a-brief-introduction-to-reinforcement-learning-7799af5840db.

[38] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, Massachusetts: The MIT Press, 2018, ISBN: 978-0-26-203924-6.

[39] J. Hui, "Rl—model-based reinforcement learning", *Medium*, Sep. 25, 2018. [Online]. Available: https://medium.com/@jonathan_hui/rl-model-based-reinforcement-learning-3c2b6f0aa323.

[40] T. Xia and Z. Han, "Path planning using reinforcement learning and objective data", M.S. thesis, Department of Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden, 2017. [Online]. Available: http://publications.lib.chalmers.se/records/fulltext/256451/256451.pdf.

[41] C. J.C. H. Watkins and P. Dayan, "Q-learning", *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992, ISSN: 1573-0565. DOI: 10.1007/BF00992698.

[42] A. Juliani, "Simple reinforcement learning with tensorflow part 4: Deep q-networks and beyond", *Medium*, Sep. 2, 2016. [Online]. Available: https://medium.com/@awjuliani/simple-reinforcement-learning-with-tensorflow-part-4-deep-q-networks-and-beyond-8438a3e2b8df.

[43] T. Simonini, "An introduction to policy gradients with cartpole and doom", *Medium*, May 9, 2018. [Online]. Available: https://medium.freecodecamp.org/an-introduction-to-policy-gradients-with-cartpole-and-doom-495b5ef2207f.

[44] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms", *31st International Conference on Machine Learning, ICML 2014*, vol. 1, Jun. 2014.

[45] E. Bibbona, G. Panfilo, and P. Tavella, "The ornstein–uhlenbeck process as a model of a low pass filtered white noise", *Metrologia*, vol. 45, Dec. 2008. DOI: 10.1088/0026-1394/45/6/S17.

[46] K. Mohiuddin, J. Mao, and A. K. Jain, "Artificial neural networks: A tutorial", *Computer*, vol. 29, pp. 31–44, Mar. 1996, ISSN: 0018-9162. DOI: 10.1109/2.485891.

[47] L. Dormehl, "What is an artificial neural network? here's everything you need to know", *Digital Trends*, Jan. 5, 2019. [Online]. Available: https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/.

[48] A. Sharma, "Understanding activation functions in neural networks", *Medium*, Mar. 30, 2017. [Online]. Available: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0.

[49] P. Emami, "Deep deterministic policy gradients in tensorflow", *My summaries of Machine Learning papers and investigations into various topics concerning artificial intelligence*, Aug. 2016. [Online]. Available: https://pemami4911.github.io/blog/2016/08/21/ddpg-rl.html.