Self-Learning Longitudinal Control for On-Road Vehicles

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

der KIT-Fakultät für Elektrotechnik und Informationstechnik
des Karlsruher Instituts für Technologie (KIT)

angenommene

Dissertation

von

Luca Puccetti, M. Sc.

aus München (Bayern)

Tag der mündlichen Prüfung: 17. Februar 2023
Erster Gutachter: Prof. Dr.-Ing. Sören Hohmann
Zweiter Gutachter: Prof. Dr.-Ing. Jürgen Adamy

Abstract

Advanced driver assistance systems are an important selling point for vehicles, but they require high development effort. For longitudinal control, a common foundation for driver assistance systems, tuning requires time and effort to balance accuracy and passenger comfort. Reinforcement learning is a promising approach for automating this, but until now has mostly been applied to simulated examples that provided ideal conditions and nearly infinite training time.

Among the major challenges for applying reinforcement learning to longitudinal control in a real vehicle, there are partially observed dynamics and tracking control. In order to be applicable in real-world applications, the learning process for the optimal controller needs to converge within minutes. On top of that, the target speed can change arbitrarily in this use case, which is challenging to solve with reinforcement learning.

This work proposes two computationally lightweight reinforcement learning algorithms that address these issues. First, a model-free algorithm is introduced. It is based on the actor-critic architecture that employs a special structure in the state-action value function approximator to handle the partially observed system. In addition, it is proposed to learn a feedforward speed tracking controller that uses a projection and training data manipulation.

A second proposal within this work is a model-based algorithm that is based on policy search. It is accompanied with an automated inversion-based feedforward controller design method.

The proposed algorithms are compared across a series of scenarios in a real vehicle and learning on-line, i.e. while driving closed-loop. While the algorithms react slightly different to choices of exploration noise, both learn robustly and quickly, are able to adapt to different points of operation, e.g. speeds and gears even when faced with disturbances during training. To the author’s knowledge, this is the first application of reinforcement learning that successfully learns on-line in a real vehicle.

Zusammenfassung

Fahrerassistenzsysteme (Advanced Driver Assistance Systems) sind ein wichtiges Verkaufsargument für PKWs, fordern jedoch hohe Entwicklungskosten. Insbesondere die Parametrierung für Längsregelung, die einen wichtigen Baustein für Fahrerassistenzsysteme darstellt, benötigt viel Zeit und Geld, um die richtige Balance zwischen Insassenkomfort und Regelgüte zu treffen. Reinforcement Learning scheint ein vielversprechender Ansatz zu sein, um dies zu automatisieren. Diese Klasse von Algorithmen wurde bislang allerdings vorwiegend auf simulierte Aufgaben angewendet, die unter idealen Bedingungen stattfinden und nahezu unbegrenzte Trainingszeit ermöglichen.

Unter den größten Herausforderungen für die Anwendung von Reinforcement Learning in einem realen Fahrzeug sind Trajektorienfolgeregelung und unvollständige Zustandsinformationen aufgrund von nur teilweise beobachteter Dynamik. Darüber hinaus muss ein Algorithmus, der in realen Systemen angewandt wird, innerhalb von Minuten zu einem Ergebnis kommen. Außerdem kann das Regelziel sich während der Laufzeit beliebig ändern, was eine zusätzliche Schwierigkeit für Reinforcement Learning Methoden darstellt.

Diese Arbeit stellt zwei Algorithmen vor, die wenig Rechenleistung benötigen und diese Hürden überwinden. Einerseits wird ein modellfreier Reinforcement Learning Ansatz vorgeschlagen, der auf der Actor-Critic-Architektur basiert und eine spezielle Struktur in der Zustandsaktionswertfunktion verwendet, um mit teilweise beobachteten Systemen eingesetzt werden zu können. Um eine Vorsteuerung zu lernen, wird ein Regler vorgeschlagen, der sich auf eine Projektion und Trainingsdatenmanipulation stützt.

Andererseits wird ein modellbasierter Algorithmus vorgeschlagen, der auf Policy Search basiert. Diesem wird eine automatisierte Entwurfsmethode für eine inversionsbasierte Vorsteuerung zur Seite gestellt.

Die vorgeschlagenen Algorithmen werden in einer Reihe von Szenarien verglichen, in denen sie online, d.h. während der Fahrt und bei geschlossenem

Regelkreis, in einem realen Fahrzeug lernen. Obwohl die Algorithmen etwas unterschiedlich auf verschiedene Randbedingungen reagieren, lernen beide robust und zügig und sind in der Lage, sich an verschiedene Betriebspunkte, wie zum Beispiel Geschwindigkeiten und Gänge, anzupassen, auch wenn Störungen während des Trainings einwirken. Nach bestem Wissen des Autors ist dies die erste erfolgreiche Anwendung eines Reinforcement Learning Algorithmus, der online in einem realen Fahrzeug lernt.

Contents

List of Figures

List of Tables

Acronyms

ADAM adaptive moment estimation.

ARX autoregressive model with external inputs.

COVID corona virus disease.

FIR finite impulse response.

GPS global positioning system.

Lidar light detection and ranging.

LM Levenberg-Marquardt.

MB model based.

MDL minimum description length.

MF model free.

PID proportional, integral and derivative.

RL reinforcement learning.

RNN recurrent neural network.

TD temporal difference.

WLTP worldwide harmonized light vehicle test procedure.

ZPETC zero phase error tracking control.

1. Introduction

1.1. Motivation

Do you still remember your first driving lesson? Back then, even basic tasks like keeping a constant speed seemed complicated. Around 2016, experts predicted that adolescents would soon be spared even from these challenges. They expected driver's licences to become obsolete within a few years since autonomous vehicles on public roads would become mainstream. While this excitement has cooled off considerably [135] and fully autonomous driving has entered the "trough of disillusionment" [45], advanced driver assistance systems are steadily entering mass markets. On the one hand this is due to regulatory demands [163], on the other hand many new car buyers choose these systems as an option, especially in rapidly growing markets such as China [116]. Forecasts assume that 85% of cars built in 2025 will be equipped with advanced driver assistance systems [134]. This trend aims not only to increase driving comfort, but also to enhance safety [37].

In the long run, these developments aim at gradually making human supervision superfluous [3], even though this goal is still more than a decade away according to experts [135].

Advanced driver assistance systems – be it a simple one like advanced cruise control or more advanced ones – generally consist of a chain of effects that covers [160, Fig. 7.1]:

1. **Sensing**, i.e. measuring states of the environment. This for example involves taking Lidar (light detection and ranging) or GPS measurements,

2. **Perception**, i.e. fusing sensor data into a unified environment model, e.g. a local map of lanes and other vehicles,

3. **Decision Making/Planning**, e.g. assessing the situation, planning a path,
4. **Actuation/Control**, e.g. ensuring that the car follows the curvature of a planned path or accelerates to a desired speed.

These components become increasingly complex the more advanced a driver assistance system becomes. While early assistance systems limited themselves to speed control in free-flowing traffic, expanding the state of the art today requires operation in dense, multi-modal traffic and possibly less structured environments [163, p. 1576] without relying on a driver for supervision [44]. This requires highly accurate sensing, perception, planning and control, which stifled the optimism soon after car makers embarked on development efforts toward fully autonomous vehicles. In fact, experts assume that 45% of automotive software development cost will be spent on efforts towards this goal by 2030 [3]. However, recent studies concluded that customers are not willing to pay high premiums for assistance systems [116, 117], an effect that has been aggravated by the COVID-19 pandemic suffocating the world economy from 2020 on [116].

For cost-effective software development, modularity and code re-use are key. While this works for some of the more abstract components such as planning, these methods are not applicable to control. Control design entails devising a controller structure, which can generally be transfered between systems or vehicle models, and also tuning the controller. The tuned parameters are usually specific to a system, and must therefore be adapted if a new vehicle model is designed, a vehicle configuration added and sometimes even during development if low-level controllers change their dynamics or vehicle parameters change.

The parameters are usually optimized manually to maximize a (subjective) rating scale [64, 65, 33, 75] combining a perception of accuracy, comfort and safety. Optimizing parameters in test vehicles manually is a time-consuming (and boring) process for engineers, which, if avoided, could help reduce development cost for modern driver assistance systems.

Controllers that tune their parameters according to the environment are known as adaptive controllers, see [88] for an overview. In this context, we are looking for adaptive control that optimizes a performance criterion, a task that has been researched in the name of reinforcement learning (RL). RL is a technique that learns optimal controllers through trial and error.

Despite impressive displays of its potential in simulated tasks, it has not yet been widely applied to real systems, since it is known to not be sample efficient, struggles with partially observed noisy systems and is typically computationally heavy [79, 32].

This work adresses a part of this problem by enabling RL for speed tracking control. For this, it extends the capabilities of RL to enable online learning in a real vehicle, i.e. the gain is adjusted while the controller is following a varying speed using RL. Algorithms following two different paradigms in RL are presented: model based (MB) and model free (MF), that contrary to previous algorithms run online on limited hardware, are robust towards disturbances and delays in the real vehicle and are able to track arbitrarily time-varying setpoints. In addition to the feedback controller, two methods to incorporate a feedforward controller[1] are presented. In experiments on a real vehicle these RL algorithms are the first to ever successfully, automatically and repeatably tune controllers on a real vehicle within a few minutes, proving the fitness of the proposed solutions for day-to-day engineering practice. The results show that learning is robust and simultaneously adapts to changing conditions such as different speeds or disturbance levels.

The rest of this chapter introduces the problem of speed tracking control in more detail, placing it in the context of other works and stating our contribution, see Fig. 1.1 for an overview. Chapter 2 introduces groundwork and notation for parameter optimization and RL for continuous control, upon which we build our proposed algorithms in chapter 3. Experimental validation in a range of experiments in a real car is given in chapter 4 and discussed in Chapter 5.

1.2. Problem Description

In this section we introduce the technical environment of our controller, define the learning objective, the desired controller structure and introduce the constraints of our problem.

[1] In combination, feedback and feedforward controller are known as 2-degree-of-freedom controller, as feedforwards dynamics can be designed independently of feedback dynamics, i.e. responses to setpoint changes and disturbances can be influenced separately.

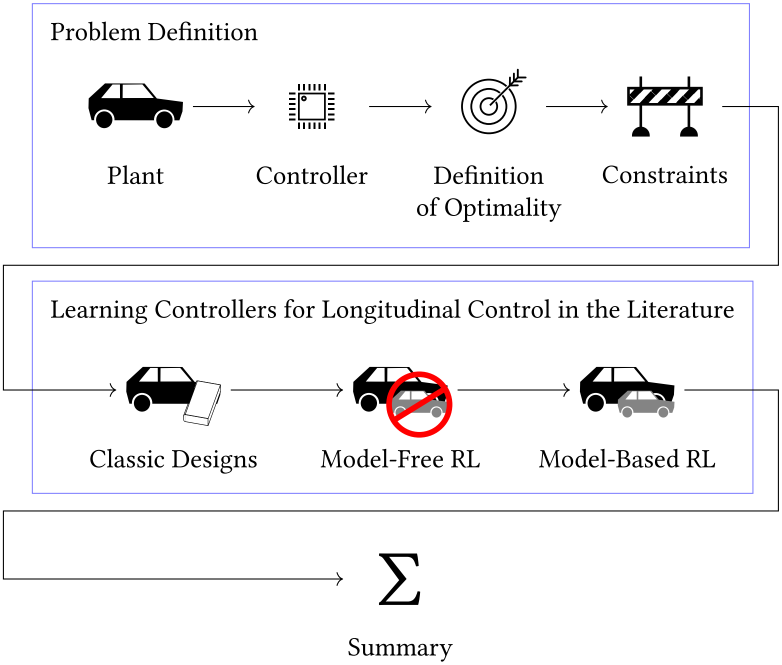


Figure 1.1.: Overview of the remaining introduction chapter: first, Section 1.2 provides a more detailed description of the problem. It begins with an introduction of the plant, i.e. longitudinal vehicle dynamics with its respective interface. The controller structure closing the loop with the plant is presented next. It includes the parameters that we set out to automatically learn in this work. Here, learning entails optimizing the controller parameter set, therefore this work covers the definition of optimality in the following part. The problem definition section is completed by constraints on the learning and controlling task this work strives to solve. With the task defined potential approaches are surveyed in Section 1.3. The section starts with classic (non-learning) controller design methods. Next, it glances over model-free RL methods, which will be discussed in more depth later, and gives an intuition on the relation to model-based RL. Finally, the introductory chapter concludes with a summary and sketches the research gap this work aims to close. At the end of Chapter 2 the description of the research gap is revisited once necessary concepts and notation are introduced.

1.2.1. Plant

Traditionally, control for road vehicles is divided into longitudinal and lateral control [160, p. 44]. In this work, we focus on the former. This section provides the context to the controller to be designed by giving an overview

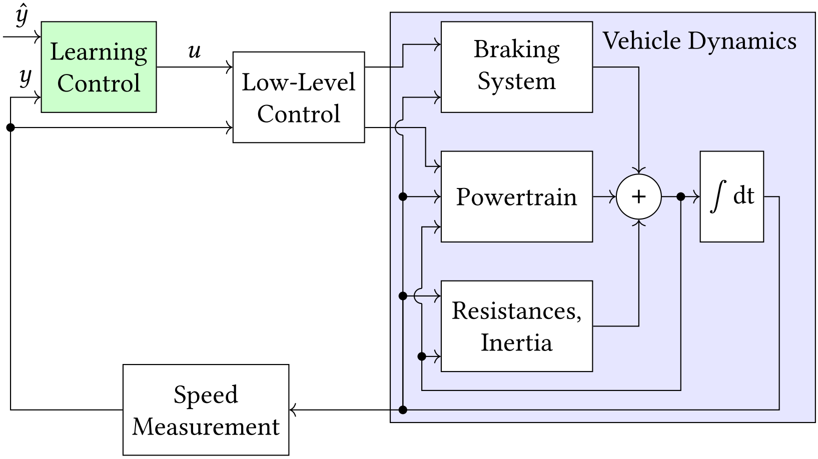


Figure 1.2.: Overview of the plant structure. The goal of this work is to design the learning controller (marked in green) that is fed a measurement of the speed y and a target speed $\hat{y}$, possibly with a preview. It outputs a commanded acceleration u that serves as an input to the low-level controller. The low-level controller tries to provide steady-state accuracy on acceleration level and allocates controls to the actuators. Longitudinal dynamics consists of resistances, inertia, hydraulic braking system, a powertrain containing a combustion engine, a hydraulic torque converter with lock-up clutch and an automatic transmission. Resistances include internal resistances, rolling resistance, aerodynamic drag and road grade. The sum of these elements forms the acceleration that is then integrated to vehicle speed. Speed is measured by wheel speed sensors and an evaluation logic.

of the plant structure and controller interface, see Fig. 1.2 for a schematic overview. Additionally, a simplified model of the plant dynamics is derived that is later used to give an intuition of the effect of hyperparameters, i.e. parameters that influence the learning behavior and result, and illustrate some of the algorithm design choices.

This work designs a speed tracking controller that aims to follow a target speed $\hat{y}$ based on measurements of the current speed y by demanding an acceleration u from a low-level controller. Trajectory planning is outside the scope of this work, see e.g. [123, 53] for state of the art approaches[2]. We assume a trajectory that can be previewed over a limited horizon but is otherwise arbitrary. The low-level controller tries to achieve steady-state accuracy of acceleration and thus compensate for steady-state resistances like road grade or aerodynamic drag using an integrator. It allocates controls to the actuator subsystems in longitudinal dynamics, i.e. the braking system and powertrain and makes use of drag torque of the engine. Commands to the powertrain are filtered depending on vehicle speed to prevent excessive wear on the variable valve timing assembly while allowing precise parking maneuvers. The two actuator systems differ in their lag time and dynamic response: the brake system reacts more promptly to torque demands than the powertrain. Both are affected by communication delays and are controlled in a purely feedforward fashion. Disturbances like road grade and driving resistances, consisting of aerodynamic drag, rolling resistance and internal resistances as well as inertia affect vehicle acceleration additionally. Vehicle acceleration is integrated to vehicle speed, that is measured using wheel speed sensors and an evaluation logic: Wheel speed sensors are sensors that measure rotation increments of a wheel and are inherently discretized (see e.g. [163, p. 288]). An evaluation logic detects slipping wheels and chooses the most appropriate evaluation method for the operation conditions and returns a quantized, slightly delayed speed signal.

While this work does not assume a valid model of the plant exists, most building blocks would require a complex model with unobserved internal states and possibly unknown parameters:

1. The low-level controller would be available, but its internal states (e.g. integrators, estimators) are not fed back to the controller to be developed. In addition it has situational and nonlinear behavior.

2. The dynamics of the braking system depend on disk temperature, wear and surface deposits (e.g. rust, water) – and of course traction and the respective control systems, e.g. anti-lock. None of these factors is measured directly, and no feedback is given to the learning controller.

[2] We limit ourselves to speed tracking control to limit the complexity of the learning problem, albeit traditional control approaches often also include position, acceleration, and sometimes even additional derivatives of acceleration in their planning and control.

3. The powertrain in itself is highly complex, its internal states like manifold pressure, temperatures, engine speed or gear are not available to the learning controller.
4. While some resistances are canceled out in the steady state, short disturbances resulting from bumps, short ramps, wind gusts or tight cornering affect longitudinal dynamics. Inertia may change as an effect of vehicle load or gear change. These effects are not directly reported to the controller.

The problem at hand therefore features a combination of unknown states and parameters for a complex dynamic system, which prevents the use of model-based observers. Learning controllers as investigated in this work could circumvent this issue.

Since the real plant poses challenges beyond a simulated environment, we will not limit ourselves to learning in simulation. Nontheless, we provide a simplified linear model in the box below, which we will use to illustrate some characteristics of the proposed algorithms.

Linear Model for Plant Dynamics

The following presents a simplified model that disregards nonlinear effects, e.g. due to torque magnification at low speeds or asymmetry between braking system and powertrain. It serves as an example for development of the proposed approaches.

For this, the low-level controller is assumed to be ideal and able to compensate resistances. The remaining longitudinal dynamics consisting of actuator dynamics and inertia are modeled using an ideally dampened system of second order with time constant $T_{\text{acc}} = 0.1$. A similar structure is chosen in [2, 123]. The acceleration is integrated to vehicle speed, leading to a (continuous-time) third order system

$$\dot{\breve{x}} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & -\frac{1}{T_{\text{acc}}^2} & -\frac{2}{T_{\text{acc}}} \end{bmatrix} \breve{x} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{T_{\text{acc}}^2} \end{bmatrix} u \tag{1.1}$$

with delay

$$x(T) = \breve{x}(T - 80\,\text{ms}). \tag{1.2}$$

T is the continuous time coordinate. The system is discretized[a] to a step size $\Delta t = 20\,\mathrm{ms}$. The measurement of speed is corrupted by noise ξ from a normal distribution with standard deviation[b] $\nu_\mathrm{y} = 0.01$. This yields a discrete-time system of order 7. With index t as the discrete time step we write

$$\begin{aligned} x_{t+1} &= A\, x_t + B\, u_t, \\ y_t &= C\, x_t + \xi_\mathrm{y} \text{ with} \\ A &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0.02 & 0.0002 \\ 0 & 0 & 0 & 0 & 0 & 0.9825 & 0.0164 \\ 0 & 0 & 0 & 0 & 0 & -1.6375 & 0.6550 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0.0001 \\ 0.0175 \\ 1.6375 \end{bmatrix}, \\ C &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \end{aligned} \tag{1.3}$$

In the examples given in the course of this thesis the matrices A, B and C are considered unknown in size and value. This model is used for experimental pre-studies in Chapter 3 and in Appendix A.3.

[a] The discrete time state x has a different order than its continuous time counterpart for ease of notation.

[b] The influence of the noise amplitude is investigated in Appendix A.3.5.

The next section closes the loop between commanded acceleration and measured speed by introducing the controller.

1.2.2. Controller Structure

This work aims to automatically adapt the parameters of an existing output feedback controller[3], that is introduced in this section. Its foundations were laid by Adiprasito [2, Sec. 4.3] and significantly reworked by Rathgeber [123,

[3] This choice enables us to compare the learned result with an expert's tuning in Chapter 4. Future work may entail searching for more advanced learning structures that e.g. integrate more of the low-level controllers.

Sec. 5.3.2][4]. The controller concept comprises elaborate feedforward and measures to counter stationary disturbances as well as position control in both longitudinal and lateral direction. In this work, we make use of the existing low-level controller to counter stationary acceleration disturbances, but otherwise limit our scope to speed control.

The controller to be learned is divided in feedforward and feedback[5]:

$$u_t = u_{\mathrm{ff},t} + u_{\mathrm{fb},t}. \tag{1.4}$$

To account for interpretability, both are chosen to be structures with tunable parameters that are well known in classic control theory. This work presents two options for the structure of the learning feedforward:

- an automated inversion-based feedforward design, represented in the form of a discrete-time linear system (see Section 3.2),
- or a learning optimal feedforward controller in the form of a finite impulse response filter for coefficients of the trajectory (see Section 3.1.2).

The feedback controller is designed as an output feedback controller as in [2]:

$$u_{\mathrm{fb},t} = \theta_{\mathrm{c}}(\hat{y}_t - y_t), \tag{1.5}$$

with (to-be-learned) controller gain θ_{c}. We assume that plant is stabilizable (see e.g. [85]) using output control. The optimal output feedback controller depends on the initial state (distribution) of the plant [90]. This is shown using the example system (1.3) in the appendix A.5.

Next, we turn our attention to the goal we want to achieve by tuning the parameter θ_{c}.

1.2.3. Definition of Optimality

In this section the optimization criterion for the learning controller is introduced. After a brief overview of the key optimization aspects, the criterion is

[4] In production vehicles, the controller schedules different gains according to vehicle speed.

[5] During learning, we add exploration noise as a third component.

presented in an RL compatible form alongside with configuration possibilities.

The presented criterion applies specifically to the feedback part of the controller. The feedforward part of the controller does not have to be optimal, since optimization has already been accounted for in the planning of the trajectory. The sole criterion for the feedforward component is therefore accuracy. The following applies to the feedback component.

Most important to tracking a planned trajectory is accuracy, but this needs to be balanced with comfort in most scenarios[6]. Traditionally, this parameter tuning goal is encoded in a subjective rating scale. It aims to capture an experienced passenger's perception of comfort and safety [64, 33, 65, 75]. These two aspects are linked according to [38]. Controllers for driver assistance systems are therefore tuned to be frugal in their outputs, since brisk actions are not only perceived as uncomfortable but may seem erratic and suggest lacking safety of the system. Commonly, the magnitude of acceleration is deemed to have predominant influence on ride comfort in the literature, i.e. to maximize comfort, acceleration should stay close to 0 [14, p. 87]. The controller therefore needs to balance accuracy with acceleration magnitude.

For the feedback controller, the problem can be described along the lines of classic (discrete-time) optimal control. The goal is to maximize a discounted infinite sum of rewards

$$\mathbb{E}_{y,\hat{y}}\left(V^{\rho}(y,\hat{y})\right) = \mathbb{E}_{y,\hat{y}}\left(\mathbb{E}\left(\sum_{i=t}^{\infty}\gamma^{i-t}R(y_i,\hat{y}_i,\rho(\hat{y}_i,y_i))\middle| y_0 = y, \hat{y}_0 = \hat{y}\right)\right) \tag{1.6}$$

with discount factor $0 \leq \gamma \leq 1$ and ρ denoting the control law to be learned. The operator $\mathbb{E}(\cdot)$ is the expectation operator, the vertical bar imposes conditions on the expression the expectation is drawn from. V is the state value, which is formed by the sum of rewards incurred by following the policy/control law ρ from state y and target $\hat{y}$ onward, see Section 2.2.1 for a more detailed introduction. The reward function is

$$R(y_t,\hat{y}_t,u_t) = -C_{\mathrm{y}}(\hat{y}_t - y_t)^2 - C_{\mathrm{u}}u_t^2. \tag{1.7}$$

[6] In most cases, it is the goal to keep passengers content, but for safety critical cases this principle must be abandoned, e.g. avoiding a suddenly appearing obstacle.

C_y and C_u are positive coefficients, the first weighs deviation from the control target, i.e. it accounts for tracking accuracy. The second punishes the magnitude of commanded acceleration. In the following, an aggressive controller is intended as a controller that counters control errors fiercely through its output and thus accounting less for any penalty on its output.

The optimization target can therefore be configured in the following way:

- The parameters C_y and C_u can be used to balance between tracking accuracy and comfort. [7].
- The discount factor γ makes the algorithm value events in the near future over those in the more distant future. The cost should be discounted as little as possible[8], i. e. γ should be close to 1, since from a passenger perspective the time of occurence of an unpleasant event does not matter.

Alternative Reward Function Including Jerk

Commonly, the magnitude of acceleration is seen as the predominant contributor to the passenger's perceived comfort. In some works, however, jerk, i.e. the derivative of the acceleration, is suggested as an additional influence [14, 38]. While this work mostly relies on the reward function (1.7), an alternative reward function including a measure of jerk is

$$R_\text{J}(y_t, \hat{y}_t, u_t, u_{t-1}) = -C_\text{y}(\hat{y}_t - y_t)^2 - C_\text{u}u_t^2 - C_{\Delta\text{u}}(u_t - u_{t-1})^2. \quad (1.8)$$

It includes an additional term accounting for the commanded jerk weighted by positive coefficient $C_{\Delta\text{u}}$. Chapter 4 shows that the proposed approaches are capable of learning with this reward formulation.

[7] Extreme choices, e.g. $C_\text{y} >> C_\text{u}$ can harm learning performance, see Appendix A.3.1.

[8] A discounted reward may not always define a sensible design objective, but has beneficial influence on value estimation and learning, see appendix A.3.1.

The state value (1.6) shall be maximized over the states that are encountered while running the policy ρ with parameter θ_c:

$$\theta_\mathrm{c} = \mathrm{argmax}_{\theta_\mathrm{c}} \mathrm{E}_{\hat{y},y} \left(V^{\pi}(\hat{y}, y) \right) . \tag{1.9}$$

However, the to-be-designed learning process must work with several limitations, that are introduced in the next section.

1.2.4. Constraints

The approach to be designed needs to comply with several constraints to be applicable in engineering practice:

1. **Learning Time:** The learning process must complete within minutes, otherwise it does not bring any benefit to manual tuning and incurs high cost for vehicle operation and supervision.
2. **Online Learning:** The algorithm must be capable of learning online, i.e. while the car is under control of the learning algorithm, to ensure that the learned controller does not provoke a system response outside recorded or simulated data. This additionally allows the supervising engineer to assess the controller performance during the learning process.
3. **Limited Knowledge:** No accurate model of the plant is available, requiring the learning algorithm to learn from experience.
4. **Limited Hardware:** Since network connection is not always available during test drives, learning needs to be done using onboard hardware, i.e. with limited computational power. A *dSpace Autobox* [30] containing a *dSpace DS1007 PPC* Processor Board [29] is employed that hosts additional tasks not related to this work. See Chapter 4 for more details.
5. **Limited Measurements:** Apart from the speed measurement signal y no information on the longitudinal dynamics state is provided.
6. **Limited Target Preview**: The control target $\hat{y}$ is known only over a finite horizon and it cannot be accurately predicted, since it may change arbitrarily.

With our task sketched out and the given constraints in mind, we take a look at what the state of the art has to offer in this context.

1.3. Learning Controllers for Longitudinal Control in the Literature

Before describing the research gap in the next section this work is put in the context of related literature. We begin with classic design methods for longitudinal control that generally rely on a known model. Then we turn our attention to RL, a method for learning optimal controllers from experience. First, MF approaches are surveyed, then methods that learn a model and use it to update the controller.

1.3.1. Classic Designs for Longitudinal Control

Controllers that rely on a known model are the state of the art in production ready vehicles. A variety of competing approaches exist:

- **PID Controllers** [10] are popular throughout many applications in control theory because of their intuitive, yet powerful design. The output consists of a sum of three terms: One is proportional (P) to the current control deviation, one integrates (I) past control deviations, and another one tries to predict future control deviations through the use of derivatives (D) of the control deviation. By tuning the gain for each of these, their influence can be adjusted up to a point of deactivating parts of it, creating e.g. a PD controller by deactivating the integrating compontent. This design has been adopted by [2, 23, 58, 122, 100, 132, 156]. A fractional order controller is obtained if not only the control deviation itself is integrated but also non-integer powers of it. This can help overcome shortcomings like overshoot or resonance with traditional PI(D) control and is therefore adopted by [66, 149].
- **State regulators** feed back multiple/all states of the plant. For vehicle applications controllers like this often vary their gains depending on the operating point [109].

- A different method to design a controller is to iteratively add control loops, treating the previously designed as part of the system. This design method called **cascaded control** is popular since longitudinal control involves acceleration, speed and distance traveled and therefore lends itself to such an approach. It has been applied by [98, 123].
- Decoupling or **linearizing controllers** try to impose independent or desired dynamics on specific states. Examples in longitudinal control are provided by [97, 99].
- **Fuzzy Control** is an approach that encodes semantic information for a control application. It tries to form a control law from intuitive verbalized knowledge (e.g. 'brake when you get too close') by blending multiple mappings from certain state measurements ('less than 4 m is too close') to actions ('braking means decelerating with $3\,\mathrm{m\,s^{-2}}$'). Examples for longitudinal control can be found in [28, 47, 100, 34]. The goal of a fuzzy design process is often to get near a desired way to operate a system but not necessarily an optimal way.
- **Neural Networks** (see 2.1.2) have been used to encode complex nonlinear control laws [47, 34] at the cost of interpretability. Training methods often replicate expert knowledge and rely on a simulated system.
- **Sliding Mode control** drives a system to a resting position along a pre-specified trajectory by switching between two or more control regimes that are divided by this trajectory. It is considered to be effective even when faced with uncertainties of the plant and has been applied by [34, 36, 62].
- **Model Predictive Control** is a method to optimally control a system based on dynamic programming. Starting at the current system state, a series of controller inputs is found that optimizes a predefined criterion over a time horizon in the simulation. Only the first control step is applied to the system, then a new planning loop begins. This approach is followed in [150, 34].
- **Robust Control** design methods assume a model and a measure of the insecurity of the model, often given as an interval. Robust control design compares controllers using the respective worst case of parameters within the uncertainty margins of the plant and picks the

controller that optimizes a given performance metric. H-∞ control is an example of such a metric that is used in [34]. If insecurities are large, the controller returned may be overly cautious and therefore exhibit low performance.

Given that traditional vehicle control is a mature field of research, this list likely does not do justice to this field of research. For this problem definition, these approaches are not applicable, since no accurate model of the plant is available.

Adaptive controllers [86, 88] provide an alternative to approach uncertainties in system models[9]. In contrast to robust control the control law adapts to changing system characteristics at runtime, e.g. to account for parameter variation. This may help achieve superior performance. An important distinction within adaptive control is the design (or performance) goal: some approaches define a target behavior, e.g. using a reference model [96], while other approaches optimize a cost/reward functional. The latter class is commonly referred to as RL[10]. For the application targeted by this work an optimal behavior reference cannot be given a priori, since the idea is to maximize passenger comfort and controller precision. The goal is therefore formulated as a to-be-optimized functional, making RL methods the algorithm group of choice.

We now turn our attention to MF RL control design methods.

1.3.2. Model-Free RL

Among adaptive controllers, the discipline of RL covers approaches that strive to learn optimal controllers by trial and error.

As the name suggests, MF RL methods estimate a measure of their performance directly from returns calculated from experience gained by interacting with the plant. Based on this estimate, they improve their control policy. We give a brief overview of applications that made this class of methods popular,

[9] Note that the ’adaptive’ in adaptive control has a more general meaning than in adaptive cruise control: while the control law is adapted in the former, the latter switches the control target depending on the traffic situation.

[10] A popular index term in control literature is approximate dynamic programming, which adresses a subgroup of RL approaches.

then we turn our attention to methods in control theory and applications to real-world systems. Finally, a few examples of applications in the automotive context will be given.

MF RL was recently used for impressive displays of potential. Among the most influential was AlphaZero [139], who clearly beat the earlier, but world-famous[11] RL algorithm AlphaGo [138] in the game of Go. A later iteration [137] was extended to support other games like chess. In [20] an artificial player managed to consistently win against a round of professional Texas Hold'em Poker players. In the wake of this success RL was applied to more complex video games like Quake III [71], Dota 2 [108] and StarCraft II [155], often at levels that compete with or exceed top-level human players. Beyond games, MF RL has been applied to recommendation systems at Facebook [46] and the Chinese bing News platform [169].

Due to its relation to control theory, applications as controllers have been proposed early on: In [19] the authors apply MF RL to a linear quadratic regulator problem in 1993, later on [6] compared RL to a PI controller for a simulated heating coil. Several benchmark problems have been proposed since [59].

Most of the applications for RL are in simulated or virtual domains. Applications in real-world tasks are considered hard for RL [79, 32] due to factors like sample efficiency, training time, delays, partially observed states, noise and others. Nonetheless, there are a few notable examples of successful RL applications with physical systems. Robotic grasping has been addressed in [9, 74, 51]. Four-legged robots have learned to walk using MF RL in [54, 56]. To the author's knowledge, no approach that learns to control longitudinal dynamics of a real vehicle online has been presented. Longitudinal dynamics differs from hand-like manipulators and walking robots by its comparably slow and asymmetric behavior and its integrator properties along with very limited measurements and computational resources.

Applications in the vehicle domain are based on simulation or on recorded data, but occurred early on: In [42] the authors propose an algorithm to adapt parameters for active roll stabilization and [67] introduces a method for tuning a PID controller for idling the engine using RL. A method for learning steering control from prerecorded data has been proposed in [124]. After that,

[11] AlphaGo was the first algorithm to clearly defeat a human professional player.

several applications of RL to vehicle control have been presented, see [87] for a survey. All of the presented methods for longitudinal dynamics control learn in simulation or on recorded data. An impressive example is given by [40], where a policy combining planning and control is used for navigating a parking lot after being trained in simulation over several hours.

1.3.3. Model-Based RL

MB RL at least partly replaces data from the plant of interest with a (learned) model thereof. Potential benefits over MF RL can be sample efficiency or added safety. The risk is that the learned model does not capture the system dynamics well and the learned control policy may fail to achieve the goal.

MB RL has been pioneered by the Dyna architecture [144] and has since risen to milestone examples like Google's datacenter cooling application [89]. Very generic algorithms like Dreamer [57] have proven that MB algorithms can bring benefits even in partially observed, high-dimensional and very diverse environments.

By learning a model and estimating its accuracy, ideas from robust control can be incorporated to guarantee safety, which is especially beneficial when working with real systems, e.g. as demonstrated using a quadrotor [15] or an inverted pendulum [16]. In [27] the MB approach was used for fast learning of a control law for a unicycle and cart-pole.

The survey [115] names several applications of MB RL to unmanned aerial, underwater and ground vehicles, mostly focused on navigation. Applications of MB RL to problems in vehicle control are rare. The authors of [165] propose a method to tune a PID speed controller using evolutionary parameter search for the controller in a simulated task. In [24], a policy search algorithm aided by simulated models of different complexity learns to drift a radio-controlled model car.

1.4. Summary and Research Gap

This section briefly summarizes the problem statement and condensates it in the form of three research questions that point at the contribution of this work.

Automated tuning of vehicle speed tracking controllers could save automotive control engineers time and effort. This work bases the controller on a given design to ensure interpretability of the gains and aim to tune it in accordance with an optimal control performance criterion balancing ride comfort and tracking accuracy over a long horizon. The task is challenging due to multiple aspects: the plant is nonlinear and its state only partially observed. The control target is known only over a limited horizon and can develop arbitrarily. Beyond that, practical application brings several limitations with it, such as limited computation power, time constraints, noise and stochasticity.

While learning algorithms have given impressive displays of their potential, speed tracking control for a real vehicle has not yet been solved, likely due to the asymmetric and comparably slow, partially observed dynamics as well as arbitrarily evolving control target. On the other hand, traditional algorithms require a model for their design, which requires additional effort to validate and will have limited accuracy. This work therefore sets out to make use of the potential of RL algorithms for speed tracking control in real road vehicles.

To solve the challenge, the capabilities of RL need to be extended. This task is divided in three main questions:

1. How can optimal speed tracking controllers be learned? For this, the capabilities of RL have to be expanded to tracking control and online learning in a real vehicle.

2. How do MB and MF methods compare in this task?

3. How can a feedforward be learned in an RL framework?

First and foremost this work strives to successfully enable RL for tracking control and online learning in a real vehicle, answering question 1. For this, it needs to enable RL for tracking control and online learning in a real vehicle. Two RL algorithms, one MF and one MB are proposed to address this question.

The MF algorithm is based on the actor-critic architecture with two modifications: first, this work presents a reconstructing structure for the state-action value function along with the required state augmentation pattern (see Section 3.1.1). Then a method to modify experienced state transitions for training is presented that restores the Markov property for training with arbitrarily varying control targets (see Section 3.1.2).

The proposed MB algorithm (see Section 3.2) is based on model learning and policy search. A decaying maximum norm is employed to dampen variance during training.

Experiments show that both algorithms are capable of learning on constrained hardware under different conditions in Chapter 4.

Both MF and MB methods contain sources for potential bias and variance that can yield suboptimal results or amount to failure of the learning task in the worst case. The question of sample efficiency is important when operating a real-world system where experiment duration is an important factor. This work therefore compares the two approaches to answer question 2. The experiments show that both learn fairly quickly and succeed in most use cases, while the MB algorithm exhibits slightly higher robustness towards the choice of exploration signal. This qualifies especially the MB learning method for day-to-day engineering practice.

In order to achieve state of the art controller performance at following a trajectory, feedforward control is necessary. Two different ways are proposed to respond to question 3: one approach extends the proposed training data manipulation to a framework that enables following arbitrary trajectories with a learned MF algorithm in Section 3.1.2. As a comparison method a non-optimal inversion-based design for the MB approach is proposed in Section 3.2.2. An experimental comparison shows that the optimal design yields a slightly softer controller (see Chapter 4).

The next chapter introduces important concepts in the state of the art that lay the groundwork for our proposed MB and MF algorithms. After introducing important notation and concepts, it specifically describes how the challenges at hand affect RL algorithms. At the end of the chapter these challenges are therefore revisited to accurately describe the research gap in a technical way, which is the starting point for the subsequent Chapter 3, in which the proposed enhancements on MB and MB RL algorithms are laid out. These algorithms are then put to test in Chapter 4 in a series of experiments in real vehicles. Chapter 5 rounds this work off with a conclusion.

2. Prerequisites

This chapter covers the essential concepts and notation upon which this work builds its contribution in the next chapter. It begins with parameter fitting and optimization. These are used for adapting function approximators in RL, which is introduced afterwards: First, function approximation is applied to learn the state-action value function using temporal difference learning. Then policy search methods are explained and how they can make use of state-action value functions using the (deterministic) policy gradient. It is shown how learning a model can be incorporated in RL, e.g. to enhance the data efficiency. Since RL relies on trial and error, the concept of exploration is introduced. This chapter ends with a summary of the challenges that are still open beyond the state of the art for the envisioned task in speed tracking control.

2.1. Parameter Estimation

RL relies on fitting parameters of an approximator for the control policy, value functions or models. An important part is therefore parameter optimization. In this section we briefly introduce a general optimization problem, a few hints for popular optimizers[12] are given in the appendix. The section ends with a generalization of a function approximator with parameters, which is a form of an artificial neural network. The following section is based on [48, Sections 4.3 and 8.5] except where stated otherwise.

[12] In this work we use the term *optimizer* as a synonym to *optimization algorithm*.

2.1.1. Parameter Optimization Problem

Here we have a brief look at an optimization problem with the respective parameter vectors and the loss function. We point out some aspects that are special to the problems encountered in this work such as stochasticity.

Optimization strives to find the value of a parameter that drives an expression to an extreme value. Within this work we consider only a subset of optimization problems that are constituted by the expression L called loss function or target function and a continuous parameter vector θ with no additional constraints. The goal is to find the parameter vector θ^* that minimizes the value of L:

$$\theta^* = \arg\min_{\theta} L(\theta, \{X\}). \tag{2.1}$$

This task is considered a supervised learning problem in artificial intelligence. The operator $\{\cdot\}$ marks a set of elements of the same kind, e.g. state vectors. The loss function L may have other inputs $\{X\}$ in addition to the parameter vector θ and is typically intended to approximate a (large or infinite) sum over a per-element loss function l. An example problem could be the estimation of model parameters θ for a dynamic model with continuous state vector included in X such that it reproduces the behavior of a real system. The loss function could in this case be intended as the squared approximation error l of the model output to the system output averaged over the entire state space. Since computing such a value exactly is not feasible, either because it is mathematically not tractable, empirically infeasible or prohibitive performance-wise, this performance metric is approximated using a randomly sampled finite set of datapoints $\{X\}_b$. This finite set of samples is often referred to as minibatch and is sampled out of an available batch of data, that is generally finite as well. The number of samples per minibatch is called batchsize b. In the context of this work, loss functions have the form

$$L(\theta, \{X\}_b) = \frac{1}{b} \sum_{i=1}^{b} l(\theta, \{X\}_{(i)}), \tag{2.2}$$

where $\{X\}_{(i)}$ denotes the i-th element of the batch $\{X\}$. This approximation comes at the cost of stochasticity, i.e. each call to the loss function yields a slightly different result due to randomized sampling of the minibatch. Solution algorithms for stochastic optimization therefore need to be tolerant towards

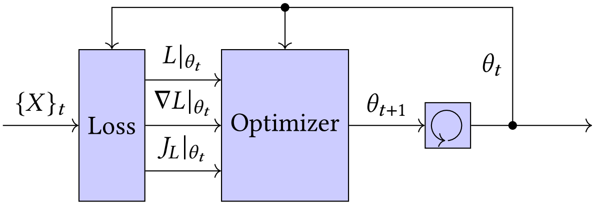


Figure 2.1.: Schematic of a supervised learning problem. At each time step t a loss function computes value L_t, gradient ∇L_t and jacobian $J_{L,t}$ from a set of parameters θ_t and a minibatch of additional inputs $\{X\}_t$. The optimizer iteratively returns new estimates of the parameter vector θ_{t+1} from the loss function outputs and the former parameter vector θ_t.

this additional challenge. Some optimization problems even involve a non-stationary loss function. In this case the optimization goal changes over time.

Optimization problems are generally solved by an iterative optimization algorithm, e.g. by taking steps in the opposite direction of the gradient $\nabla L = {}^{\mathrm{d}L}/_{\mathrm{d}\theta}$, possibly making use of the jacobian $J_L = {}^{1}/_{b}\left[\, {}^{\mathrm{d}l(\theta,\{X\}_{(1)})}/_{\mathrm{d}\theta} \cdots {}^{\mathrm{d}l(\theta,\{X\}_{(b)})}/_{\mathrm{d}\theta} \,\right]^{\top}$. If these are not available, in some cases they can be numerically approximated by finite differences. Fig. 2.1 is a schematic for an iterative solution of a supervised learning problem consisting of a loss function and an optimizer.

In the following, we will omit the data $\{X\}$ when referring to the loss function in most occasions and only explicitly include it if necessary. Instead, we will use the notation $\cdot|$, e.g. $L|_{\theta_t}$ to express that the loss function L has been evaluated using parameters θ_t.

Algorithms designed to solve these optimization problems are briefly presented in Appendix A.1.

2.1.2. Neural Network Concept

Artificial neural networks are a powerful yet flexible concept that is employed for function approximation (regression) in the context of this work [128]. We introduce the general concept, the backpropagation mechanism for calculating derivatives and a few common types of layers, i.e. building blocks of a neural network.

Despite more complex network patterns exist in the literature, we limit ourselves to the following structure: An artificial neural network $N : X_N \mapsto Y_N$ consists of one or more functions $f : X_f \mapsto Y_f$, each with a set of parameters w which is often called weights in the literature. m layers are daisy-chained to one another to form the network N:

$$N\left(X_N\right) = f_m\left(f_{m-1}\left(\ldots\left(f_1\left(X_N\right)\right)\right)\right). \tag{2.3}$$

Each layer may consist of subfunctions that divide the inputs to the layer among each other, allowing arbitrary serial[13] and parallel combinations. The advantage of building complex functions as a chain of subfunctions is that they lend themselves to tasks that require abstraction while their gradient can be computed by iteratively applying the chain rule:

$$\frac{\mathrm{d}N}{\mathrm{d}X_N} = \left.\frac{\mathrm{d}f_m}{\mathrm{d}X_m}\right|_{X_m=f_{m-1}(X_{m-1})} \left.\frac{\mathrm{d}f_{m-1}}{\mathrm{d}X_{m-1}}\right|_{X_{m-1}=f_{m-2}(X_{m-2})} \cdots \left.\frac{\mathrm{d}f_1}{\mathrm{d}X_1}\right|_{X_1=X_N}. \tag{2.4}$$

Similarly, as any parameter in the function can be treated as an input, gradients with respect to parameters $\frac{\mathrm{d}N}{\mathrm{d}\theta_i}$ with $i = 1, \ldots, m$ can be formulated. The backpropagation algorithm provides a computationally efficient way for calculating this gradient. The workflow for training a neural network N, i.e. optimizing its parameters $\theta = \begin{bmatrix} w_1 & \ldots & w_{m-1} & w_m \end{bmatrix}$ to make it approximate a desired mapping $\hat{N} : X_{\hat{N}} \mapsto Y_{\hat{N}}$ requires a loss function, e.g. $L(\{X\}_b) = {}^1\!/_b \sum_{i=1}^{b} l(\{X\}_{(i)})$ with $l(X_L) = {}^1\!/_2 \left(\hat{N}(X_L) - N(X_L)\right)^2$. Then, training consists of repeatedly sampling a batch $\{X\}_b$, computing the value and gradient (and possibly the jacobian) of the loss function and performing an optimizer step until the loss value falls below a chosen threshold, see Fig. 2.1 for a schematic.

The data used for training should cover the area that the function approximator should be trained for and for stochastic optimization to work best, the samples used for training are assumed to be representative of the underlying distribution. Therefore, it is often assumed that the data fed to the training process is indepentently and identically distributed [128].

[13] In addition to feedforward networks in the literature there are networks that base their output on their previous outputs, known as recurrent layers, e.g. [35], which are useful to describe processes over time. This, however, comes at the cost of a more complicated training procedure [61]. A neural network containing one or more recurrent layers is called recurrent neural network (RNN).

Different kinds of layers exist, e.g. convolutional or normalizing layers. The most popular kinds consists of a nonlinear activation function $a : Z_a \mapsto Y_a$ that is applied to a biased and weighted sum of its inputs:

$$Y_f = a\left(w \begin{bmatrix} X_f \\ 1 \end{bmatrix}\right). \tag{2.5}$$

Popular activation functions a include arc tangent, rectified linear units and even a purely linear, i.e. "pass-through" variant in some cases. Depending on the number of parameters and the computational complexity of the layer, training can become resource intensive for large networks.

Neural networks are being used as a framework for estimation of models, value functions and control policies, all of which can be parts of RL algorithms presented in the following.

2.2. RL for Continuous Control

This section aims to explain the actor-critic architecture, which is a popular setup for RL algorithms in continuous domains. Actor-critic combines an element estimating *how good* it is to choose a specific action in a specific state (a state-action value function) with an element that chooses said action depending on the state (a policy).

The following section is structured as follows:

- It starts with the definition of the RL problem. We explain the important concepts of state(-action) values.
- Then, we show how these state-action values can be learned using temporal difference learning. We point out that state-action-values are sufficient to design a basic RL algorithm (Q-Learning).
- An explicit (additional) policy is advantageous in continuous domains. Therefore, we explain how policy gradients can be used to improve a policy encoded in a function approximator.
- Many popular extensions to the actor-critic architecture have been presented. Experience replay and target networks are shown, which help make the training process more robust and precise.

- Since RL is based on the trial-and-error principle, exploration is vital for the learning success. We give an overview of ways to ensure exploration during learning.
- At the end of the section we explain how a learning model can be ingrained to the algorithm.

In the next section we round off with a summary of the deterministic actor-critic algorithm for continuous control that serves as a basis for this work and highlight the challenges we face when applying the algorithm to a real car. These challenges are adressed with the proposed algorithms in Chapter 3.

2.2.1. The RL Problem and State(-Action)-Values

Here we define the RL problem, consisting of the plant, a reward function and the RL agent, i.e. the learning controller. Special attention is paid to the aspects that set the problem at hand apart from the classic RL problem. The goal is to maximize the long term reward, which is defined through the accumulation of rewards to a state value.

For this work, we assume a discrete equidistant time scale with step index t. According to [145, Section 3.1] the RL problem is divided in two parts (see Fig. 2.2):

- an environment, that accepts a control action u_t and returns a state[14] vector s_{t+1} along with an instantaneous reward r_{t+1} and
- the RL agent, i.e. an algorithm that strives to learn to map the state vector s_t to an action u_t from observed state-transition tuples[15] of the form $s_k, u_k, r_{k+1}, s_{k+1}$ (and possibly u_{k+1}).

The environment is modeled as a Markov decision process that performs two mappings: it maps the state vector $s_t \in \mathbb{R}^n$ and action[16] $u_t \in \mathbb{R}$ to the following state s_{t+1} according to an unknown dynamics G and to a reward

[14] Since this section aims to introduce RL in general, we skip a detailed definition of the state vector s for now. The definition used for this work is given in equation (3.2) and expanded to include a target preview in (3.7).

[15] The tuples to learn from may be observed at a time step $k \neq t$, see Section 2.2.4.

[16] The action space can have multiple dimensions, but we limit ourselves to scalar action spaces in this work.

$r_{t+1} \in \mathbb{R}$ according to a reward function R [145, Section 3.1]. The environment therefore comprises the plant, accounting for the dynamics and a reward function R used to compute the reward signal r from s and u. The reward function encodes the task goal, e.g. balancing deviation from a trajectory with control effort. The Markov decision process is memory-less, i.e. the state vector s must be fully observed, but the process may be stochastic [145, Section 3.1].

Within this work, this definition is altered slightly (also see Section 1.2 and Fig. 2.3):

- In real-world applications, the state vector may be only partially available (e.g. due to delays or partial measurements). For clarity this work therefore differentiates between the plant state x, the plant output y and the state vector s that is provided to the RL algorithm. For the proposed approach s is defined differently, but for now we assume $s = x$, the Markov assumption is fulfilled.
- Later on a control target is provided as part of the state vector s. It varies according to a planned trajectory according to factors not available to the agent. Therefore, it violates the Markov assumption.
- Additionally, since the reward function is often designed by the engineers designing the controller, the reward function is assumed to be known. It is also redefined later to not rely on the full state.

The goal for this work is to learn an optimal deterministic mapping $\rho_{\theta_\mathrm{c}} : s_t \mapsto u_t$ which is called policy or control law such that it maximizes rewards "in the long run" [145, Section 3.3]. The policy ρ_{θ_c} is parameterized by a vector θ_c. In order to be able to enhance the policy from experience, RL algorithms often resort to a random component (exploration noise) added to their policy to explore a larger and more diverse portion of the state-action space [145, Section 13.1]. The behavior policy π used during training therefore may be stochastic despite the learned policy ρ_{θ_c} being deterministic.[17] Throughout this work the behavior policy π_{θ_c} is understood as the result of adding a random component with expectation zero commonly referred to as exploration noise to the learned/learning policy ρ_{θ_c}. See Section 2.2.5 for the kinds of exploration noise used in this work.

[17] Learning is possible without a random component if the plant is sufficiently stochastic [136]. In that case the behavior policy π can be equal to the learned policy ρ_{θ_c}.

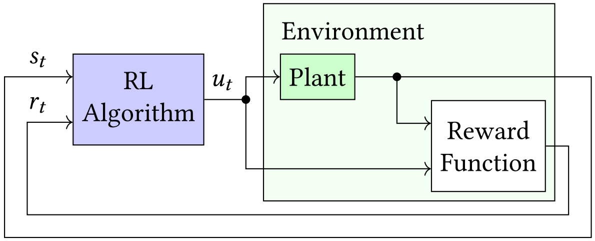


Figure 2.2.: Overview of the RL problem defintion. The plant (i.e. a transfer function G) is acted upon by the action u_t from the RL algorithm and returns a state vector s_t. The reward function R computes the reward signal r_t from action u_t and state s_t. Together, plant and reward function are considered a Markov decision problem with unknown dynamics, which is often referred to as environment in the literature [145, Section 3.1]. In the literature, both components are assumed to be unknown. This work considers the reward function to be known and will therefore include it in the RL algorithm later. The RL algorithm that solves the RL problem accepts a reward signal r_t and the state vector s_t, from which it learns to optimally pick an action u_t.

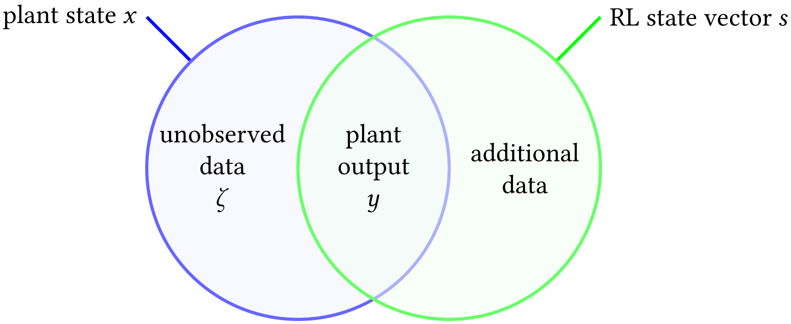


Figure 2.3.: Venn diagram for altered state definition: The plant state x, plant output y and the state vector s presented to the RL algorithm have separate symbols within this work. In vast parts of the RL literature, problems with the Markov property are treated that allow to treat these quantities as equals: $x = y = s$. Within this section we introduce the basic algorithm with this assumption in mind, but since we intend to apply the algorithm to a partially observed plant (x and y can have different dimensions) later, we augment the plant output with additional data (y and s can have different dimensions). These vectors will therefore be different.

The goal of learning an optimal policy can be formalized using the intermediate notions trajectory τ, return U and state value V.

- A **trajectory** τ is a semi-infinite series of states s and control actions u starting at time index t: $\tau_t = \langle\langle s_t, u_t\rangle, \langle s_{t+1}, u_{t+1}\rangle, \dots\rangle$ [145, Section

3.1]. If the trajectory results from applying actions that stem from policy π_{θ_c} to the environment, we mark it as τ^π. For convenience, we define the operators $\mathrm{s}(\tau_t, i_\tau) = s_{t+i_\tau}$ and $\mathrm{u}(\tau_t, i_\tau) = u_{t+i_\tau}$ that accept a local index i_τ as an offset to the trajectory index and return the respective state and action.

- The **return or utility** $U^\pi(\tau) = \sum_{i=0}^{\infty} \gamma^i R(\mathrm{s}(\tau^\pi, i), \mathrm{u}(\tau^\pi, i))$ of a trajectory is the discounted reward accumulated along a trajectory following policy π [145, eq. (3.11)]. Due to stochastic effects in the environment or policy two returns may differ despite starting at the same state, i.e. $U^\pi(\tau_k) \neq U^\pi(\tau_m)$ may occur for $k \neq m$ despite $\mathrm{s}(\tau_k, 0) = \mathrm{s}(\tau_m, 0)$ [145, Section 3.3].
- The **state value** $\hat{V}^\pi(s)$ is the expectation over the return U^π for trajectories starting from s and following policy π: $\hat{V}^\pi(s) = \mathbb{E}\left(U^\pi(\tau_i^\pi) | \mathrm{s}(\tau_i^\pi, 0) = s\right)$ [145, eq. (3.12)]. In other words, the state value $\hat{V}^\pi(s)$ is the expectation of the accumulated discounted rewards moving from a state s onwards when following policy π_{θ_c}. The Markov assumption ensures that a mapping from the state vector s to the expectation over the returns exists, i.e. the most accurate estimate of the expected reward from following the policy can be formed using only the state vector [145, Section 17.3].

The discount factor $\gamma \in [0, 1]$ is part of the definition of the optimization goal, reducing the impact of rewards in the distant future and resulting in a more impatient behavior if chosen close to 0 or a far-sighted behavior if chosen close to 1. In our application rewards are valued independently of when they are incurred, but it can have beneficial impacts on learning if chosen to be lower than 1 (see Section 4.1).

The learning goal can be formulated as maximization of expected state value using the policy parameter vector θ_c [145, eq. (3.13)]:

$$H = \mathop{\mathbb{E}}_{s\sim\int}\left(\hat{V}^\pi(s)\right) \tag{2.6}$$

$$\theta_\mathrm{c}^* = \arg\max_{\theta_\mathrm{c}} H \tag{2.7}$$

The distribution we take the expectation over arises by applying the control law π_{θ_c} to the plant G, i.e. the expectation is taken over the distribution of experienced states $\int$ from applying the control law π_{θ_c} that may differ[18] to some extent from the learned policy ρ_{θ_c}. The assumption that this distribution is stationary and exists is equivalent to the assumption that the combination of MDP and controller is ergodic or that the controller is stable within the experienced part of the state space[19] [145, Section 10.3].

The state-action value $\hat{Q}^{\pi}(s, u)$ is a reformulation of the state value $\hat{V}^{\pi}$ that is helpful for deriving the controller optimization algorithm later. It is defined as the expectation of the accumulated discounted reward gained from taking (arbitrary) controller action u_t in state s_t and following policy π_{θ_c} from the following state s_{t+1} on [145, Section 3.5]:

$$\hat{Q}^{\pi}(s_t, u_t) = \mathbb{E}\left(R(s_t, u_t) + \gamma U^{\pi}(\tau_{t+1})\right). \tag{2.8}$$

It can be shown that

$$\hat{V}^{\rho}(s) = \hat{Q}^{\rho}_{\theta_c}(s, \rho(s)) \tag{2.9}$$

for deterministic policy ρ_{θ_c} or

$$\hat{V}^{\pi}(s) = \mathop{\mathbb{E}}_{\pi_{\theta_c}(s)}\left(\hat{Q}^{\pi}(s, \pi_{\theta_c}(s))\right) \tag{2.10}$$

for stochastic policy π. The difference of state-action value function $\hat{Q}^{\pi}$ and state value function $\hat{V}^{\pi}$ is known as advantage function, as it encodes the difference in value of actions compared to the policy's choice depending on the state. In the next subsection we introduce a popular method to learn the state-action value function from experienced state transitions.

[18] Since learning hinges on the presence of exploration noise, it has to be included in the optimization goal. Usually, the actual goal is to optimize a deterministic policy ρ_{θ_c}. It is therefore necessary to find a balance between proximity to ρ_{θ_c} and exploration in the behavior policy π_{θ_c}.

[19] In [145, Section 10.3], the authors admit only combinations of policies and plants that result in stationary distributions of states that are independent of the initial state. This cannot bijectively be mapped to a stability definition in control theory, but concepts from control theory can be found that fit this description. For example, neutrally stable [10, Section 5.3], time-invariant combinations of plant and controller result in a temporally infinite, but repetitive trajectory in the state space. This guarantees the state to be within a finite portion of the state space (independently of the starting point, as long as it is within the set), thus assigning a finite and time-invariant probability density function to the respective subspace. Conversely, an unstable combination of plant and controller does not allow to confine the state to a finite area for an infinite amount of time, and thus prohibits the formulation of a time-invariant probability density function.

2.2.2. Temporal-Difference-Learning in Continuous State and Action Spaces

This section presents a loss function for learning state values, which are used to update the control law in the next section. We rely on [145] throughout this section.

In this section we try to train a function approximator to be an estimation of the state-action value function, which will later help maximize the state value. As we introduced in Section 2.1.2 training a function approximator $Q^{\rho}(s_t, u_t)$ requires target values, i.e. outputs $\hat{Q}^{\rho}(s_t, u_t)$ to given input tuples $\langle s_t, u_t \rangle$. While these can be obtained through straightforward Monte Carlo methods [145, Chapter 5], i.e. estimating state values by averaging utility from experienced rewards, methods based on bootstrapping [145, Chapter 6] have become more popular due to their superior performance. Following the bootstrapping principle means to estimate the utility, state value or state-action value using a combination of experienced rewards and the to-be-trained function approximator, thus allowing learning from incomplete trajectories down to even single state transitions. Bootstrapping is possible thanks to the recursive nature of the utility (e.g. $U^{\rho}(\tau_t) = \sum_{i=0}^{m} \gamma^i R(\mathrm{s}(\tau_t, i), \mathrm{u}(\tau_t, i)) + \gamma^{m+1} U^{\rho}(\tau_{t+m+1})$) and the linearity of the expectation. State(-action) values therefore fulfill the (one-step[20]) Bellman equation [145, Section 3.5]

$$\hat{Q}^{\rho}(s_t, u_t) = \mathbb{E}_{s_{t+1}, u_{t+1}\ \ \rho(s_t)} \left(r_{t+1} + \gamma \hat{Q}^{\rho}(s_{t+1}, u_{t+1}) \right). \tag{2.11}$$

The expectation on the right hand side provides a convenient way to calculate a target value for a function approximator $Q^{\rho}_{\theta_{\mathrm{sav}}}$ (parameterized using

[20] Multi-step formulations are possible by using more rewards from the experienced trajectory [145, Section 7.1], i.e. $r_{t+1}, r_{t+2}, \ldots, r_{t+k-1}$ instead of only r_{t+1}. The value $\hat{Q}^{\rho}(s_{t+k}, u_{t+k})$ from the function approximator is therefore used at a more distant time step ($t + k$ instead of $t + 1$), which has less influence if a discount factor $\gamma < 1$ is used. A possibly harmful bias of the function approximator (e.g. due to insufficient approximation power or simply because of unlucky initialization values) on the training target is therefore minimized. However, the experienced trajectory may be noisy and therefore not ideally reflect the behavior intended by the policy to be evaluated. Using more steps in this bootstrapping equation may therefore increase variance.

a vector θ_{sav}) evaluated at $\langle s_t, u_t \rangle$ by recurring at its value[21] resulting from the successor state-action combination $\langle s_{t+1}, u_{t+1} \rangle$. Depending on whether u_{t+1} and subsequent actions stem from experience, i.e. the behavior policy π, or the learning policy ρ, the state-action value function of the behavior policy π or the learning policy ρ is approximated (disregarding differences in the distribution of experienced state transitions). The former is considered on-policy learning and the latter off-policy [145, Section 5.4]. Off-policy learning is known to diverge if the learned policy significantly diverges from the behavior policy used for gathering experience [145, Section 11.3]. This work uses off-policy learning and tries to avoid this source of divergence by choosing π as a stochastic variant of ρ.

To make this use of relationship (2.11) for a real implementation, the expectation is estimated from single samples (i.e. the expectation is dropped). The approximation error δ_{TD} resulting from an imperfect function approximator[22] $Q^{\rho}_{\theta_{\text{sav}}}$ in this equation is known as temporal difference (TD) error [145, Section 6.1]:

$$\delta_{\text{TD}} = r_{t+1} - Q^{\rho}_{\theta_{\text{sav}}}(s_t, u_t) + \gamma Q^{\rho}_{\theta_{\text{sav}}}(s_{t+1}, \rho(s_{t+1})) \tag{2.12}$$

Training a function approximator to reduce a norm of the TD error using samples distributed throughout the area of interest in the state-action space can serve as a basis for taking a local step towards an improved policy [145, Chapters 9-11]. Algorithms proposed in this work minimize the squared per-element loss $l_{\text{TD}}(\langle s_t, u_t, r_{t+1}, s_{t+1} \rangle) = {}^1\!/_2 \delta^2_{\text{TD}}$ through loss

$$L_{\text{TD}}(\{\langle s_t, u_t, r_{t+1}, s_{t+1} \rangle\}_{b_{\text{crit}}}) = \frac{1}{b_{\text{crit}}} \sum_{i=1}^{b_{\text{crit}}} l_{\text{TD}}\left(\{\langle s_t, u_t, r_{t+1}, s_{t+1} \rangle\}_{(i)}\right). \tag{2.13}$$

b_{crit} is the batch size. This training resembles a supervised learning problem, except that the training target may move between optimization steps, see Fig. 2.4.

[21] Despite the dependence of $\hat{Q}^{\rho}(s_{t+1}, u_{t+1})$ on the parameters of the function approximator, the value is typically treated as a constant when it is included in a loss function, i.e. ignored when computing the gradient [91]. In this work, we adhere to this practice with a few exceptions (mostly published in [119]), which we mark as 'double-sided gradient' as opposed to 'single-sided gradient' in appendix A.4.

[22] Approximation errors may be due to suboptimal parameters or structure.

[23] The subsequent action u_t is calculated using the policy ρ, hence the schematic in Fig. 2.4 is off-policy.

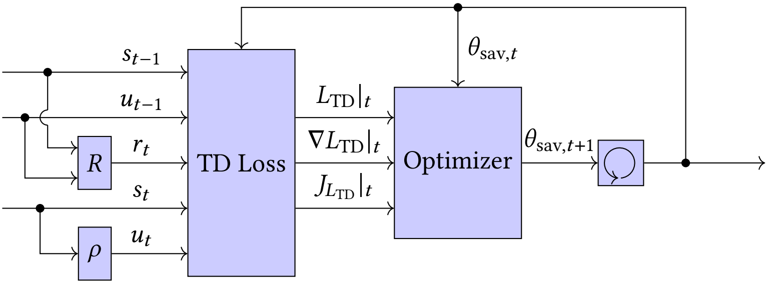


Figure 2.4.: Schematic of the off-policy temporal difference learning algorithm. TD learning can be implemented as a conventional optimization problem consisting of optimizer, memory and a loss function. However, it differs from the supervised learning problem in Fig. 2.1 in a way that the approximation typically changes over time (e.g. because the policy ρ evolves.). The inputs to the off-policy TD learning process are[23]the previous state s_{t-1}, the previous action u_{t-1} and the current state s_t. With the help of the reward function R and the policy ρ the TD loss value, gradient and jacobian can be obtained. For brevity, we use the notation $L_{\mathrm{TD}}|_t$, $\nabla L_{\mathrm{TD}}|_t$ and $J_{L_{\mathrm{TD}}}|_t$ to note that the expressions L_{TD}, ∇L_{TD} and $J_{L_{\mathrm{TD}}}$ were evaluated using the inputs of time step t, i.e. s_{t-1}, u_{t-1}, r_t, s_t and u_t along with parameter vector $\theta_{\mathrm{sav},t}$ and the policy ρ available at that time step. Along with the former iteration of the parameter vector $\theta_{\mathrm{sav},t}$, these are the input to the optimizer, which computes the next parameter vector iteration $\theta_{\mathrm{sav},t+1}$. In Section 2.2.3 we add target networks, an extension to TD learning that relies on separate copies of the parameters for policy and state(-action) value function for bootstrapping.

The conceptually simplest way of updating the policy once a state-action value function is available is to make the policy choose the action that maximizes the Q^ρ value[24] in each state s [145, Chapter 8]. For environments with discrete action (and state) spaces that may even work with a table for value function approximation, e.g. by a (hopefully simple) search, but since optimizing a (possibly large) neural network can be too computationally expensive, this approach known as Q-Learning is limited to the so-called tabular case. While efforts have been taken to allow the transfer of this idea to arbitrary neural networks by defining a final layer whose optimum can be trivially found [12], policy search and policy gradient methods have become far more popular. These methods are presented in the next section.

[24] For this maximization to yield the optimal action, the state-action value function approximator Q^ρ has to have a maximum in the location of the maximum of the actual state-action value function $\hat{Q}^\rho$, but may be otherwise imperfect.

2.2.3. Policy Search and Policy Gradient Methods

The ultimate goal to RL is making optimal decisions. The mapping from an environment state to an (optimal) decision is the (optimal) policy. It can be derived from the (optimal) state-action value function as we hinted at the end of the last section, but in continuous dimensions it is often more convenient to directly learn that mapping. Algorithms following this paradigm constitute the class of policy search algorithms [145, Chapter 13].

In the following we briefly outline a few ways to forego learning a state(-action) value function by sampling the returns from experience or using a learned model. Then, we turn our attention to policy gradients that improve the policy based on information from an estimated value function. This yields the basic actor-critic algorithm.

If the objective (2.7) can be sampled, e.g. by averaging over returns computed by applying a policy ρ with parameters θ_c to an available or learned model, then naive approaches like genetic algorithms/hill climbing can be successful [145, Section 1.5],[165]. For a more goal-oriented optimization, a gradient from finite differences of sampled rewards could be used. However, the sampled learning objective is usually subject to high variance, which may result in slow learning. We follow this approach in our MB RL algorithm (see Section 3.2).

Policy gradient methods instead do not require a model of the plant. They strive to provide an accurate estimate of the gradient of the learning objective in (2.7).

This work relíes on the deterministic policy gradient [136], which can be interpreted as replacing $\hat{V}^\rho$ with $\hat{Q}^\rho$ according to (2.9) in the policy learning objective (2.7) and then applying the chain rule to compute a derivative

$$\frac{\mathrm{d}H}{\mathrm{d}\theta_\mathrm{c}} = \mathbb{E}\left(\left.\frac{\mathrm{d}\hat{Q}^\rho}{\mathrm{d}u}\right|_{s,u=\rho(s)} \left.\frac{\mathrm{d}\rho_{\theta_\mathrm{c}}}{\mathrm{d}\theta_\mathrm{c}}\right|_s \right) \tag{2.14}$$

This gradient can be sampled from experienced states s_k with k denoting time steps during the agent's learning phase. The combination of learning a

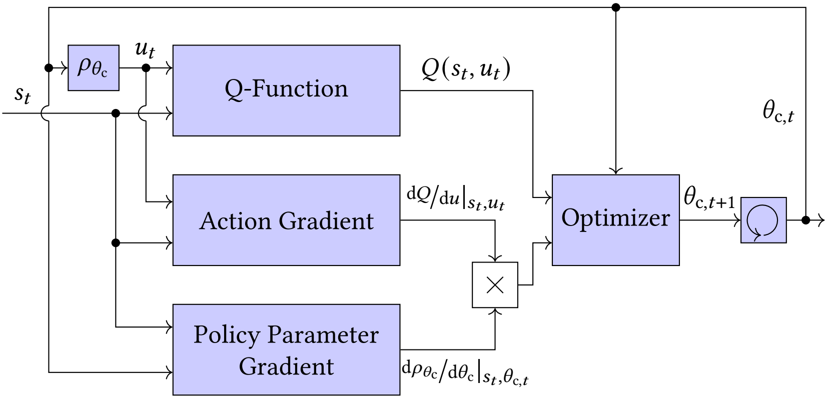

ρ_{θ_c}
u_t
s_t
Q-Function
$Q(s_t, u_t)$
$\theta_{c,t}$
Action Gradient
$\mathrm{d}Q/\mathrm{d}u|_{s_t,u_t}$
Optimizer
$\theta_{c,t+1}$
×
Policy Parameter Gradient
$\mathrm{d}\rho_{\theta_c}/\mathrm{d}\theta_c|_{s_t,\theta_{c,t}}$

Figure 2.5.: Schematic of the deterministic policy gradient algorithm assuming a learned state-action value function Q (Q-Function). The optimizer is provided with the state-action value $Q(s_t, u_t)$ and the deterministic policy gradient (2.14), consisting of a product of the gradient of the state-action value function with respect to the action and the parameter gradient of the policy. These quantities are computed based on the current action u_t and the current state s_t. The optimizer returns an updated estimate of the policy parameters $\theta_{c,t+1}$.

state-action value function with a (deterministic) policy gradient for policy updates yields an actor-critic algorithm[25] depicted in Fig. 2.5.

Provided the value function is approximated with high fidelity, actor-critic can learn quickly due to its low-variance gradient estimates. Depending on the quality of the function approximator, bias may be introduced [136, 145]. The next chapter provides two common extensions that help with accurate state-action value function learning.

[25] Methods that explicitly learn a policy are considered actor-only methods that e.g. directly optimize the policy on the average reward. Methods that only implicitly learn a policy like Q-Learning are referred to as critic-only. Actor-critic combines both elements.

2.2.4. Common Extensions: Experience Replay and Target Networks

Common extensions to the basic actor-critic algorithm are experience replay and target networks. These were invented to make learning more robust and minimize variance and bias in the learning process and were presented in [91].

Experience replay aims to provide the learning process of both state-action value function and policy with meaningful minibatches of experience.

In physical systems, states that occur within rapid succession to one another are often close value-wise. Using single samples sequentially to update the respective function approximators would therefore violate the assumption of independent samples in the training batch (see Section 2.1.2). Experience replay helps to break the temporal correlation by keeping a circular buffer of experienced state transitions to sample batches from [92]. By making the buffer large, the individual samples in the training batch are potentially temporally distant and therefore less similar. They may, however, have been collected using a policy that is outdated by the time they are included in the training batch. It has therefore been suggested to use an off-policy algorithm with experience replay. Another potential benefit of experience replay is that experience can be included in multiple training steps, thus increasing data efficiency. A schematic overview is given by Fig. 2.6. Appendix A.3 gives an intuition of the effect of batch size and experience storage size in a simulated example.

Target networks aim to eliminate another source of divergent learning: since the state-action value function approximator is used to calculate the target value, the target may vary considerably between two updates in the same state due to evolving weights. Since this variance can cause the learning process to diverge, a target network is used instead of the original state-action value function approximator. The target network weights $\tilde{\theta}_{\mathrm{sav},t}$ are updated using an exponential average of the parameters $\theta_{\mathrm{sav},t}$ in the state-action value function approximator:

$$\tilde{\theta}_{\mathrm{sav},t} = (1 - \eta_{\mathrm{crit}})\tilde{\theta}_{\mathrm{sav},t-1} + \eta_{\mathrm{crit}}\theta_{\mathrm{sav},t}. \tag{2.15}$$

The parameter $\eta_{\mathrm{crit}} \in (0, 1]$ can be chosen to favor smooth but slow progression of the target network weights $\tilde{\theta}_{\mathrm{sav},t}$ if chosen to be small or, if chosen close to 1, to allow for close tracking of the parameters $\theta_{\mathrm{sav},t}$ in the actual

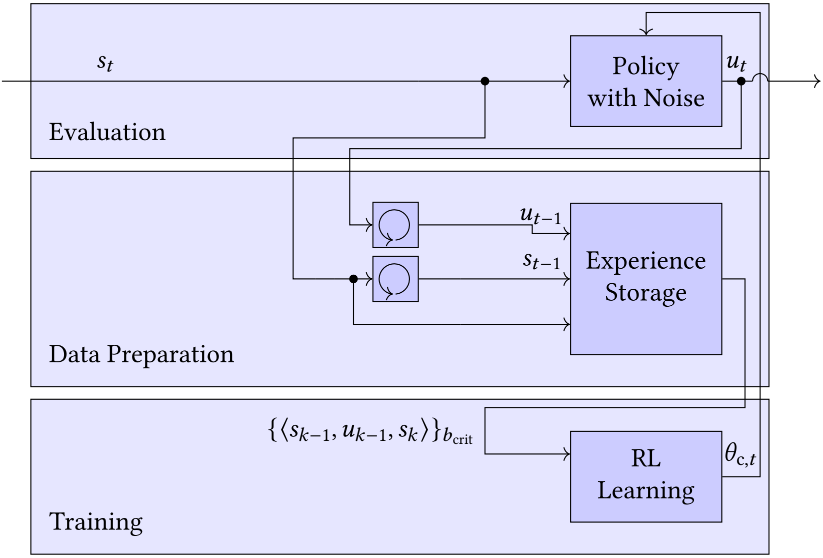


Figure 2.6.: Schematic of an RL algorithm using an experience storage. For better overview the algorithm is structured in three sections: evaluation contains the policy with exploration noise; the data preparation layer accepts the current state and action, and feeds the experience tuple $\langle s_{t-1}, u_{t-1}, s_t \rangle$ to the experience storage (reward omitted to avoid cluttering). From the experience storage the algorithm randomly picks experience tuples for the training part. Commonly, a minibatch $\{\langle s_{k-1}, u_{k-1}, s_k \rangle\}$ of more than one tuple is sampled at random from the experience storage, containing more diverse data compared to using only the latest experience tuple. The training step combines TD learning (see Section 2.2.2 and Fig. 2.4) for a state-action value function, which in turn is the basis for the deterministic policy gradient (see Section 2.2.3 and Fig. 2.5). This work expands this architecture with elements to make it applicable to the partially observed dynamics and enable learning for tracking control.

state action value function. While this extension may slow learning, it may increase stability as the authors of [91] claim[26]. In Appendix A.3 we provide an intuition on the effect of target networks on the learning process.

[26] A target network has also been used to avoid overestimation of state-action values in Q-learning variants [152]. Target networks for the policy are common, too [91].

Outlook: Beyond Deterministic Actor-Critic

This work focuses on deterministic actor-critic algorithms, but literature offers several popular alternatives. We only include them here for reference, since most of these improvements were added to enhance learning from visual representations using large networks.
In TD3 [43] the authors try to extend the deterministic policy gradient approach beyond DDPG [91] by decreasing the policy update frequency, applying the ideas from [153] in the form of clipped double Q-learning and other measures.
In recent years, stochastic policies have gained more attention. One of the central developments adding stability followed the idea of natural gradients [73], a method to enhance learning speed, in the form of trust-region policy optimization (TRPO) [130], proximal policy optimization (PPO) [131] and actor-critic using kronecker-factored trust region (ACKTR) [167]. Additionally, ideas presented for deterministic policy gradients were incorporated, e.g. experience replay [159] and soft actor-critic [55, 56].

2.2.5. Exploration

Even with off-policy TD-learning, the actor-critic algorithm relies on an exploration policy π that to some extent resembles the current policy ρ, yet deviates from it to estimate the advantage function[27]. Additionally, it needs to excite the system in a way that creates a rich distribution of states to learn from, which can be challenging when learning stable controllers. We introduce several options to add exploration noise and give a few pointers to other options for exploration, e.g. safe exploration.

Depending on the perspective, exploration noise helps the learning process in different ways:

[27] Similarity between the current policy ρ and the exploration policy π helps learning in configurations where the agent is not able to extrapolate experience between states. This may be due to the system behaving nonlinearly, but also due to the function approximators used in the agent. Yet, a slight deviation helps the agent understand if abandoning the currently learned policy can be beneficial, i.e. estimating the advantage.

- Within the RL literature, it is often considered as a means to explore the effect of alternatives to the current policy's choice and discover better options [145].
- Exploration noise serves as a signal for system identification, breaking the correlation between disturbances and commanded action [69].
- With a wider variety of experienced states and actions, the learned state-action value function and policy can be trained on more diverse batches (making individual learning steps more meaningful, see Section 2.1.2) and potentially contain valid information for a larger portion of the state-action space, enhancing the stability of the learning process and the quality of the learned solution.

RL was first introduced for discrete (action-)spaces, and exploration therefore typically amounted to nothing more than picking a random action every once in a while (ϵ-greedy) [145], possibly with schemes that reduce the random component with learning progress (e.g. in Boltzmann Exploration [158] or the more optimal, yet more complex Bayesian Exploration [143]), or assume very high returns for unvisited states (optimistic) [145]. For systems with a continuous action space, exploration noise is either inherent to a stochastic policy, defined as a distribution the action is sampled from [145] or as a random number added to the output of a deterministic policy [136]. Other choices include count-based exploration, i.e. preferring rarely visited state-action combinations [148], or curiosity-driven, i.e. preferring actions where a simultaneously trained model returns erroneous predictions [112] which can be seen as concepts within intrinsic motivation, i.e. optimizing an additional, known reward function that rewards exploration [13].

For real-world systems, purely random signals can be an inapt choice: on the one hand they can cause high wear on actuators, on the other hand they may fail to excite the system due to their emphasis on high frequencies. It has therefore become common to incorporate some temporal correlation between samples of the exploration signal, e.g. sampling from an Ornstein-Uhlenbeck process [91]. This work follows an idea from [161] and samples the random component for exploration at a lower frequency than the evaluation frequency of the controller.

Within this work we consider three options of adding noise to the deterministic policy:

- We add a random vector v_θ sampled from a (possibly multidimensional) uniform distribution in the interval $[-d_{\text{uniform}}; d_{\text{uniform}}]$ in a regular interval Δt_{noise} that may differ from the evaluation interval Δt to the policy parameter vector θ_{c}. This approach is known as parameter space noise in the literature [126, 154, 114].
- We add a random number v_{u} sampled from a uniform distribution in the interval $[-d_{\text{uniform}}; d_{\text{uniform}}]$ in a regular interval Δt_{noise} that may differ from the evaluation interval Δt to the policy output $\rho(s)$. The idea to add random noise to the output of deterministic policies was proposed by [136].
- We add a value v_{u} that toggles between $-d_{\text{rectangle}}$ and $d_{\text{rectangle}}$ in a regular interval Δt_{noise} that may differ from the evaluation interval Δt to the policy output $\rho(s)$. Rectangular signals are popular in control theory since they excite a vast range of frequencies according to Gibb's phenomenon [69].

While the three proposed approaches to exploration are applicable to the vehicle, it remains open if they succeed at providing sufficient exploration and excitation as well as making the agent experience a meaningful distribution of states. In Section 4.4 we therefore run experiments with each variant.

2.2.6. Connection to Model-Based RL

MB RL is often defined as substituting (at least a part of [72]) the experience from the environment with data generated by a (learned) model, but otherwise using the principles introduced above, see Fig. 2.7 for an overview on the integration of a model within this work. Some approaches deviate from that, e.g. by making use of specific model estimation methods to incorporate a notion of uncertainty in the learning process or by using more traditional control design methods. In this section we give a brief overview of model estimation methods and architectures used for MB RL.

Model estimation is a developed field of research (see e.g. [93]). In the context of RL, many kinds of model have been proposed. Popular ones include linear models [26], gaussian processes (e.g. PILCO [27]), nonlinear feedforward (e.g. MLAC [50]) or recurrent networks (e.g. Dreamer [57]); see [115] for a more extensive overview. Generally, it is assumed that the full state vector x of the

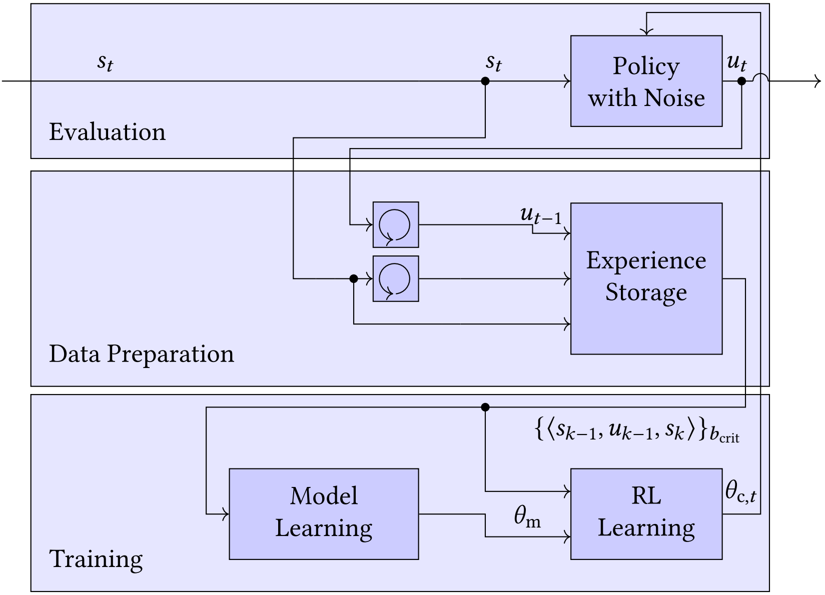


Figure 2.7.: Schematic of an RL algorithm using an experience storage and a learning model. A model is trained using experienced state transitions sampled from the experience storage. The trained model can either be used to generate additional experience tuples for learning a state(-action) value function or, as in this case, be directly used in the policy learning step, e.g. for sampling returns using simulations.

plant is observed (i.e. $s = x$), which is often not the case with real systems [32]. Partially observed systems require simultaneous estimation of state vector and model parameters. For some applications, ignoring unobserved dynamics, i.e. learning a model using only the observed portion of the state vector may yield a sufficiently accurate model [140]. The authors of [104] group approaches for reconstructing a complete state from partial state observations in windowing (concatenating past observations and actions in the state vector), belief states (maximize likelihood of observation), recurrency (implicitly form a state in a trained recurrent neural network) and external memory (memory cells that can explicitly accessed by a neural network).

MB algorithms are often claimed to be more sample-efficient, since a model can be learned from limited experience data and then used to generate a large amount of hypothetical experience [144] to learn a policy from at comparably low cost [115]. A policy may be learned by (MB) policy search, value function methods or actor-critic methods as introduced above. Some approaches make additional use of some characteristics of the estimator or the estimation process to guide the learning process, e.g. to account for robustness using the approximation error [15, 16, 25].

While MB algorithms have the potential to learn from less data, their success depends on the accuracy of the learned model, which can be a challenge for partially observed environments.

This work therefore follows two parallel approaches, one MB and one MF, which allows to compare them in the envisioned application.

2.3. Summary, Technical Description of the Research Gap

This chapter introduced a deterministic actor-critic algorithm as groundwork for this thesis. It is briefly summarize here and provide a synopsis of the challenges arising from the application to speed tracking control in a real vehicle is given.

From the point of view of the RL algorithm, the input is a state and the output is a commanded action, both are vectors of real continuous-value, discrete-time signals. The reward function is treated as part of the algorithm and defines the learning goal. The expectation of the discounted reward sum, i.e. the state value, is to be maximized in the long run over the distribution of visited states. Everything outside the algorithm is considered the environment and supposed to adhere to fully observed dynamics.

In practice, policy search methods and actor-critic designs are the most popular choices for continuous control tasks. Policy search aims to directly optimize control parameters to enhance the controller's performance, e.g. by sampling returns (Monte Carlo methods) and applying hill-climbing methods. The actor-critic architecture adds a value function estimator to guide policy improvement with less variance.

The deterministic framework this work harnesses the deterministic policy gradient to estimate the policy gradient from a Q-function learned by TD learning. Both deterministic policy and state-action value function are learned by applying LM optimization to function approximators trained on batches of experience that are stored in a ringbuffer. MB RL algorithms strive to increase sample efficiency by learning a model of the environment first and then employ it to generate experience to learn from.

When it comes to real applications, however, some of the assumptions that RL algorithms rely on, do not hold. In the case of speed tracking control, the challenges fall into three groups:

1. The algorithm has to work with relaxed assumptions on the environment:
    a) The plant dynamics violate the Markov assumption: it exhibits delays and only provides a speed measurement. This makes learning a model or a value function harder.
    b) The control target is arbitrary: there is no global description for its dynamics. State values can therefore only be predicted with large variance.
2. The algorithm has to be resource efficient: The learning process has to converge quickly within limited computational resources to avoid wear and experiment costs[28] while staying within onboard processing power. This is at odds with the many examples in the literature, where large amounts of training time on powerful computers and data are used for training an RL algorithm. Rapid learning on limited hardware is therefore a challenge for state of the art algorithms.
3. The algorithm has to be robust to real-world conditions:
    a) All occurring optimization problems are noisy, e.g. due to quantization or measurement noise.
    b) The vehicle itself exhibits some stochasticity, behaves nonlinearly at low speeds and its reaction to control inputs is not symmetric at high frequencies due to the brake system reacting more promptly. Low-level algorithms filter some of the inputs to limit wear on the powertrain. Since many of the

[28] Since giving an exact time frame for manual tuning is not possible, this work aims to solve the task within minutes, which would guarantee a time benefit over manual tuning.

displays of performance have been achieved in simulated environments that behave ideally, challenges from real-world systems have been neglected so far.

The next chapter proposes MB and a MF candidate designs to overcome the first two challenges. After that, we demonstrate that the proposed algorithms are capable of handling real-world conditions in Chapter 4 and thus the third challenge.

3. Proposed Approaches

Based on the basic concepts of RL introduced in Chapter 2, this chapter proposes two algorithms for vehicle speed tracking control. The work in this chapter can be understood as enabling MB and MF RL for the vehicle application, such that they can be used in experiments in Chapter 4.

The first algorithm is based on the MF deterministic actor-critic architecture and employs a reconstructing state-action value function network to overcome challenges arising from partially observed states. This work proposes to locally approximate the control target to remove variance due to changes in the control target and thus enable tracking control.

The second algorithm learns a model from windowed past states and actions that is then used to sample returns for policy search. The estimated model additionally serves as a basis for an automated inversion-based feedforward design.

Both algorithms create little computational burden, e.g. by relying on few parameters for approximation.

3.1. MF RL Algorithm

Here the actor-critic architecture introduced in Section 2.2 is expanded in two ways: Section 3.1.1 proposes a combination of augmenting the state vector and a special architecture for the function approximator in order to cope with the kind of partially observed systems arising from communication delays and slow actuators. Additionally, Section 3.1.2 proposes a local approximation of the control target along with a manipulation of the training data to enable learning with little variance while following arbitrary trajectories. Finally, Section 3.1.3 provides an overview of the complete algorithm.

3.1.1. Reconstructing State-Action Value Function Approximator for Partially Observed Plants

This section proposes an architecture for the state-action value function approximator to cope with partially observed dynamics (see challenge 1a) while keeping a small computational footprint (see challenge 2). The idea is to reconstruct part of the unobserved state from past actions since it is mainly a comparably slow actuator whose internal states are unobserved. This work has been published in [119] and [118].

The section begins by surveying approaches to partially observed systems in RL, then introduces three variants of the proposed combination of reconstructing layer and state augmentation. A few insights from a simulated example are given to underline the effectiveness of the proposed structure. As a preliminary to the extensive vehicle experiments in Chapter 4, we evaluate the reconstruction variants in the car.

3.1.1.1. Approaches for RL in Partially Observed Plants

There are two ways to apply RL to partially observed environments: either the measurement is treated as if it was the full state, i.e. the unobserved part of the state vector is disregarded, or past observations and actions are used to reconstruct the missing information.

Ignore Unobserved Dynamics With only partial information on the system state available, predictions for the coming states are more uncertain. This uncertainty in predicted states carries over to rewards and thus estimated returns. With the optimization goal being subject to variance, the optimization process is often slower and yields less precise results. For RL, this means possibly slow learning speed and sub-par performance in the converged state. Yet, ignoring the hidden portion of the state may work [140], especially RL algorithms with large nonlinear function approximators seem to fare well in these scenarios [133]. These are not feasible due to performance restrictions in our application (see challenge 2).

If the policy relies exclusively on observed features (i.e. entries in the state vector), a value function approximator can be constructed that guarantees bias-free policy gradients according to compatible function approximation theory [146, 136]. However, it has been shown that these simplest forms of compatible function approximation yield the same level of variance as an algorithm that learns without a value function [147].

While disregarding unobserved dynamics may seem tempting from a theoretical point of view, the toll on learning speed and accuracy can be high, as an example shows in Section 3.1.1.3. To avoid poor learning performance, missing state information can be reconstructed from past states and actions.

Reconstruct Missing Information from Input and State History Apart from estimators based on a known or learned model, reconstruction can be done using either a finite window of past observations and actions (also known as finite history, finite memory or windowing) as an input to a feedforward function approximator or using a memory in the function approximator [113, 1]. In environments with comparably fast unobserved dynamics, e.g. blinking in Atari games, using several past measurements has been proven to be effective [102, 79]. Augmenting the input space usually comes at the price of a larger function approximation network, i.e. higher computational complexity. Additionally, it raises the question of how to form the augmented state vector, i.e. how many past state measurements and/or actions to include. The authors of [61] point out that the algorithm's performance can suffer if important events fall beyond the limited horizon provided in the state vector and suggest to use recurrent neural network (RNN) (see Section 2.1.2). These come at the cost of a more difficult training procedure that is inherently in contrast to experience replay, since RNNs require the inputs to be presented in chronological order (see also [60]). Despite these challenges, impressive results have been reached using recurrent structures, e.g. in [63, 162].

The proposed approach resembles the windowing method, yet the function approximator is kept lean by tayloring it to a class of plants. Several methods are investigated in the following, one uses the windowed input in an RNN-like way, the other two are feedforward networks. By performing a preliminary experiment the most appropriate reconstruction method is selected, a linear filter-like layer with a constraint on the weights.

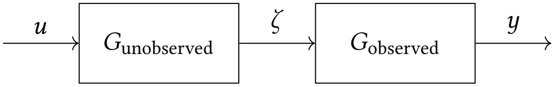


Figure 3.1.: Assumed structure of the learning problem. The plant consists of two blocks $G_{\text{unobserved}}$ and G_{observed} in a series connection. The state vector ζ of the first block is not observed; the state vector y of the second block is the plant output.

3.1.1.2. Proposed State-Action Value Function Approximator Structure

This work proposes an architecture for the state-value function approximator for a special class of partially observed systems. Two intuitions motivate its design: Actuator dynamics can be reconstructed from input history and the optimal value function structure for the fully observed case. The design therefore combines a reconstructing filter whose output is fed to a subsequent function approximator. Since both filter and subsequent function layer are part of the state-action value function approximator, both learn using standard TD learning.

This section begins with the optimal state-action value function for the fully observed case, then introduces three candidates for the reconstructing filter and explains how to implement them.

For the linear-quadratic regulator (LQR) setting, i.e. a linear controller $\rho_{\theta_{\mathrm{c}}}$ for a linear, fully observed, plant with a quadratic reward function (cf. (1.7)), the true state-action value function is known to be quadratic [19]:

$$\hat{Q}^{\rho}_{W}(x,u) = \begin{bmatrix} x^\top & u^\top \end{bmatrix} W \begin{bmatrix} x \\ u \end{bmatrix} \tag{3.1}$$

with W a quadratic matrix of appropriate size.

For the longitudinal speed control problem (see Fig. 1.2) the full state is not available. The dynamics are therefore split in two groups: an unobserved system containing actuators and resistances accounting for the acceleration and integrator returning the speed, i.e. an observed system[29], see Fig. 3.1.

[29] Note that this division is strict if considered in continous time, but not in discrete time: in continuous time the integrator is solely fed acceleration, but the discrete-time form uses small components from other unobserved states due to matrix exponential/numeric integration. The error introduced is small if the time step size Δt is small.

The next step towards the state-action value function structure is to build a learning observer for the unobserved portion of the state to be used in combination with the quadratic structure. The estimation can occur from two perspectives: from a causal perspective the unobserved dynamics is before the observed part and could therefore be inferred from (past) actions, we call this view the integrator perspective. Another option would be to reconstruct state vector entries from the output history, which will be referred to as differentiator perspective. Since the plant dynamics is slow compared to the sampling time, successive actions from an integrator point of view are unlikely to have vastly different impact, i.e. the influence of past actions on the current state estimate to is expected to develop continuously over time. In the differentiator perspective, approximate derivatives, i.e. finite differences of the measured outputs, are weighted over time. Here the impact of derivatives is assumed to evolve in a continuous way over time. In practice, these perspectives are difficult to separate since the actions are computed based on output measurements[30].

In the remainder of this section three reconstruction concepts following the integrator perspective[31] are proposed:

1. a weighted sum of past actions, i.e. a finite impulse response (FIR) filter with regularization (published in [119]),
2. a FIR filter with smoothed coefficients and regularization (published in [118]) and
3. a linear recurrent layer (published in [118]).

In the following, concept 1 is referred to as 'normalized' layer, concept 2 as 'gaussian' layer and to concept 3 as 'rnn' layer. All presented approaches assume the state vector to contain the current speed measurement y_t and a window of h_u past actions $\begin{bmatrix} u_{t-h_u} & \dots & u_{t-1} \end{bmatrix}^\top$:

$$s_t = \begin{bmatrix} y_t & u_{t-h_\mathrm{u}} & \dots & u_{t-1} \end{bmatrix}^\top . \tag{3.2}$$

Despite the plant having likely more than one unobserved state dimension, the proposed structure worked best by learning to reconstruct only a single

[30] Exploration noise plays a crucial role in breaking the correlation between measurements and inputs, see Section 2.2.5 and Section A.3.4 in the Appendix.

[31] The idea behind following the integrator perspective is that differentiation can potentially amplify noise.

(i.e. one-dimensional) state. This may be due to issues with learning reconstruction using a large number of parameters or difficulties at discerning between different hidden states[32].

Approach 1 can be implemented as a fully connected feedforward layer (see Section 2.1.2) with h_u inputs and 1 output. It adds only h_u parameters to the network. FIR filter (see e.g. [95] for an introduction) usually become more accurate with increasing length. For the system at hand this gives a good rule of thumb for tuning the filter length: it should be chosen according to the impulse response of the system. Increasing the filter beyond a certain value yields little accuracy gains and makes learning more difficult due to the increasing dimension of the parameter space. An example is given in appendix A.3. In order to remove unnecessary degrees of freedom[33], weight normalization [127] is used to fix the norm of the filter weight vector to 1. This does not affect the approximation power of the approximator as following layers can compensate for scaling errors.

The proposed FIR filter is designed with the integrator perspective in mind and is less apt to work with the differentiator perspective, since it cannot simultaneously compute all possible finite differences of its inputs and weight them. However, the proposed structure could approximate this by learning gains of alternating signs with magnitude according to the influence of the respective finite difference. This would be equivalent to skipping every second finite difference. In our experience the learned set of FIR parameters is usually a mixture of the (smoother) impulse response (see Fig. 3.5 in the next section for a simulated example) and an alternating series when applied to the real vehicle. Concept 2 therefore is designed to enforce smooth FIR coefficients

[32] Note that it is difficult to find a ground truth for the reconstructing layer if more than one system state is unobserved. Reconstructing only one replacement signal for unobserved states requires learning a blended signal. The ratio, however, depends on the distribution of the training data and aims to compensate for missing monomial multiplication in the subsequent layer.

[33] Another method for limiting unusable degrees of freedom in an approximator that uses an FIR filter for reconstruction is to fix a weight in the linear layer after it.

by reducing the number of learnable coefficients from h_u to n_weights and interpolating between them using gaussians[34]:

$$w_{i,\mathrm{smoothed}} = \sum_{j=1}^{n_\mathrm{weights}} w_{j,\mathrm{train}} \exp\left(\frac{(j-i)^2}{\sigma}\right) \tag{3.3}$$

with σ being a hyperparameter we choose to be 1, and $i = 1, ..., h_\mathrm{u}$. Each training step only updates the meta parameters w_train, but then the parameters w_smoothed are calculated and used for evaluation. This is done by including the derivative of (3.3) in the backpropagation algorithm. Despite this solution, the meta-parameters w_train may still be learned as a series with alternating signs, effectively only forming finite differences over a longer time interval.

Concept 3 also reduces the number of parameters by feeding the appended actions to a linear recurrent elman network structure [35] in chronological order[35].

The proposed estimators are combined with a quadratic function approximator. Fig. 3.2 gives a schematic overview for the implementation of the FIR filter approach 1 with the quadratic function approximator. The reconstructing layer can also be combined with other structures after it, e.g. fully connected networks.

3.1.1.3. Simulation Results

This section aims to prove the intuition that the proposed reconstruction method is both effective at reducing the variance and can be modularly used in different approximator architectures. The proposed approximator structure configuration 1 is applied to the example system (1.3) to show that it is able to reduce the variance in the learning process. This section shows that other approximator structures are viable by using a fully connected network instead of the quadratic approximator at the cost of slightly slower learning and reduced accuracy.

[34] This is motivated by how radial basis function are used to approximate smooth functions with a limited set of parameters.

[35] This interpretation of parts of the state vector as trajectories avoids the issues of RNNs with experience replay, but potential issues for training RNNs known as exploding/vanishing gradient persist (see e.g. [111, 48]).

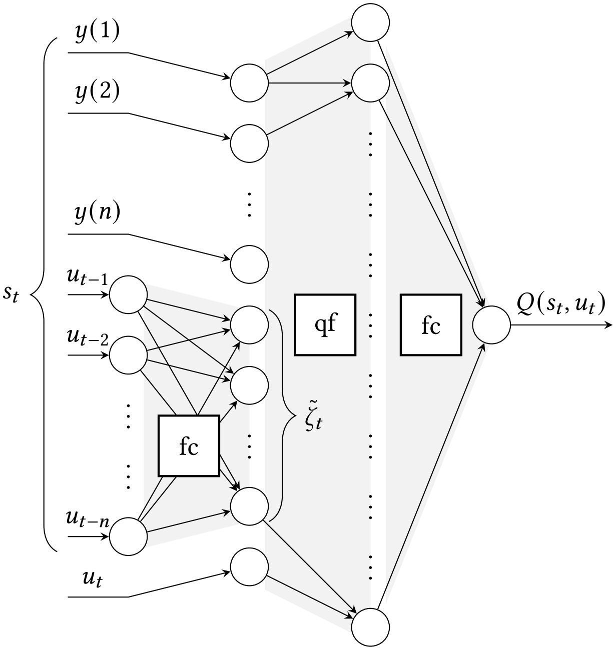


Figure 3.2.: Schematic of the proposed network architecture from [119]. The first layer passes the measurement y_t (here a version with more than one measured quantity is shown), the reconstruction output[32] $\tilde{\zeta}$ and the current action u_t to a second layer without learnable parameters that computes all monomials up to degree 2 of its inputs (qf). The third layer outputs a weighted sum of its inputs (fully connected, fc) that is learned to approximate the state-action value. While the reconstructing layer can be any of the presented ways, in this schematic reconstruction according to concept 1 in the form of a fully connected layer (fc) is used.

To highlight the effectiveness of the proposed approach this section compares two configurations of the deterministic actor-critic architecture. One uses a quadratic function approximator for the state-action value function, treating the measurement as full state, i.e. ignoring the unobserved dynamics. We mark this configuration as ’ignore’. The other configuration uses the proposed FIR layer for reconstruction; this configuration is marked as ’reconstruct’. The training occurs in episodes of 200 time steps each, with the plant inititalized in random states[36] and the control target is 0. Training is paused if either the buffer of past actions or the experience storage are not filled to capacity. Every 10 episodes a validation run is performed, i.e. training is paused, no exploration noise applied while the controller tries to drive the system from a fixed initial point $x_0 = \begin{pmatrix}1 & 0 & 0\end{pmatrix}^\top$ to zero in order to monitor the learning progress by the achieved state value $\tilde{V}(x_0)$ of this point[37]. This experiment was presented in [119] and used a fixed parameter in the final layer of the function approximator instead of weight normalization to eliminate the unused degree of freedom. The hyperparameters can be found in Table A.1, set A in the appendix A.4.

A comparison of TD errors during training is given in Fig. 3.3. It shows that the reconstructing state-action value function helps reducing the TD-error in the steady state.

The reduction in variance is visible from Fig. 3.4 which shows the approximated state value computed every 10th run. Not only is learning more precise, i.e. the performance exhibits less variance throughout the learning process, but also faster. Note that an output controller is not always optimal throughout the entire state space. It may therefore be misleading to judge performance from a single state value.

In order to prove the flexibility of the proposed approach, the proposed combination of FIR-like reconstructing layer and quadratic function approximator is compared with a combination of FIR-like normalized reconstructing layer and a shallow fully connected network. Fig. 3.5 is a plot of the FIR coefficients at the end of the training. Both approaches learn a similar distribution, with the quadratic form exhibiting less variance.

3.1.1.4. Comparison of Recurrent Layer, Interpolated Weights and Fixed-Norm FIR in the Vehicle

When applied to a real vehicle, the three proposed reconstruction methods exhibit different behavior. This comparison was published in [118]; see

[36] Episodic training is common in RL, since regular resets to random states help compensate for poor performance of the agent and help with exploring the state space [145, Section 5.2].

[37] A truncated version of the state value is used for computational feasibilty. We truncate the infinite sum once the norm of the state falls below 10^{-4}.

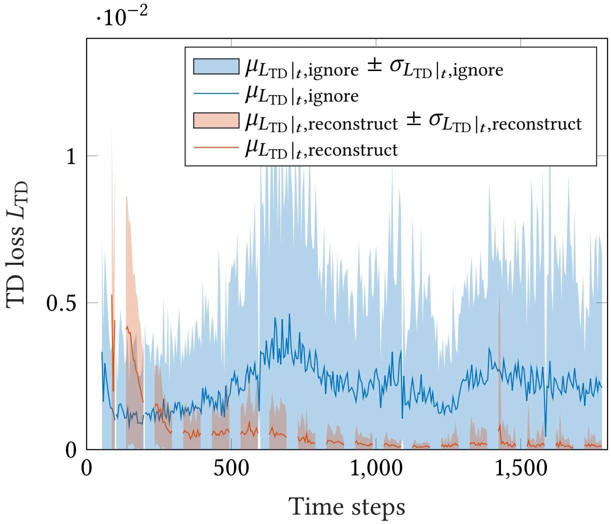


Figure 3.3.: Mean (μ) and standard deviation interval ($\mu \pm \sigma$) of mean squared TD error L_{TD} while learning a state-action value function for a fixed policy plotted over training episodes from [119]. The experiment was performed 50 times. The proposed reconstruction method lowers the steady-state TD error. The experiment was conducted in an episodic setting, which causes the state augmentation buffer to be purged in reqular intervals, halting learning. Since the buffer for state augmentation takes longer to fill than the buffer for the TD error, the interruptions of the learning are only visible for the reconstructing value function.

Chapter 4 for more details on the experiment setup and Table A.1, set C, for hyperparameters. Fig. 3.6 shows the progression of the controller parameter over training time. Both gaussian weights layer and normalized FIR yield fast convergence and stay in a small region when converged. The RNN-based variant exhibits higher variance and takes longer to converge. For simplicity, this work therefore focuses on the normalized FIR reconstruction method.

For the MF approach, this rounds off the proposal for solving the challenge of a partially observed environment. Next, we turn our attention to tracking a time-varying target.

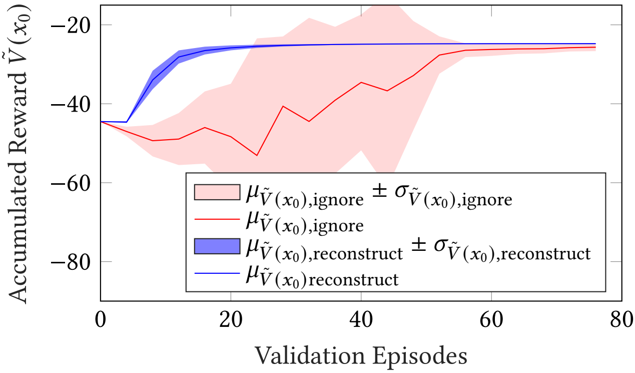


Figure 3.4.: Mean μ and standard deviation interval $\mu \pm \sigma$ of approximated undiscounted and truncated state value $\tilde{V}(x_0)$ of a fixed initial state x_0 over training episodes from [119] averaged over 50 runs. The sum was truncated once the norm of the state vector falls below 10^{-4}. The reduced variance when reconstructing unobserved states translates into more accurately learned control parameters and thus more steady performance, especially when converged. Better performance in a single state does not necessarily translate to better overall performance in optimal output control.

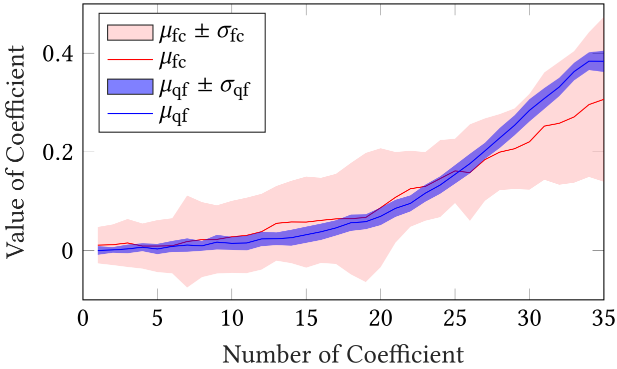


Figure 3.5.: Mean (μ) and standard deviation interval ($\mu \pm \sigma$) of learned weights in the normalized FIR filter when paired with a shallow fully connected layer ('fc') compared to a combination with a quadratic function approximator ('qf') averaged over 50 runs from [119]. Both roughly retrace the (inverted) impulse response of the unobserved dynamics apart from scaling effects[32], but the coefficients learned in combination with the quadratic function approximator exhibit less variance. The pattern of the FIR coefficients suggests that a longer FIR filter would not yield benefits when learning with this environment as the coefficients approach zero for actions close to the end fo the window of past actions.

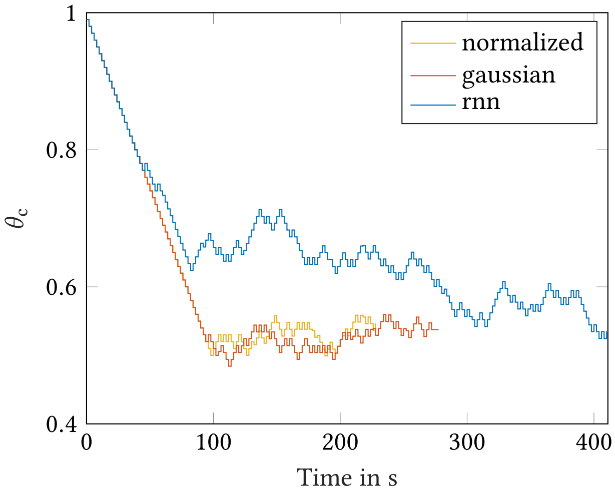


Figure 3.6.: Actor weights for different reconstruction methods learned in a real car from [118]. The controller parameter is updated only every 2 s resulting in the step-line-like graph and a maxnorm constraint in the optimizer causes the linear slope on the way to the steady state. For both the gaussian weights (marked as 'gaussian') and the normalized FIR layer ('normalized') the controller gain stabilizes around 0.5. The variant using an RNN cell for the reconstruction converges later and exhibits higher variance.

3.1.2. Tracking Control and Preview Compression

It is difficult to learn a tracking controller (challenge 1b) using RL: This is because arbitrarily changing target values make it impossible to accurately estimate the expected return if the target is not known on an infinite horizon. In engineering applications, this information is usually not available.

For the application at hand, the challenge can be decomposed in two subproblems:

- the target is known only on a finite horizon and
- it may change arbitrarily, but slowly to maximize passenger comfort most of the time.

Including a preview helps to predict the state-action value, but bloats the input space of the function approximator[38]. This may enable learning with a moving target, but makes training costly and difficult. Since the target changes only slowly, the entries in the prediction horizon exhibit only minor differences. This makes it difficult to differentiate between the influences of individual entries on the overall value, which further slows the learning progress, i.e. requires small learning steps and large batches.

The proposed solution answers these issues: It projects the trajectory preview on a low-dimensional polynomial. This simultaneously reduces the state dimension and allows extrapolation to an infinite horizon[39]. The proposed projection is combined with a set of methods for training data manipulation providing for compliance with the Markov assumption, without limiting the proposed approach to trajectories generated by a time-invariant external system. Using off-policy learning, one variant of the proposed manipulation methods allows to replace the noise in the action space with noise on the target, simultaneously adding flexibility and a method for guiding the agent during learning. This subsection is mostly based on [81], where the proposed method is presented in a general way and applied using linear polynomials, i.e. an acceleration feedforward. Some parts stem from [119], which uses a constant polynomial, i.e. no feedforward controller, but highlights some of the aspects on exploration.

The following section is structured as follows: First, this work is put in the context of literature on tracking using RL. Then, the proposed projection and training data manipulation method are presented. Using simulation, it is shown that the target can account for exploration instead of directly adding noise to the action. Finally, the simulated example (1.3) is used to prove that the proposed training data manipulation allows to significantly reduce variance during learning.

[38] Even for short preview lengths the number of points is large due to the short sampling interval. The naive approach to use a more coarse grid can be seen as a special case of the presented approach.

[39] The infinite horizon is helpful for low-variance estimation, yet discount limits the effective horizon.

3.1.2.1. Literature on Tracking using RL

The following briefly reviews literature on RL for tracking control. For a more in-depth discussion see [118, 83, 82]. Tracking control signifies following a varying control target, yet a feedback-controller reacting solely to the deviation of the system state from the desired state (see e.g. [107, 157, 119]) often lags behind the desired trajectory.

If the trajectory is known at training time and does not vary in the use case, a learning algorithm can be trained to follow a specific trajectory [166], but has to be re-trained each time the trajectory changes.

A slight generalization is to assume the trajectory stems from an exogenous system with constant dynamics (see e.g. [78, 103, 80, 94, 77]). Since these dynamics are perceived as part of the environment by the learning algorithm, standard RL methods can be applied. However, re-training is necessary as soon as the trajectory, i.e. the related exogenous system changes. The proposed algorithm can be seen as an extension to this approach, allowing for arbitrary trajectories thanks to the training data manipulation.

More general methods follow the idea of universal value function approximators [129], which aim to return a state(-action) value conditioned on a target, allowing to learn tasks with different objectives. In their work, however, the target is assumed to be a time-invariant goal state. For time-varying targets, i.e. trajectories, a major challenge is their representation, especially since they are usually only partially known.

Incorporating preview can be done by explicitly including a finite window of length h_y of reference values $\hat{y}_t, \ldots, \hat{y}_{t+h_\mathrm{y}}$ in the state vector. Since the Q-function for TD-learning is defined on an infinite horizon, an assumption has to be made on the continuation of the profile, e.g. assuming constant values beyond the provided horizon [83]. It may however be difficult to make a good choice in h_y, since longer previews reduce bias but bloat the input space, i.e. increase the number of critic weights. This is prevented by using a low-dimensional polynomial projection as proposed in [82] in this work.

Our proposed algorithm therefore combines feedforward, arbitrary trajectories and limited preview with low variance during learning.

3.1.2.2. Proposed Method for Integrating Feedforward in the RL Algorithm

The proposed approach consists of two parts:

1. The state vector is augmented with a vector of polynomial coefficients obtained by projection,
2. Compatibility is enforced on the corresponding entries in the state vectors of the experience tuple for training. This is equivalent of using an approximate optimization goal or hiding unexpected target dynamics from the algorithm.

The theoretical and more general foundation for this method has been published in [82].

These components are reflected in the layout of this section: It begins by encoding a set of preview values for the target as polynomial coefficients, then substitutes the performance measure (1.7) with an approximate target that assumes the task is to follow the approximate polynomial forever[40]. This substitution is done through a manipulation of the training data. This section is rounded off with an overview of the complete algorithm.

Polynomials are a common method to compress or encode information[41], often used in the form of orthogonal polynomials (or "Polynomial Chaos", see e.g. [164] for an introduction), as they allow to flexibly and accurately reflect continuous functions with a small number of parameters. In the following polynomials are used to reflect a preview from a finite horizon using polynomial coefficients. The preview $\hat{y}_{t+\tilde{t}_t}$ is assumed to be given at constant offsets relative to the current time. While these preview points do not have to be equidistant, for simplicity here it is assumed that the preview is sampled at the system sample rate. The offset[42] from the current time step t can therefore be denoted using a discrete index $\tilde{t}_t$.

[40] This assumption allows to formulate the value function as an infinite sum, which can be bootstrapped in TD learning.

[41] Encoding using a combination of basis functions weighted by parameters is popular throughout many domains, e.g. cosine projection in the popular JPEG image compression format (see e.g. [68]). Other approximation methods that allow for propagation similar to (3.8) are possible in our algorithm, too [82].

[42] Note that the local "time" coordinate frames $\tilde{t}_i$ may be arbitrarily rescaled according to numerical needs.

We employ a base set of polynomials ϕ, that allows to recover approximate target values $\tilde{\hat{y}}_t^{(\tilde{t}_t)}$ valid for time step $t + \tilde{t}_t$ from a set of coefficients $p_t = \begin{bmatrix} p_{0,t} & p_{1,t} & p_{2,t} & \dots & p_{n_\mathrm{p},t} \end{bmatrix}$ obtained at time step t:

$$\tilde{\hat{y}}_t^{(\tilde{t}_t)} = p_t \phi(\tilde{t}_t) \tag{3.4}$$

with e.g. $\phi(\tilde{t}_t) = \begin{bmatrix} 1 & \tilde{t}_t \Delta t & (\tilde{t}_t \Delta t)^2 & (\tilde{t}_t \Delta t)^2 & \dots & (\tilde{t}_t \Delta t)^{n_\mathrm{p}} \end{bmatrix}^\top$ and $p_{0,t}$ denoting the constant term in the approximating polynomial resulting from projecting the preview available at time step t, $p_{1,t}$ the coefficient for the linear term etc. up to polynomial order n_p.

The assumption of constant samples allows for efficient encoding using a time-invariant projection matrix[43]

$$P_\mathrm{project} = \left(\begin{bmatrix} \phi(0) & \phi(1) & \dots & \phi(n_\mathrm{p}) \end{bmatrix}^\top \right)^+ \tag{3.5}$$

with $(\cdot)^+$ being the operator for a pseudo inverse, e.g. by singular value decomposition. Then, the projected polynomial parameters can be retrieved using the vector-matrix product

$$p_t = \begin{bmatrix} \hat{y}_t & \hat{y}_{t+1} & \hat{y}_{t+2} & \dots & \hat{y}_{t+h_\mathrm{y}} \end{bmatrix} P_\mathrm{project} \tag{3.6}$$

In the following the polynomial that encodes the reference trajectory is called reference polynomial.

In contrast to the limited horizon preview, the reference polynomial is defined on an infinite horizon[44] in $\tilde{t}_t$. Its coefficients are appended to the state vector

$$s_t = \begin{bmatrix} y_t & u_{t-h_u} & \dots & u_{t-1} & p_{0,t} & \dots & p_{n_\mathrm{p},t} \end{bmatrix}^\top . \tag{3.7}$$

See Section 3.1.1 for the augmentation with previous actions.

Now an optimization goal is defined on a surrogate state value that is based on the assumption that the reference polynomial is followed infinitely.

[43] Other methods for obtaining polynomial coefficients, e.g. projection on orthogonal polynomials, are viable, too.

[44] It is assumed that either the extrapolated values resemble the not yet known trajectory, reference values in the far future do not affect the currently optimal choices or that the problem is discounted enough to render extrapolation errors insignificant.

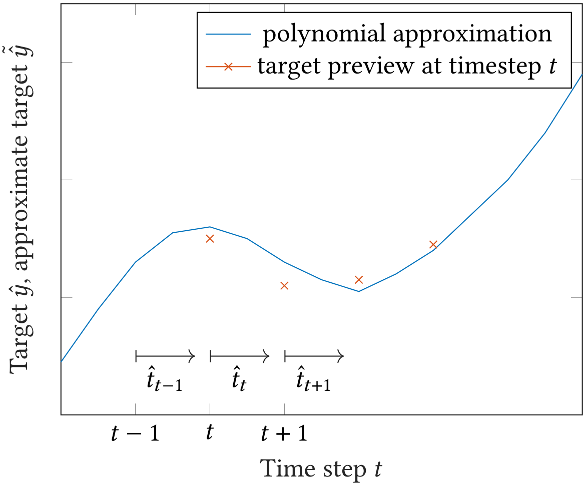


Figure 3.7.: Qualitative example for a polynomial approximation of the target trajectory. The original control target is given by a finite set of preview points starting in the current time step t. By projecting on a polynomial, a slight approximation error may be introduced and the domain is extended to infinity. The reference polynomial can be expressed in different time coordinate frames, e.g. $\tilde{t}_{t-1}$, $\tilde{t}_t$ or $\tilde{t}_{t+1}$, to ensure coherent state vector definition for each time step.

Since the reference polynomial is defined relative to the current time step t, a transformation T_{shift} to subsequent time steps is necessary, e.g. $t+1$ in order to use the same reference polynomial in future state vectors: a shifted reference polynomial allows to define the state vector s_{t+1} according to (3.7), i.e. with an index $\tilde{t}_{t+1}$ starting at $t+1$. This transformation ensures coherence of the state definition by transforming the reference polynomial to coordinate frames of interest, e.g. $\tilde{t}_{t+1}$. An exemplary trajectory is shown in Fig. 3.7.

For equidistant steps, polynomial parameters can be transformed using a constant linear operation

$$p_t^{(1)} = p_t T_{\text{shift}} \tag{3.8}$$

with $p_t^{(1)}$ indicating that the polynomial coefficients p_t have been shifted from a time coordinate frame $\tilde{t}_t$ starting at t to $\tilde{t}_{t+1}$ starting at $t+1$. This linear transformation is equivalent to applying a time-invariant dynamics of an exo-system to the respective part of the state vector and depends on the choice of base polynomials. By equating the coefficients of $\sum_{i=0}^{n_\mathrm{p}} p_{i,t}^{(0)} \left((\tilde{t}+1)\Delta t\right)^i = \sum_{i=0}^{n_\mathrm{p}} p_{i,t}^{(1)} \left(\tilde{t}\Delta t\right)^i$ the transformation matrix[45]

$$T_\mathrm{shift} = \begin{bmatrix} \binom{n_\mathrm{p}}{0}\Delta t^0 & \binom{n_\mathrm{p}}{1}\Delta t^1 & \binom{n_\mathrm{p}}{2}\Delta t^2 & \cdots & \binom{n_\mathrm{p}}{n_\mathrm{p}}\Delta t^{n_\mathrm{p}} \\ 0 & \binom{n_\mathrm{p}-1}{0}\Delta t^0 & \binom{n_\mathrm{p}-1}{1}\Delta t^1 & \cdots & \binom{n_\mathrm{p}-1}{n_\mathrm{p}-1}\Delta t^{n_\mathrm{p}-1} \\ \vdots & & \cdots & & \vdots \\ 0 & \cdots & \cdots & 0 & \binom{0}{0}\Delta t^0 \end{bmatrix} \tag{3.9}$$

can be derived.

In the real use case, however, the target will change arbitrarily and the reference polynomials obtained from approximating the preview points at time steps t and $t+1$ may not fulfill the relation (3.8), i.e. $p_t^{(1)} \neq p_{t+1}$. We suggest to hide these incoherences from the learning process by manipulating the data used for training the state-action value function[46], i.e. learning the surrogate value function

$$\bar{V}^\pi(y_t, p_t) = \sum_{i=t}^{\infty} \gamma^{i-t} R\left(y_t, \tilde{\hat{y}}_t^{(i-t)}, \pi\left(y_i, p_t^{(i-t)}\right)\right) \tag{3.10}$$

or a correspondingly defined state-action value function. When evaluating the TD error, the algorithm uses the same reference polynomial for augmenting the state vector s for both time steps t and $t+1$, but shifts it to match the difference in time according to (3.8). Two approaches are possible here:

- Substitute the reference polynomial $p_t^{(0)}$ for time step t with a reference polynomial $p_{t+1}^{(-1)}$ in accordance with time step $t+1$. We call this manipulation method M_t.
- Substitute the reference polynomial $p_{t+1}^{(0)}$ for time step $t+1$ with a reference polynomial $p_t^{(1)}$ fitting the reference polynomial for time step t. We call this manipulation method M_{t+1}.

[45] Despite not being part of the publication, this form was developed jointly with Florian Köpf for [81].

[46] Value functions with encoded preview are still quadratic if the environment is fully observed and the controller is linear [82], therefore the intuition behind the value function approximator introduced in Section 3.1.1 still holds.

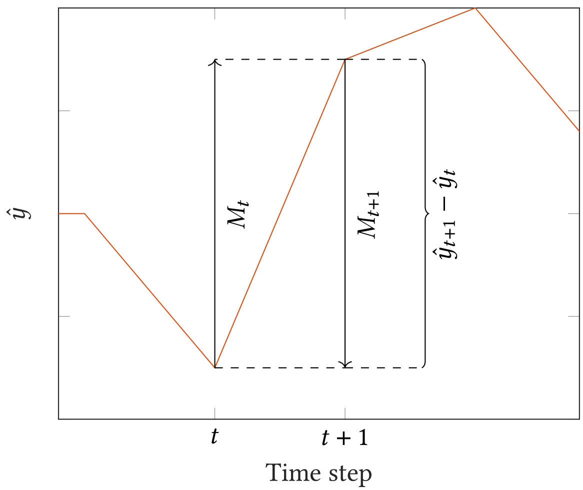


Figure 3.8.: Example for manipulation of the target according to methods M_t and M_{t+1} for a zero-order reference polynomial adapted from [118]. The rationale behind both manipulations is to hide the target change $\hat{y}_{t+1} - \hat{y}_t$ from the learning algorithm. For learning, in the tuple covering time steps t and $t+1$ either the target at time step t can be changed to match the target at time step $t+1$ (manipulation M_t) or the target at time step $t+1$ can be changed to match the target at time step t (manipulation M_{t+1}). For this example with a constant approximation of the target, both result in equal target values for time steps t and $t+1$. Note that this manipulation is done solely to the tuple $\langle s_t, u_t, s_{t+1} \rangle$ used for training the state-action value function. Controller evaluation is unaffected and data for other time steps is manipulated independently, such that the manipulated target and the target the policy is evaluated with do not drift apart.

See Fig. 3.8 for an example. We mark state vectors that contain a manipulated target as $\bar{s}$. Since the reward depends on the control target, these manipulations entail recalculating the reward. The manipulations are solely and indepentently done in the tuples used for training the state-action value function, e.g. before storing them in the experience storage. In our experience the actual and the manipulated target are often only slightly different, enough to move exploration to the target in some cases (see Section 3.1.2.3), but still close enough to learn a meaningful state-action value function.

The important goal of preemptively acting on target changes, i.e. before they affect the control error, can be achieved by learning a policy that uses the coefficients of the reference polynomial as its features. Note that for the policy, no manipulation of the target is applied, such that it accepts truly arbitrary targets as its input. In our case, we use a linear policy (1.5) to benefit from the linear-quadratic architecture in our state-action value function.

Next, a short example from [118] is presented to show the variance reduction capabilities of the proposed manipulation. For this, the system[47] (1.3) and a random target[48] is used. The target is given with one time step preview, i.e. just the target for the next time step is provided[49]. Two training runs are performed to compare the algorithm's behavior with training data manipulation M_{t+1} and without any training data manipulation. The hyperparameters are given in Table A.1, set B in the appendix. Fig. 3.9 shows the progression of the TD error during training. The proposed manipulation effectively reduces the TD error, suggesting that variance is lowered significantly during training. Lower variance helps faster and more precise controller learning, as Fig. 3.10 shows.

3.1.2.3. Shifting Exploration between Target and Action

Manipulating the target can be advantageous for RL, as we show in the following.

Using manipulation M_t on the tuple $\langle s_t, u_t, s_{t+1} \rangle$ changes s_t such that the action u_t (calculated from s_t prior to manipulation) would become off-policy even if no exploration noise had been applied. Theoretically, this may allow the agent to learn without additional exploration noise. In practice, however, it may be necessary to choose a target profile that forces the controller to excite the system, and to ensure that the controller is aggressive enough to translate the excitation in the target to the plant (see Section 2.2.5 for purposes of exploration noise).

Experiments with several configurations of exploration noise are performed using the system[47] (1.3):

[47] These experiments were performed without adding the delay included in system (1.3), i.e. with a reduced system order.

[48] The target is chosen randomly over time, but in a coherent way, i.e. we stick to what we gave as a preview but choose randomly how to extend it in subsequent time steps.

[49] The projection on a constant polynomial is trivial in this case.

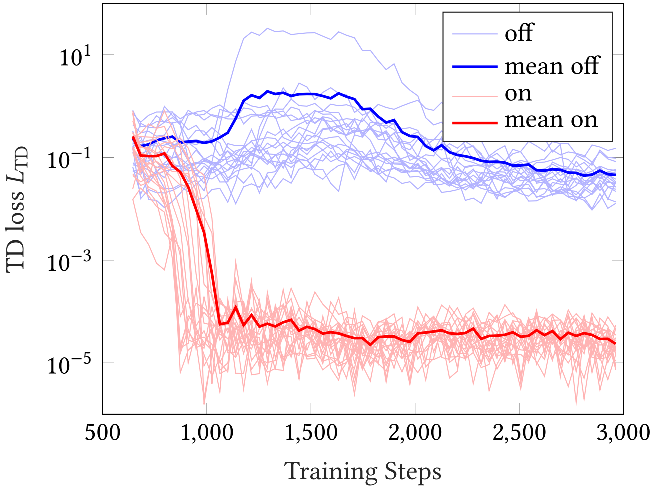


Figure 3.9.: Mean squared TD loss over training steps for configurations using manipulation M_{t+1} (marked as "on") vs. no manipulation (marked as "off") from [118] for multiple training runs. The agent training on the manipulated data exhibits TD errors several magnitudes lower than the agent training on the non-manipulated data.

- A baseline configuration that uses exploration noise[50] added to the controller output (we mark this as "ex. c") is compared
- against a configuration without exploration noise on the controller output, but with the random component added[51] to the target profile (marked as "ex. t"),
- against a configuration that has uniform exploration noise on both controller output and target (marked as "ex. c,t"), and
- against a configuration without any additional noise (marked as "-").

This experiment uses the hyperparameters in Table A.1, set B, in the appendix, but varied the exploration as described. The evolution of the controller gain during training is diagrammed in Fig. 3.11. Applying exploration noise to both

[50] We use a random number sampled every 100 time steps from a uniform distribution between -1 and 1 as exploration noise in this example.

[51] The target profile consists of an accumulation of a uniform random number sampled between -0.1 and 0.1 at every time step.

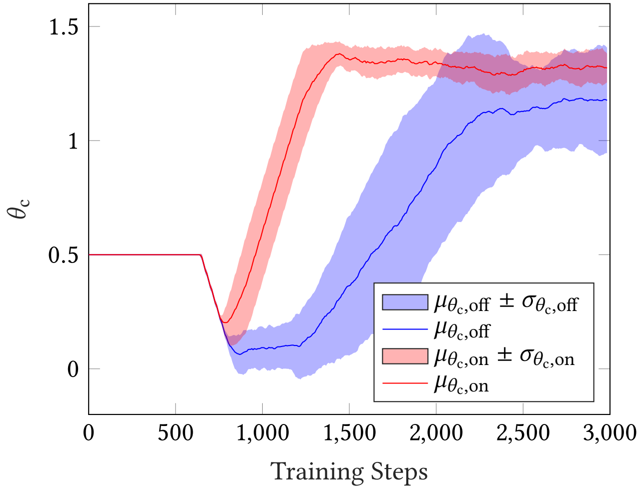


Figure 3.10.: Mean μ and standard deviation interval $\mu \pm \sigma$ of controller gains θ_c over training steps for a configuration using manipulation M_{t+1} (marked as "on") vs. a configuration training on non-manipulated data (marked as "off") from [118] for multiple training runs. The agent training on manipulated data exhibits lower variance and converges faster. One or both solutions are biased.

target and action ("ex. c,t.") yields least variance during training, applying none to both can make the learning process fail, i.e. the controller gain diverge. Both variants that have noise either on controller output or target learn, but have higher variance in comparison and therefore learn slightly slower. It is therefore advisable to be careful when foregoing the use of exploration noise.

The proposed modifications could be even extended beyond what has been presented here: arbitrary many modified variants of a state transition tuple could be used in the training data set as long as the targets within the tuple are made to comply with each other. This is possible as the proposed algorithm learns off-policy and explicitly includes a notion of the control target in the state-action value function (cf. Universal Value Function [129]). By assuming different targets, the agent can learn different state-action values from a single state transition tuple if the reward function is available, therefore efficiently

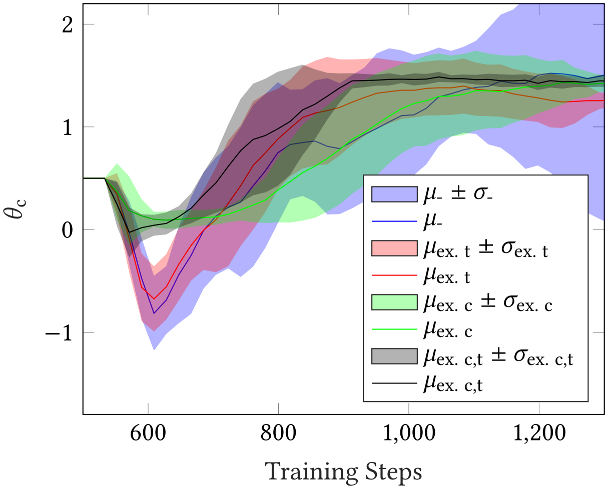


Figure 3.11.: Mean μ and standard deviation interval $\mu \pm \sigma$ of controller gains during training with the example system (1.3) with different exploration noise configurations from [118]. When applying no exploration noise to either target or action (marked as "-"), the learning process may fail. With exploration noise on either target (marked as "ex. t") or controller (marked as "ex. c"), learning succeeds. Best results are obtained with noise on both (marked as "ex. c,t"), where learning occurs quickly and with minimum variance.

learning the assumed target dynamics (3.8). This idea has become popular from the concept of hindsight experience replay [8][52].

This is not only useful for generalizing between targets, but can also be used to virtually explore target dynamics, i.e. learn quickly without actually adding noise to the target. This is possible by adding noise to the target in the training data only. This virtual target noise can be applied before applying either of the manipulations M_t or M_{t+1} directly on the reference polynomial coefficients and helps explore the exogeneous system dynamics (3.8). We mark state vectors with added noise to the reference polynomial coefficients as $\check{s}$. Virtual target noise can lead to faster learning, but may skew the distribution of observed targets from the intended use case.

[52] In [8] the authors give an example about missing a shot in a Hockey game: instead of simply discarding the actions taken prior to the miss as worthless, an agent could learn that if the goal had been in a different position, there would have been a reward.

3.1.3. Summary

This section presented an algorithm based on the (deep) deterministic actor-critic architecture. The main contributions are a state-action value function that allows learning with the partially observed dynamics of the powertrain and a manipulation of the training data that allows learning a feedforward with an arbitrarily changing control target.

For this, the state vector is extended: it now consists of coefficients of a reference polynomial, measurements of the plant output and past actions. The state-action value function approximator uses a learned FIR-filter-like layer to reconstruct missing state information and combines polynomial coefficients, reconstructed state information, output measurement and the action in a quadratic function that yields a low-variance estimate of the state-action value. The proposed method employs a projection on polynomial parameters, optional virtual target noise and a training data manipulation to learn tracking control with little variance.

The proposed algorithm is partitioned in three parts (see Fig. 3.12 for a schematic):

- The **evaluation** step assembles the state vector s_t from output measurement y_t, past actions $u_{t-h_y}, \dots, u_{t-1}$ and target $\hat{y}_t$ (and possibly additional preview values). It computes the current policy's response u_t and applies this action to the system. The current policy is updated using parameters θ_c from the training step. State vector s_t along with current action u_t are provided to the subsequent data preparation step.
- **Data Preparation** is fed the current state vector s_t and current action u_t and outputs batches of training data $\{\langle \bar{\bar{s}}_{k-1}, u_{k-1}, \bar{\bar{s}}_k \rangle\}$ to train state-action value function approximator and policy on. This involves applying noise to the target, buffering states and action, applying training data manipulation and keeping a ringbuffer of state transitions to sample batches from.
- The **training step** is based on regular actor-critic learning after evaluating the reward function. The state-action value function is updated using TD-learning with LM optimization; the policy is updated using the Deterministic Policy Gradient and LM optimization. The training step can be run at a sample rate independent of the

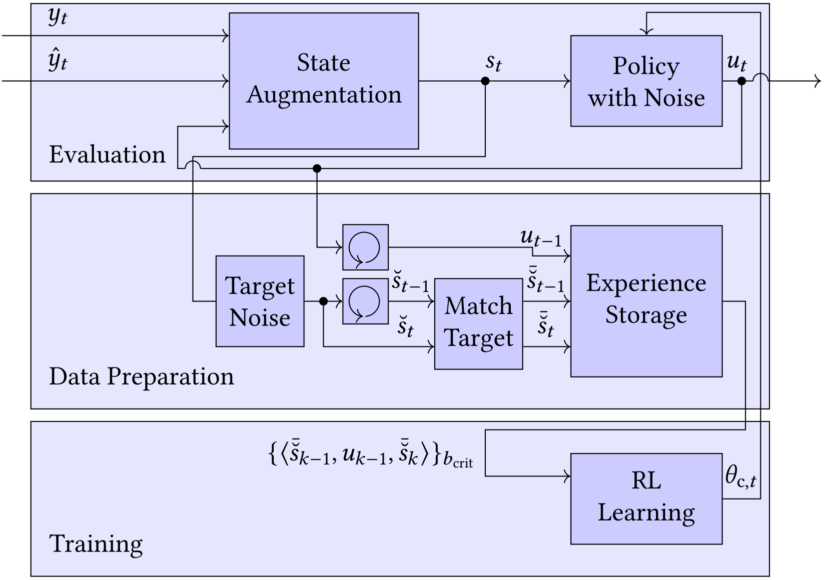


Figure 3.12.: Schematic of the proposed MF RL architecture. In addition to the MF RL architecture from Fig. 2.6 we added three blocks: a state augmentation block augments the measured input y_t with past actions and coefficients of a reference polynomial from $\hat{y}_t$. We omitted additional target preview points to avoid cluttering the overview. The target noise block adds a random component on the reference polynomial coefficients to accelerate learning of the target dynamics. The match target block ensures coherence between the targets in the state vectors $\breve{s}_t$ and $\breve{s}_{t-1}$ by applying manipulation method M_t or M_{t+1}. Note that the manipulation of the target only affects data entered to the experience storage for training. The state vector that is fed to the policy is not manipulated, therefore allowing to follow arbitrary targets $\hat{y}$. The overall algorithm can be split into (policy) evaluation, data preparation and training, which can run at independent sample times.

evaluation and data preparation step, allowing to trade off computational load and learning speed.

The proposed measures allow for learning robustly with little parameters. Chapter 4 shows that they fulfill the requirements to be applied to a real vehicle.

The next section presents another candidate algorithm that uses a learned model to search a policy with to be compared with in Chapter 4.

3.2. Model-Based RL Algorithm

This section proposes a computationally efficient MB policy search algorithm.

MB algorithms have the potential for high data efficiency, as they use the data from the real plant to learn a model that in turn can be used to generalize the experience to not yet seen portions of the state space at very low cost (see Section 2.2.6).

Partially observed systems pose the added challenge of simultaneously estimating state and dynamics, which the proposed algorithm avoids by windowing and estimating an autoregressive model with external inputs (ARX). Based on the estimated model, an optimal output feedback controller is searched and an inversion-based feedforward controller is designed automatically. With the exception of the feedforward design, the proposed algorithm has been published in [120].

This section starts by introducing our approach to model identification, then briefly touches upon our feedback control optimization using policy search and finally proposes an algorithm for automatically designing a feedforward controller.

3.2.1. ARX Model Estimation for Partially Observed Plant

Since the to-be-learned model serves as a basis for policy search, any error in the approximated dynamics may lead to a bias in the derived policy. It is therefore important to ensure approximation power to be able to reflect the true system behavior – and of course the algorithm must be able to harness these degrees of freedom. On the other hand, a too high number of learnable parameters may not only cause the learning process to overfit, e.g. to noise, but also reduce learning speed. This work tries to find an optimal balance by optimizing a criterion that considers both accuracy and complexity.

The following briefly explains ARX models and the chosen estimation method.

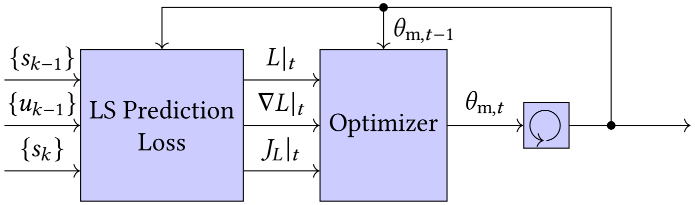


Figure 3.13.: Schematic of a model learning problem, which is a variant of the parameter estimation problem (cf. Fig. 2.1). Based on a batch of tuples $\{\langle s_{k-1}, u_{k-1}, s_k \rangle\}$ and the latest estimate of model parameters $\theta_{\mathrm{m},t-1}$ the value $L|_t$, gradient $\nabla L|_t$ and the jacobian $J_L|_t$are calculated. These are used in an optimizer to get an updated estimate of the model parameter vector θ_{m}. The process is repeated with newly sampled batches until convergence.

The following model estimation method is taken from [93, p. 176]. It is based on an ARX model that provides a prediction $\mathring{y}_{t+1}$ of the next output y_{t+1} from n current and past actions and states[53]:

$$\mathring{y}_{t+1} = \sum_{i=0}^{n-1} \left(\theta_{\mathrm{y},i}\, \mathring{y}_{t-i} + \theta_{\mathrm{u},i}\, u_{t-i}\right). \tag{3.11}$$

For ease of notation, we define the model parameter vector $\theta_{\mathrm{m}} = [\, \theta_{\mathrm{u},0} \, \dots \, \theta_{\mathrm{u},n-1} \; \theta_{\mathrm{y},0} \, \dots \, \theta_{\mathrm{y},n-1} \,]$. To minimize the one-step prediction error, the per-element loss function

$$l(y_{t-n+1}, \dots, y_{t+1}, u_{t-n+1}, \dots, u_{t+1}, \theta_{\mathrm{m}}) = \left(y_{t+1} - \left(\sum_{i=0}^{n-1} \theta_{\mathrm{y},i}\, y_{t-i} + \theta_{\mathrm{u},i}\, u_{t-i}\right)\right)^2 \tag{3.12}$$

is defined and optimized over batches of $b_{\mathrm{m}} = 200$ samples drawn from the experience storage of size 5000. LM optimization is employed and a decreasing-over-time maxnorm constraint is enforced using norm clipping (see Section A.1). This aims at keeping variance of model parameters to a minimum that would otherwise drive up the variance of the learned controller parameters. A scheme of the optimization process is shown in Fig. 3.13.

Model accuracy and model complexity are balanced by choosing the model order n using the minimum description length (MDL) criterion [142, 106],

[53] We differentiate between the model output $\mathring{y}$ and plant output y for clarity. Later on, we will use plant output values y in the place of model outputs $\mathring{y}$ to estimate model parameters.

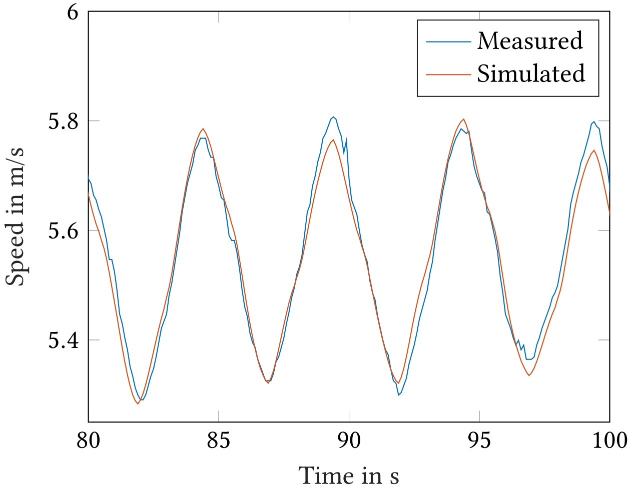


Figure 3.14.: Output of a learned model of order 20 and measured speed from [120]. At time 0 s the model was initialized with measured outputs and then simulated for 100 s using the inputs recorded in the car. The recording is of a controller trying to follow a target speed varied every 1.25 s by 1 km h^{-1} between 19 km h^{-1} and 21 km h^{-1}. The model output ('Simulated') is close to the measured speed ('Measured').

yielded a consistent optimal model order close to $n = 20$ [120] for different dataset lengths as opposed to the equally popular Akaike Information Criterion (AIC) [4, 106].

The resulting model is able to precisely predict the plant output even over extended periods as the simulation result in Fig. 3.14 suggests.

3.2.2. Sampling-Based Controller Design and Inversion-Based Feedforward Design

This section briefly states the proposed approach on sampling-based policy search. Then it describes the automated design process for the inversion-based feedforward.

The proposed algorithm updates the feedback controller through policy search using numeric gradients of values estimated using the model (3.11) introduced above. The state value is approximated by a truncated sum of rewards calculated from a simulation of model (3.11) with controller (1.5). The infinite sum (1.6) is cut off[54] once $\gamma^{i-t}R(y_i, \hat{y}_i, \pi(\hat{y}_i, y_i)) < 10^{-20}$ or the amount of summands exceeds 500. The performance gradient for the policy is computed by finite differences and is fed to an LM optimizer instance with a maxnorm constraint to update the controller weight. Gradients and state values are averaged over several initial states sampled from a ringbuffer of the last 5000 augmented state vectors gathered from the plant.

To enable high precision trajectory tracking, this work proposes to pair the learning feedback controller with an automatically designed feedforward controller with an accompanying reference filter according to the two degrees of freedom controller scheme [84, 11]. The popular flatness-based (see e.g. [52]) feedforward design requires planning in the coordinate system of the system-dependent flat output (which is unknown prior to model estimation), while stable inversion [170] constrains trajectory planning. The proposed feedforward controller and filter are automatically designed by combining a predefined Butterworth low-pass filter with an approximate inverse of the learned model [21], allowing for flexible trajectory planning in the output space at the cost of possibly suboptimal tracking performance. See Fig. 3.15 for a schematic overview of the automated design process.

The following applies the z-transform to the ARX model, fixes artifacts of estimation by correcting poles and filtering and applies approximate inversion.

The first step is to transform the estimated model (3.11) to the z-domain to yield the transfer function $\tilde{G}(z)$ and to divide the numerator in polynomials N_{mp} and N_{nmp} with stable and unstable zeros[55] respectively, i.e. zeros within/on and outside of the unit circle.

$$\tilde{G}(z) = \frac{\sum_{i=0}^{n} \theta_{\mathrm{u},i} z^i}{1 - \sum_{i=0}^{n} \theta_{\mathrm{y},i} z^i} = \frac{N_{\mathrm{mp}}(z) N_{\mathrm{nmp}}(z)}{D(z)}. \tag{3.13}$$

[54] This truncation criterion is intended to balance loss of accuracy against computational burden: with a stable controller and a constant target the expression abates over time while the computational effort does not. The contribution of the to-be-truncated portion of the sum tends to decrease with the amount of summands taken into account and eventually falls below numerical accuracy.

[55] Since zeros in the numerator turn into poles when inverting the transfer function, zeros outside the unit circle are considered unstable in this context.

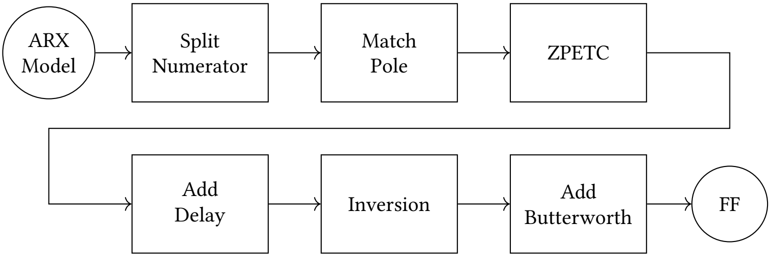


Figure 3.15.: Schematic of autoamted MB feedforward (FF) design. The design procedure takes an estimated ARX model of the plant as its input. It splits the numerator polynomial in stable and unstable zeros and ensures precise matching of the integrator pole. Then, zero phase error tracking control (ZPETC) is applied and a step of delay is added before inverting the modified model. A Butterworth filter completes the design.

Since misestimation of the integrator can yield a feedforward with nonzero steady state gain[56], the pole (i.e. zero of $D(z)$) closest to 1 is moved to exactly one, yielding $\tilde{D}(z)$. To obtain a stable inverse, N_{nmp} is replaced with $\tilde{N}_{\mathrm{nmp}}$. It is constructed according to the zero phase error tracking control (ZPETC) design (see e.g. [21]) using the zeros a_j with $j = 1 \ldots \deg(N_{\mathrm{nmp}})$ of N_{nmp}:

$$\tilde{N}_{\mathrm{nmp}} = \prod_{j=1}^{\deg(N_{\mathrm{nmp}})} (-az + 1), \tag{3.14}$$

effectively projecting unstable zeros onto stable ones while preserving the stationary gain. $\tilde{N}_{\mathrm{nmp}}$ is delayed by one time step to ensure causality. This gives

$$\hat{G}(z) = \frac{zN_{\mathrm{mp}}(z)\tilde{N}_{\mathrm{nmp}}(z)}{\tilde{D}(z)}. \tag{3.15}$$

[56] This means a feedforward controller would still command an acceleration even if the target speed is steady. Without a stabilizing feedback controller, this would cause the vehicle to continously accelerate or decelerate.

The feedforward design is completed by inverting $\hat{G}(z)$ and adding a butterworth filter $B(z)$ of order 3 (see [110]) with cut-off frequency[57] 1 Hz:

$$F_{\text{ffw}}(z) = B(z)\hat{G}^{-1}(z). \tag{3.16}$$

The accompanying reference filter results from cancelling out terms in

$$F_{\text{ref}}(z) = F_{\text{ffw}}(z)\tilde{G}(z). \tag{3.17}$$

During learning, the parameters of the feedforward F_{ffw} and the reference filter F_{ref} are adapted while they are running. With every update in parameters, their internal state vectors become meaningless, and are therefore reinitialized as if the current target y_t was their steady state to avoid the settling phase during driving. Appendix A.2 aims to give more detailed insight on this matter.

3.2.3. Summary

The proposed algorithm uses an estimated ARX model as a basis for designing a feedback controller using policy search and a feedforward controller using approximate inversion. Learning from batches along with a (dynamic) maxnorm constraint on the optimizer facilitate quick and noise-free learning purely from plant inputs and outputs, allowing to learn from a partially observed plant. The automated feedforward design is implemented in a way that circumvents pitfalls from the model estimation.

The proposed MB algorithm fits the scheme presented in Fig. 2.7: controller evaluation and data preparation are performed within the sample time Δt while the training containing model estimation and control design can be prolonged to balance performance limits against learning speed. This allows to run the algorithm online on constrained hardware.

The next chapter compares the presented MB and MF algorithms in a real vehicle across several experiment setups.

[57] Damping high frequencies in the feedforward design was introduced to limit wear on the variable valve timing assembly. Experts from BMW recommended to avoid frequencies above 1 Hz in the input to the drivetrain. In addition, damping high frequencies in the feedforward can help mitigate artifacts resulting from erroneous model estimation.

4. Validation in the Car

In this chapter, the presented MB and MF algorithms are applied to a real vehicle to demonstrate that the algorithms are capable to learn under real-world conditions and constrained resources. It shows that they deliver consistent results robustly under varying circumstances and points out limitations. The results reveal that the learning process is fast enough for application in day-to-day engineering as convergence is reached within minutes.

As a preparation step for the vehicle experiments, a simulation study presents systematic perturbations of important hyperparameters to visualize their influence and gives an intuition on why the chosen hyperparameters are deemed sensible. After that, a basic experiment setup for learning speed control in a real road vehicle is introduced. This basic experiment is conducted multiple times to showcase robustness and repeatability of learning process and result. Then factors of this experiment are varied to assess the influence of external factors on the learning results and the influence of exploration noise on the learning process and the results. See Fig. 4.1 for an overview. Throughout the presented experiments the results of MB and MF algorithms are close, suggesting the learned results are bias-free or are similarly biased. A part of the results have been published in [120].

4.1. Hyperparameter Choice

This section gives an intuition on the effect of hyperparameters on the behavior of the MF RL algorithm presented in Section 3.1. It must not be misunderstood as a strict proof for the optimality of the parameters chosen in our vehicle experiments. This is mostly due to three reasons:

- Important criteria for tuning the parameters are hard or even impossible to define formally. Variance during learning, for example, could be measured in different ways and behaves differently between

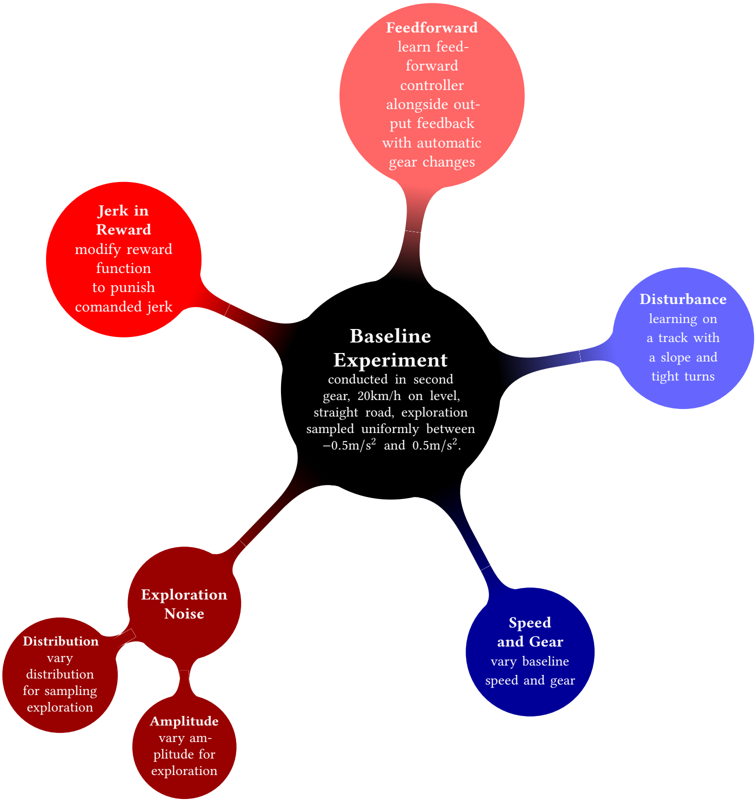

Feedforward
learn feedforward controller alongside output feedback with automatic gear changes
Jerk in Reward
modify reward function to punish comanded jerk
Baseline Experiment
conducted in second gear, 20km/h on level, straight road, exploration sampled uniformly between −0.5m/s² and 0.5m/s².
Disturbance
learning on a track with a slope and tight turns
Exploration Noise
Distribution
vary distribution for sampling exploration
Amplitude
vary amplitude for exploration
Speed and Gear
vary baseline speed and gear

Figure 4.1.: Overview of experiments. All experiments are variations of the baseline experiment in Section 4.2. Variations of the conditions outside the algorithm are marked in shades of blue: Subsequently, the disturbance level is varied by performing an experiment on a track consisting of ramps and tight turns. Experiments at different speeds (and gears) are described in Section 4.3 (dark blue). Experiments that vary components of the algorithm are highlighted in shades of red: Exploration noise configuration is changed in Section 4.4 (dark red). The reward function is modified in Section 4.5.1 (red) and a learning feedforward controller is added in Section 4.5.2 (light red).

transient phases or close to convergence. Another example could be learning speed that hinges on a definition of convergence in the learning process and a (not clear) definition of how far apart the initial parameter set and the parameters at convergence are.

- How good a hyperparameter set works depends on the system the algorithm is applied to. Since the actual goal is to learn in a real vehicle, tuning hyperparameters on an example system can help find a set of hyperparameters that works under these circumstances, but can not guarantee good performance in the real-world experiment.
- Hyperparameters may compensate each other's effect to some extent. This allows to accomodate for side effects in some choices, but often makes it difficult to choose one set over another.

The fact that hyperparameters are problem dependent and have a high impact on the algorithm's performance is known in the literature: RL algorithms may "require a high dregree of human expertise" for tuning hyperparameters that fit new applications [168]. Ambiguos performance resulting from hyperparameter tuning has sparked a debate on the reliability of performance baselines for algorithm comparison [70]. Frameworks for tuning hyperparameters in RL have started to emerge in recent years, e.g. [22, 70]. These frameworks aim to find a hyperparameter set that optimizes performance in simulation in a black-box fashion. The following is therefore intended to make the choices in this work plausible and serve as a guide to interested readers with their hyperparameter tuning tasks.

Before presenting the results of our simulation study, this section introduces an understanding of bias, variance, speed and stability, which make up the (informal) tuning goal. Then, the simulation experiment setup is presented and an overview of the hyperparameters to be varied is given.

In contrast to many publications that limit themselves to pointing out the bias-variance trade-off, this section tries to provide two additional facets to the discussion: stability and speed.

In this context, **variance** is considered a measure of how much a learned parameter fluctuates, especially (but not only) once learning has converged. This fluctuation can be understood as how much a parameter differs across different training runs, but also as oscillation or noise around a trajectory that leads the parameter towards its convergence value over training time. Generally, the goal is low variance when tuning hyperparameters. Variance can typically be traced back to the respective loss function (including the data it is evaluated on) and its gradients as well as its interactions with the function approximator. The optimizer (see Section A.1) can mitigate or aggravate this effect depending on its type and configuration. In the experiments multiple training runs are performed with each hyperparameter configuration. The output controller has only one parameter in these experiments, allowing to meaningfully visualize mean and standard deviation over training duration, which gives an intuition of the variance in policy learning. For the critic, this approach would not yield informative diagrams[58], which is why this work resorts to plotting the TD Loss (see Section 2.2.2) over training time. The TD error (2.12) is not a direct measure of variance for the critic[59], but its value tries to encode how unexpected the training data is or how good the state transitions in the training data batch match the critic's value estimation[60]. For training the actor, the Deterministic Policy Gradient is used, that, if inaccurate, can cause variance in the policy parameters. The gradient's accuracy, in turn, relies on the quality of the state-action value function approximation. The TD error can therefore be seen as an indicator for the precision of the gradient used for training the policy. Albeit having a large error margin, the TD error can therefore be be taken as a hint of how much the Deterministic Policy Gradient contributes to the variance in the actor parameters.

The following understands **bias** as the deviation of the learned parameter from the optimal/desired[61] parameter. Especially the latter definition does not allow to measure bias, since the optimal/desired value may not be available as it depends on the state distribution (see Section A.5). However, it allows to consider the parameters defining the optimization goal (e.g. in the reward function) as hyperparameters affecting algorithm performance. For example, it might be a valid choice to deviate from the actual learning objective to enhance learning behavior. Judgement of bias from the presented experiments is therefore limited: hyperparameters may well affect the optimal gain (e.g. by changing the observed state distribution), but also skew the learned policy directly. To some extent, this can be seen by comparing the performance of different hyperparameter sets, but it is difficult to factor in the effect of the

[58] Due to the high amount of parameters in a very similar range, plots of the weights over training time result in overlapping graphs that are hard to read.

[59] Changes of the TD error between learning steps can be due to noise in the parameters, too. However, there may also be other reasons, such as sampling of different batches.

[60] A vanishing TD error would only be achieved if the state-action value function had been learned to perfection and the environment is noise-free.

[61] This work treats some of the hyperparameters that define the optimum as tunable. Modifying them therefore leads to a deviation from the desired optimum, but may facilitate learning. The then-optimal learned parameter set may thus differ from the desired optimum which is considered bias according to the understanding in this work.

state distribution. The effect of the state distribution is therefore included in this work's definition of bias, allowing to infer a parameter's influence on bias by the difference in learned controller gains between tested configurations. The goal is to limit the amount of bias through hyperparameter choice where possible.

This work's understanding of **stability** is how prone the algorithm is to (catastrophic) failure during learning, i.e. divergence of the learned controller parameters. While in most cases stability is adversely affected by variance, this section points out that some hyperparameters pose exceptions to this, thus confirming the need for this additional perspective. Stability is measured by the relative frequency among experiment rollouts with state vector norms that exceed a large threshold[62]. Stability is of utmost importance for real-world applications. If a hyperparameter threatens to introduce instability it is tuned in a conservative fashion.

In the following, **learning speed** is the inverse time necessary for the learning process to converge. This can be looked at from multiple angles: in real-world applications computation time for a learning step plays an important role, as does the waiting time for filling the experience storage or optimizer performance. Another aspect influencing this notion of speed is the definition of convergence, loosely understood as steady state or plateau[63] in the policy parameter. Additionally, the amount of necessary training steps or computation time may depend on how far the starting point is from convergence, i.e. how far the parameter set the algorithm was initialized with and the parameter set at convergence are apart. Unfortunately, these concepts are not unambiguously and accurately measureable, which is why any verdict is based on experience and the comparison of example training runs. The comparison of learning speed is therefore limited to cases where the differences in how fast a plateau is reached are visible in the plot of the controller gain. For real-world applications speed is very important, since exploration causes wear on the plant and induces experiment cost.

[62] Once a simulation crosses the threshold it is terminated immediately. In the plots given in this section we only include the parameters learned in this run only up to the point the threshold was crossed.

[63] While detecting a plateau is feasible with a criterion, this criterion would need to be adapted due to the different levels of variance across experiments and over training time. An exact comparison is therefore not possible.

The following simulation study aims to showcase the influence of important hyperparameters on the four goals variance, bias, learning speed and stability. For this, it systematically varies the hyperparameters used in the vehicle experiments (with exception of the learning interval, see Table A.1), set E, and applies the resulting algorithm to the example system (1.3). For each configuration, 10 training runs are performed. The training process terminates after a simulated duration 200 s or if the norm of simulated state vector exceeds a threshold of 10^4 to catch severely unstable (i.e. rapidly diverging over a sustained period) controllers. The majority of the plots in the following are averaged over 10 training runs and we mark the interval of 1 standard deviation around the average in each. Where averaged results are presented, training runs that are prematurely terminated are subsequently excluded from the averages.

See Fig. 4.2 for an overview of the parameters varied in the simulation study. Albeit the example system 1.3 mimics the real vehicle's behavior reasonably well throughout most cases, a few exceptions were observed. We therefore comment the results with our experience from the trial runs in the real car in mind.

The complete results of our simulation study can be found in Appendix A.3. Here, we limit ourselves to giving the example of the discount factor and provide a tabular summary.

Discount Factor γ While the present use case values present and future rewards equally (see Section 2.2.2), in RL it can be beneficial to set the discount to a value lower than 1. Discount close to 1 decreases stability[64], increases variance (see TD error[65] and actor gain in Fig. 4.3) but can be seen as decreasing bias (see actor gain in Fig. 4.3). This is in line with other research that considers discount as a form of a regularizer [5] and suggests to progressively increase it during learning [41]. In the vehicle experiments the discount was chosen below 1 to accomodate for other sources of variance, e.g. from the environment. From the graph, little difference in learning speed can be seen: the actor gains stabilize between 110 s and 130 s. The graph suggests a

[64] Stability was not affected in the simulated experiments, but occasionally diverging controller gains have been observed with a discount factor of $\gamma = 0.95$ and higher.

[65] Discount has impact on TD error since it dampens approximation error for future rewards in the Bellman equation.

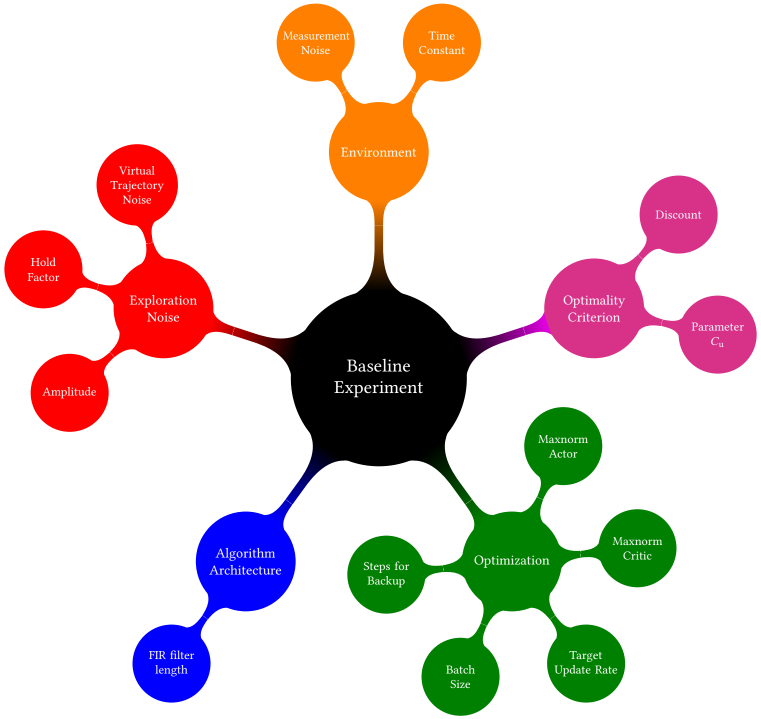


Figure 4.2.: Overview of experiments in simulation study. The complete results can be found in appendix A.3.

minor tendency of slowing learning by increasing the discount factor, but the increase in variance towards higher gains prevents such a conclusion. The chosen discount value of 0.9 leaves a margin to select other hyperparameters affecting variance aggressive enough to allow for fast learning.

Table 4.1 summarizes the intuitions on hyperparameter and noise influence on the learning process. Each row describes how an increase of the respective parameter affects stability, variance, bias and speed. The desired effects are coded in green, and undesired effects are marked in red.

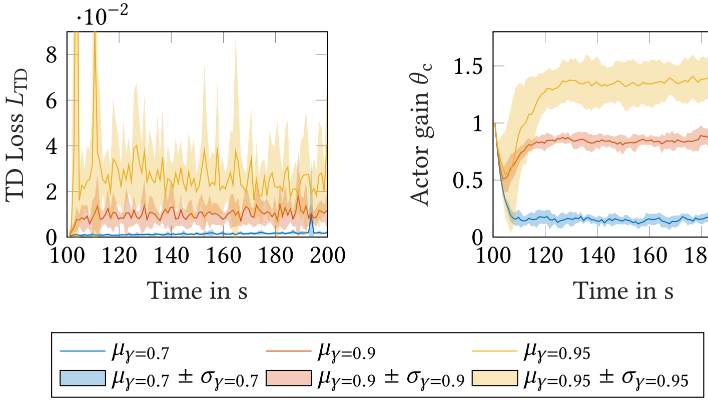


Figure 4.3.: TD error and actor gain for multiple discount factors γ averaged over 10 training runs and their respective standard deviation intervals. While a discount factor $\gamma = 1$ is considered ideal in this work, this would induce infeasible variance levels in the policy gradient, as can be seen by the TD error increasing with the discount factor. However, lower discount factors yield lower actor gains which translate to controllers that are less aggressive. This can be seen as bias with respect to the intended controller. The discount factor γ therefore epitomizes the bias-variance trade-off in RL. Training only begins after the first 100 s since state transitions from this period are used to fill the experience storage.

With the choice of hyperparameters in place[66], we embark on our vehicle tests in the next section.

4.2. Baseline Experiment

The vehicle tests begin with a standard experiment that shows the fitness of the presented approaches to this application and serves as a baseline to benchmark the effect of variations later. This section describes the vehicle setup, track and standard experiment before giving an overview of the results.

[66] In Appendix A.4 several sets of hyperparameters are given. Despite their similarity, we use several sets in the following vehicle experiments since the experiments were carried out over several measurement campaigns. It is our goal to keep the experiments comparable and refer to the corresponding hyperparameter set for each experiment. Since this work aims at comparing two different algorithm architectures, these are tuned to work with comparable hyperparameter sets, but may only reach their full potential if this constraint is lifted.

	Lower value	Intermediate value	Higher value	stability	variance	bias	speed
discount γ	0.7	0.9	0.95	🟥 ↘	🟥 ↗	🟩 ↘	
reward parameter C_u	0.01	0.05	0.2	🟩 ↗	🟩 ↘	🟥 ↗	
actor maxnorm $g_{\text{max,act}}$	0.001	0.1	10	🟥 ↘	🟥 ↗		🟩 ↗
critic maxnorm $g_{\text{max,act}}$	0.01	0.1	1	🟩 ↗	🟥 ↗	🟩 ↘	🟩 ↗
critic target update rate τ	0.025*	0.1	1	🟩 ↗	🟥 ↗	🟥 ↗	🟩 ↗
batch size b	15	150	300	🟩 ↗	🟩 ↘		🟥 (↘)
steps for backup n_{TD}	1	2	5	🟥 ↘	🟥 ↗	🟩 ↘	
experience storage size b_{st}	150	1000	5000	🟩 ↗	🟩 ↘		🟥 (↘)
FIR filter length u_u	25*	35	45	🟩 (↗)	🟩 (↘)	🟩 (↘)	🟥 (↘)
exp. noise amplitude ν_u	0.05	0.5	5	↗↘	↗↘	🟥 ↗	🟩 ↗
exp. noise sample time Δt_{exp}	0.02 s	1 s	4 s	↘↗	↘↗		
virtual trajectory noise ν_p	0.005	0.5	5		🟥 ↗	🟥 ↗	🟩 ↗
env. noise ν_y	0	0.001	0.01	🟥 ↘	🟥 ↗	🟥 ↗	🟥 ↘
env. time constant T_{acc}	0.02	0.1	0.35	🟥 ↘	🟥 ↗		🟥 ↘

Table 4.1.: Overview of hyperparameter influence observed on simulated environment. Each parameter was varied in at least three steps. An asterisk (*) marks that a fourth value has been tested. The arrows mark how an increase over the given range of the respective hyperparameter would affect stability, variance, bias and speed. Desired effects are marked in green, undesired ones in red. If no clear effect is visible, the respective cell is left grey without an arrow. For ambivalent effects, e.g. an increase at first and a decrease later, two arrows are drawn, and for weak effects the arrow is put in brackets.

4.2.1. Setup

The experiments are carried out using a test vehicle based on a *BMW 740Li* [141] (see Fig. 4.4).

The drivetrain consists of a 3.0 L inline six cylinder petrol engine with maximum power output of 240 kW and 450 N m of torque. It is equipped with a *TwinScroll* turbocharger, *Valvetronic* and *VANOS* for variable valve timing and lift. The vehicle is driven through an 8 speed automatic transmission and rear wheel drive. In the presented experiments the vehicle is operated at the upper end of the permissible weight of 2445 kg with co-driver, driver and an electronics rack installed in the luggage compartment (see Fig. 4.4).

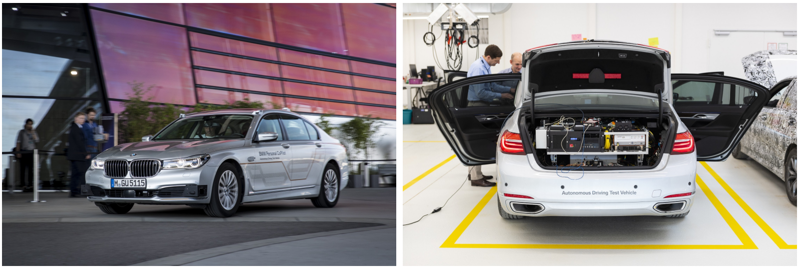


Figure 4.4.: Test vehicle based on a *BMW 740Li* [17, 18] used for online training. It is equipped with a *dSpace Autobox* that runs our proposed algorithm online, closed-loop and in real time. It accesses controllers for chassis/brake and powertrain via in-vehicle bus systems.

It is equipped with a *dSpace Autobox* [29] containing a *dSpace DS1007 PPC Processor Board*[67] [30] that is connected to in-vehicle bus systems. Through these it has access to controllers for the drivetrain which are used to command torque requests from brakes and powertrain. These commands are executed using feedforward controllers. The vehicle speed is estimated based on wheel speed sensors and available through an in-vehicle bus signal in a quantized form with little noise and delay. Delays affecting data transferred over the in-vehicle bus systems are negligible. Programming, monitoring and evaluation is done via a second computer accessing the *dSpace Autobox* via Ethernet.

The *dSpace Autobox* replaces a factory-installed vehicle controller in the vehicle network and provides the interface to the proposed algorithms (see Fig. 1.2):

- It provides the algorithms with inputs, i.e. control targets $\hat{y}$ (including preview, if applicable), measured speeds y and configuration or activation signals.
- It features a low-level controller that accepts the algorithm's output commanded acceleration u and allocates it to feedforward subcontrollers for powertrain and brakes via bus systems, making use of engine drag torque and ensuring steady-state accuracy of acceleration. This includes steady-state compensation of resistances.

[67] At the time of experiment preparation, the hardware used was among the most powerful real-time capable boards rated for in-vehicle application. Future work may harness the performance available through more recent hardware.

Commands to the powertrain are filtered for high frequencies to prevent excessive wear on the variable valve timing and lift assembly.

Both algorithms are split in two steps that are run in separate tasks on the *dSpace Autobox* (see Figs. 3.12 and 2.7):

- The controller evaluation/data preparation step is called every 20 ms. It computes the controller output u from controller target $\hat{y}$ (and preview, if applicable) and measured speed y. Also, it stores the experienced state transitions in the experience storage and provides batches of experience to the learning step.
- The learning step is executed every 600 ms to update model and controller or critic and actor, respectively.

Except where stated otherwise all our learning experiments are performed on a 1.4 km long, mostly level and straight road. The car is manually accelerated to be close to a target speed of $20\,\mathrm{km\,h^{-1}}$ in second gear, then the controller and learning process are activated simultaneously[68]. Near the end of the road the controller is deactivated, the car is turned and again accelerated to a speed close to the target speed manually, then the controller and learning process are activated again. The learning process idles until the experience storage is filled to capacity. The experiments run until a total of 1500 s of combined time for filling the experience storage and learning have elapsed.

The baseline uses exploration noise added to the controller output that is sampled from a uniform distribution between $\nu_\mathrm{u} = 0.5\,\mathrm{m\,s^{-2}}$ and $-\nu_\mathrm{u}$ every $t_\mathrm{noise} = 1\,\mathrm{s}$. While the controller is active, the control target varies in $1\,\mathrm{km\,h^{-1}}$ steps around a base speed[69] of $\bar{\hat{y}} = 20\,\mathrm{km\,h^{-1}}$ every 1.5 s in a regular pattern for additional excitation (see Fig. A.19). For a complete overview of hyperpa-

[68] Since both presented algorithms are off-policy and the idea of virtual targets has already been introduced, it seems a natural extension left as future work to gather data from manual operation of the vehicle and tune the controller in the background. This offline training setting poses the risk of driving the system in areas of the state-space that have not been observed before, but this topic has been adressed in the literature recently, see e.g. [26].

[69] The speed was chosen to reflect average driving speeds in urban areas [7]. It has an additional benefit for safety and practicability. At low speeds a trained driver can swiftly return to a safe state even in case of extreme unexpected controller outputs. High speeds have the additional disadvantage of resulting in longer distance driven, requiring either a more extensive test track (with potentially more variance in the environment) or more frequent turns.

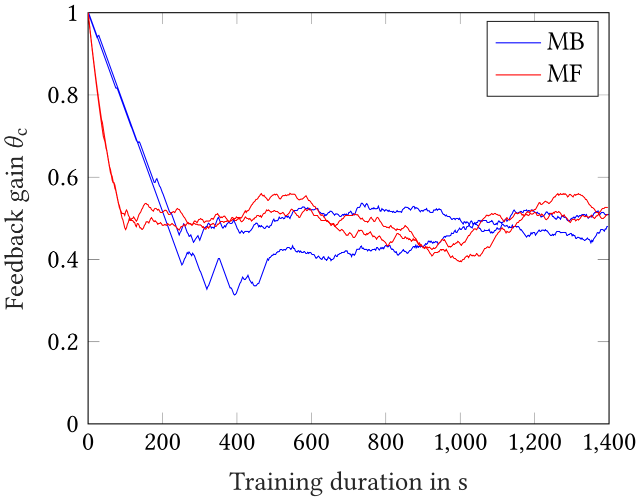


Figure 4.5.: Progression of the output feedback gain during four training runs in the default experiment setup: two with MB and two with MF learning algorithm. The initial slope of the learning curves is limited by the maxnorm constraint. Both configurations converge to similar results after a few minutes while keeping a certain amount of variance after convergence.

rameters in the baseline experiment see Table A.1, set E, for the MF algorithm and Table A.2 for the MB algorithm.

4.2.2. Result

The results of two rollouts with MB and MF algorithms each are shown in Fig. 4.5. All four runs exhibit an almost linear slope at the beginning of the experiment, where the optimizer is limited by the respective maxnorm constraint. The MB algorithm has a stricter constraint, which slows the MB algorithm down more than the MF algorithm. After around 120 s the MF algorithm reaches its steady state with its controller gain oscillating around 0.5. The MB algorithm takes around 300 s to reach a steady state, again around a controller gain of 0.5. In their steady state, both MF and MB algorithms exhibit some variance with gains[70] oscillating between 0.4 and 0.55.

These results show that both algorithms can repeatably learn under real-world conditions. With convergence reached below 2 min and 5 min respectively, both MF and MB algorithm are relatively fast[71]. The learned gains are in the same area, suggesting that either both have the same or no bias.

These results suggest that tuning a speed tracking controller using both proposed RL approaches is feasible, since both converge to similar values. With hyperparameters chosen to be comparable, this is in line with our expectations. The differences in the trajectory of the controller parameter at the beginning of the experiment result from a stricter maxnorm constraint on the MB algorithm, which is necessary to ensure stability of the learning process. Still, a significant amount of variance remains even once learning reaches a steady state for both algorithms.

Both convergence speed and variance in steady state can be influenced by optimizer tuning, but are conflicting goals. A decaying maxnorm constraint can help alleviate this. The minimum maxnorm was intentionally kept at nonzero levels even after convergence to avoid premature stopping and prove that the algorithm converges on its own[72].

The experiment was performed in a controlled environment, yet some factors still add variance: minimal differences in road grade, slight steering corrections and different initial conditions after activation of the controller contribute to variance. However, these influences are expected be minor since they should be averaged out by sampling learning data from a comparably large experience storage.

To gain an intuition on strategies to optimize rewards, experiments using different gains without exploration noise were performed. While these con-

[70] The learning result matches the gain $\theta_c = 0.5$ tuned by an expert engineer according to the subjective rating scale referred to in section 1.2.3.

[71] Convergence speed is influenced by convergence speed of the algorithm and computation time. The former depends on algorithm design (see chapters 2, 3 and appendix A.1). The latter is a mixture of algorithm design decisions like neural network architecture (see sections 2.1.2 and 3.1.1) and batch size (see appendix A.3.2).

[72] See Appendix A.3 to get an intuition on how the maxnorm affects learning in a simulated example.

siderations are not sufficient to scientifically prove the RL algorithm's choice, an interested reader can find them in appendix A.6.

With the feasability shown and the baseline established, the next experiments vary a few important factors in the experiment setup to examine the robustness of the proposed algorithms to different scenarios.

4.3. Effect of Conditions on Learning and Result

Vehicle dynamics is a composite of complex powertrain dynamics and resistances. This section shows that the controller is able to adapt to the different dynamics throughout the operational range of the vehicle by testing it at different speeds and in a scenario with a high disturbance level.

The set of experiments varying speeds relies on mostly the same setup as the baseline experiment: test vehicle, track and algorithm are identical. Instead of using target speeds from the interval of $20\,\mathrm{km\,h^{-1}} \pm 1\,\mathrm{km\,h^{-1}}$, the base speed was changed to $10\,\mathrm{km\,h^{-1}}$ and $30\,\mathrm{km\,h^{-1}}$ respectively. This reflects common speeds in urban areas [7] and puts the learning algorithm to the challenge of near-standstill vehicle dynamics on the one hand and tests its applicability to intermediate speeds on the other.

Fig. 4.6 shows the progression of the learned gain during training runs at different target speeds:

- At speeds around $\bar{\hat{y}} = 30\,\mathrm{km\,h^{-1}}$ with a variation of $\pm 1\,\mathrm{km\,h^{-1}}$ in third gear the learning results are similar to the results for $\bar{\hat{y}} = 20\,\mathrm{km\,h^{-1}}$ with a variation of $\pm 1\,\mathrm{km\,h^{-1}}$ in second gear for both MB and MF algorithms[73].
- $\bar{\hat{y}} = 20\,\mathrm{km\,h^{-1}}$ with a variation of $\pm 1\,\mathrm{km\,h^{-1}}$ in second gear is from the baseline experiment presented above.
- Training at $\bar{\hat{y}} = 10\,\mathrm{km\,h^{-1}}$ with a variation of $\pm 1\,\mathrm{km\,h^{-1}}$ in first gear results in a slightly lower gain around 0.4 for both MB and MF algorithm.

[73] This seems plausible since the powertrain is likely powerful enough to provide a similar acceleration behavior at both $20\,\mathrm{km\,h^{-1}}$ and $30\,\mathrm{km\,h^{-1}}$. This may change at very high speeds, which were not tested in this work for safety reasons.

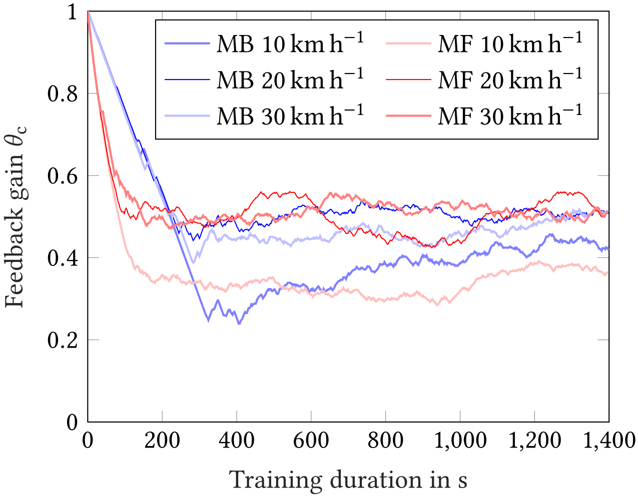


Figure 4.6.: Training runs at target speeds of $\bar{\tilde{y}} = 10\,\mathrm{km\,h^{-1}}$ with a variation of $\pm 1\,\mathrm{km\,h^{-1}}$ in first gear, $\bar{\tilde{y}} = 20\,\mathrm{km\,h^{-1}}$ with a variation of $\pm 1\,\mathrm{km\,h^{-1}}$ in second gear and $\bar{\tilde{y}} = 30\,\mathrm{km\,h^{-1}}$ with a variation of $\pm 1\,\mathrm{km\,h^{-1}}$ in third gear for MF and MB algorithm. At the lowest speed the learned gain is slightly lower at around 0.4 compared to the gain learned at $\bar{\tilde{y}} = 20\,\mathrm{km\,h^{-1}}$ and $\bar{\tilde{y}} = 30\,\mathrm{km\,h^{-1}}$ which is 0.5. The gains for MB and MF algorithm converge to similar gains at each speed, respectively. The evolution of the gains is qualitatively similar, which suggests that the algorithm is capable of learning throughout the tested speed range.

The evolution of the gains is similar to the learning process in our default configuration, suggesting that the algorithms are both capable of learning at different speeds[74]. The lower gains for the low speed experiment may be due to nonlinear adaptation of the low-level controller for parking applications or temporary disengagement of the torque converter[75] between engine and automatic gearbox.

[74] The learned controllers could conveniently be combined in a gain-scheduling fashion to provide a single controller with a wide range of operation. This is left for future work.

[75] A torque converter is a hydraulic component used between a combustion engine and an automatic transmission. It allows the engine to idle while the car is stopped in gear, boosts engine torque at very low speeds and behaves similar to a (dampened) solid connection at higher speeds. In modern powertrains the torque converter is either replaced by automated clutches or equipped with a clutch that can bridge the torque converter to avoid losses at higher speeds [101, Section 6.3.4, pp. 111].

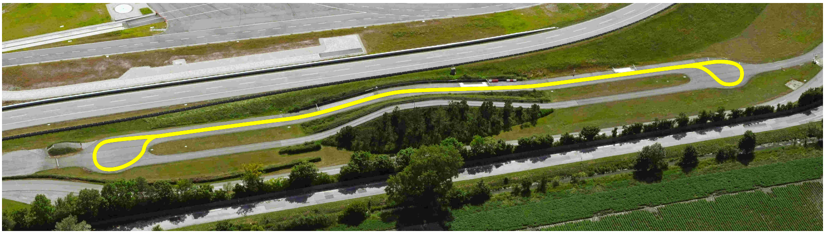


Figure 4.7.: Test track and path used for learning experiment with disturbances (modified from [49]). During learning, the vehicle was repeatedly driven over a hill with slopes of 12% and 16% and turned after passing over it. After each pass the vehicle was turned with maximum steering angle to stay within the paved area. Thight turns and slopes act as disturbances on longitudinal dynamics. Since these disturbances affect the vehicle only briefly, the steady-state compensation in the low-level controller can only partly compensate them. The path repeatedly driven during experiments is marked in yellow.

The next experiment aims to prove the algorithms' robustness towards external disturbances. It uses the same vehicle setup as the baseline experiment, but is performed on a track with slopes up to 16% and tight turns without deactivating the controller. An aerial of the track with the path driven during the experiment is given in Fig. 4.7. Due to the frequent changes in disturbance the steady-state compensation of disturbances in the low-level controller can only partly counteract them, requiring the to-be-learned controller to compensate.

Fig. 4.8 shows that both MB and MF algorithm deviate from their behavior the baseline experiment: The MF algorithm initially has a similar initial slope, but gradually climbs towards higher gains and is close to 0.7 at the end of the experiment[76]. The MB algorithm shows some variance from the start. The plot of the gain shows more pronounced interruptions of the downward trend. After around 500 s the gain is around 0.7 and stays there until the experiment is terminated.

Since the optimal output controller gain depends on the scenario, this behavior is in line with our expectations.

This shows that the algorithms are capable of learning even in scenarios with frequent and strong disturbances, i.e. in less controlled environments.

[76] The experiment had to be terminated earlier due to limited availability of the proving grounds.

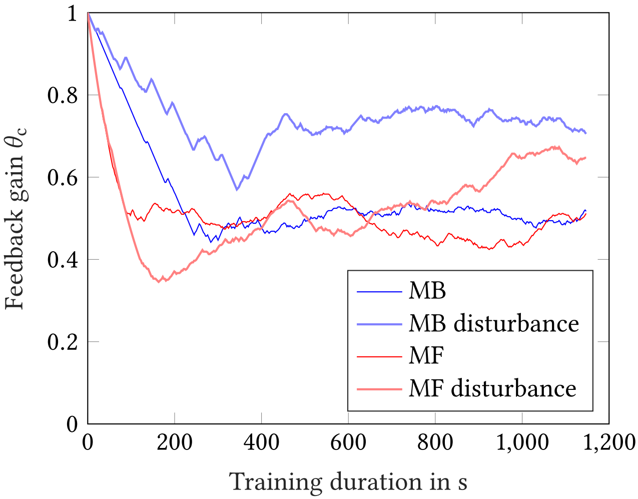


Figure 4.8.: Comparison of evolution of output gains during training under different disturbance conditions. The baseline experiment, performed on a straight, even road is compared to learning while repeatedly driving over a steep hill. In the latter case neither learning nor controller were deactivated during turning, resulting a scenario with heavy disturbance. Both MF and MB controller learn a higher gain[76].

4.4. Effect of Exploration Noise on Learning and Result

The choice of exploration noise not only affects the learning result by choosing which part of the system dynamics are exposed as a consequence of excitation but may also influence the distribution of experienced states (see Section 2.2.5). Two variations of the baseline experiment are therefore conducted, in which amplitude and distribution of exploration noise are varied, respectively. These experiments show that MB and MF algorithms react differently to variations of exploration noise. While both algorithms are susceptible to amplitude variations, especially the MF algorithm is sensitive to the distribution.

The first variation of the baseline experiment changes the amplitude ν_u of the exploration signal added to the controller output. The baseline experiment uses $\nu_u = 0.5\,\mathrm{m\,s^{-2}}$, which is lowered to $\nu_u = 0.25\,\mathrm{m\,s^{-2}}$ and increased to $\nu_u = 0.75\,\mathrm{m\,s^{-2}}$ respectively. Values beyond this range were not deemed

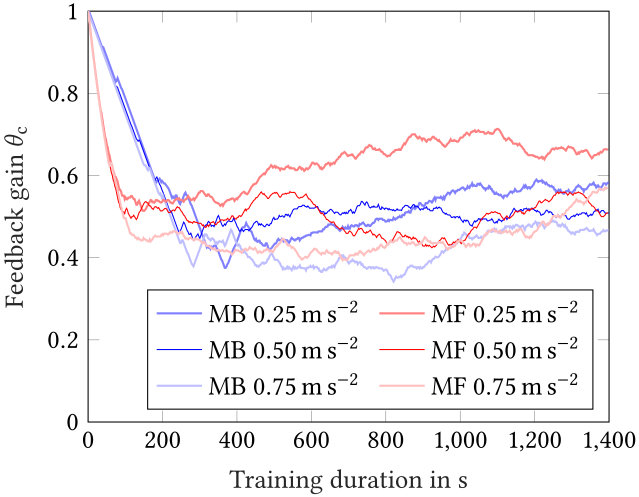


Figure 4.9.: Training runs with different amplitudes of exploration noise. The training runs with exploration noise amplitude 0.25 m s^{-2} tend to slightly higher gains on the MF algorithm than the training runs with amplitudes 0.5 m s^{-2} and 0.75 m s^{-2}. In these cases MB and MF converge to similar values around $\theta_c = 0.5$.

feasible[77]: Large amplitudes can result in commands that cause the vehicle to stop or strongly increase the engine speed if the controller allows too high of a deviation from the target speed. Additionally, exploration noise amplitude controls how much sudden changes in acceleration are magnified, which result in wear on the engine. All other aspects of the experiment setup remain unchanged compared to the baseline experiment.

The results are shown in Fig. 4.9. In the presented scenario the MF algorithm learns gains around 0.7 when run with a maximum amplitude of $v_u = 0.25\,\mathrm{m\,s^{-2}}$. The MB algorithm returns a learned gain around 0.5 in this case. Increasing the exploration noise amplitude to $v_u = 0.75\,\mathrm{m\,s^{-2}}$ yields gains around 0.5 for both algorithms.

In comparison to the learned gain in the baseline experiment, learning with the lowered exploration noise amplitude yields a higher gain, while the gain

[77] More extreme choices are presented in a hyperparameter study based on simulation in appendix A.3.4.

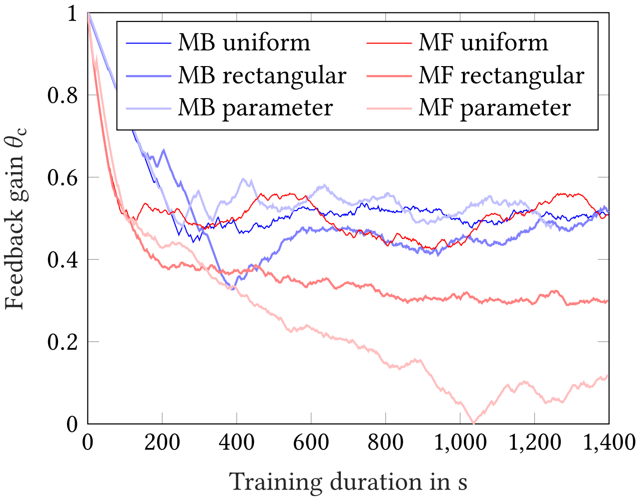


Figure 4.10.: Training runs with different kinds of exploration noise. While the MB algorithm converges around $\theta_c = 0.5$ with all tested variants of exploration noise, the MF algorithm is more sensitive to the choice of exploration noise type. The training run with rectangular noise reaches a lower convergence value around $\theta_c = 0.3$ while the run using parameter noise does not converge within the experiment duration.

learned using an increased exploration noise amplitude is similar to the one learned in the baseline experiment.

The experiment shows that the presented MF algorithm is susceptible to changes in exploration noise. This may be due to the differences in policy search: the MF algorithm optimizes its controller in noise-free simulated rollouts of its learned model. Changes in observed state distribution due to exploration are therefore not reflected in this process.

In a second variation of the baseline experiment the exploration noise distribution is changed. Again, all other factors are kept the way they are in the baseline experiment. In addition to the uniform distribution used in the baseline experiment a rectangle signal and parameter noise were used as exploration signal. The sampling interval of the exploration noise was not varied to avoid damage to the variable valve timing assembly due to high frequency. Appendix A.3.4 shows the effects of this variation in simulated experiments. See Section 2.2.5 for more information on exploration noise.

The presented MB algorithm seems to be more robust against different choices of exploration noise, with all configurations converging to matching gains as shown in Fig. 4.10. The MF algorithm exhibits large variance when paired with parameter noise, with its parameters oscillating over a large range of values (not depicted) or converging to slightly different values when combined with rectangular noise.

Exploration noise seems to be a sensitive influence on the MF algorithm, less so on the MB algorithm. We expected the optimal gain to be influenced by the exploration noise, since the optimal output control gain depends on the initial state for the optimization. The optimization of the MB controller gain results from simulated rollouts starting from observed initial states. In contrast, the MF algorithm solely relies on observed data. The distributions of simulated and experienced states may differ, with the simulated data not corrupted by exploration noise. This may explain the difference in sensitivity towards exploration noise.

4.5. Extensions

With a modification of the proposed algorithms it is possible to include the commanded jerk as an additional measure for comfort in the reward function. Learning then results in a slightly lower controller gain for the MB algorithm, but fails to converge under the MF algorithm. The MF algorithm is extended to learn an optimal feedforward using preview (introduced in Section 3.1.2) and the MB algorithm is enabled to learn an inversion-based feedforward (introduced in Section 3.2.2).

4.5.1. Including Commanded Jerk in Reward

As proposed in Section 1.2.3 a variant of the proposed algorithms includes commanded jerk in the reward function to better reflect common measures for passenger comfort for longitudinal dynamics. With the reward function changed accordingly, both algorithms undergo an experiment with all other factors equal to the baseline experiment.

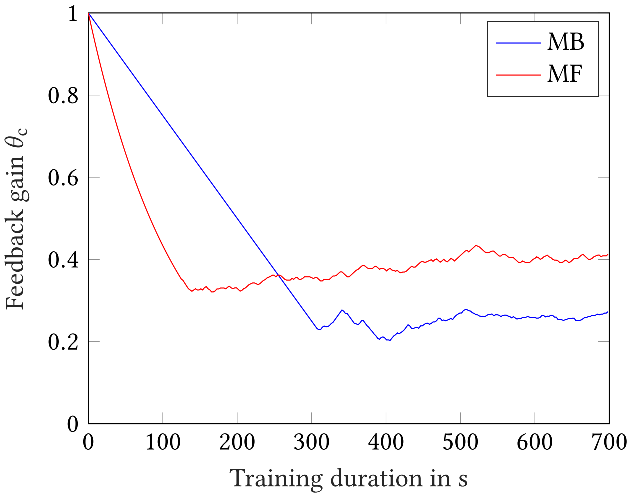


Figure 4.11.: Training including commanded jerk. Both algorithms converge to a lower gain than when trained without jerk with the MF algorithm exhibiting little difference to the baseline experiment.

Fig. 4.11 shows that both algorithms learn a slightly lower gain compared to the baseline experiment. The MF training run yields a gain around 0.4 which is close to the result of the baseline experiment.

The difference in behavior may again be due to the different distributions the controllers are optimized with: the MB algorithm relies on simulated rollouts starting from observed initial states while the MF algorithm purely works on observed state vectors. Since exploration noise increases jerk in the recorded data, but is not present in simulated rollouts, the resulting controller gains may differ.

Generally, it seems plausible that the added penalty for commanded jerk pushes the balance between controller output and tolerated control error towards higher potential control errors which can be achieved by lower controller gains compared to our baseline experiment.

4.5.2. Learning Feedforward

The experiment presented in this section varies the target speed over a range of speeds. The feedforward extensions to MB and MF algorithms are applied, respectively, and it shows that while they still converge they enhance the ability of the learned controller to follow the target profile.

As stated in Section 1.2.2, this work considers feedforward control as an additional component of the controller that enables better tracking of a changing reference value. See Sections 3.1.2 and 3.2.2 for the implementation of MF and MB feedforward algorithms, respectively.

In the experiments presented in this section the control target changes according to a repeating profile consisting of a sinusoid with its value held constant at its extremes[78], see Fig. 4.14. This maneuver[79] covers target speeds ranging between $10\,\mathrm{km\,h^{-1}}$ and $25\,\mathrm{km\,h^{-1}}$ and is therefore performed with automated gear shifts between first and second gear. The gear changes occur around $15\,\mathrm{km\,h^{-1}}$ and since they are neither directly influenced nor observed by the algorithm, the algorithms perceives them as change in the plant dynamics.

The hyperparameters for the MF algorithm for this experiment are given in Table A.1, set D, in the appendix[80]; the MB algorithm is used in its default configuration but with the added inversion-based feedforward. See Table A.2 for hyperparameters. The vehicle setup and track are identical to the baseline experiment.

Fig. 4.12 shows that the MF algorithm learns higher feedback gains than with the default target speed variation (see Section 4.2), yet converges with similar variance independently of a simultaneously learned feedforward. The MB algorithm behaves similarly as in the default case: the initial slope for the controller gain is equally limited by the maxnorm constraint. In the steady state it exhibits a similar amount of variance as in the feedback-only baseline

[78] Note that the factory controller requires differentiable trajectories and includes a position control loop and therefore cannot be compared to the learned result.

[79] The influence of the target trajectory on the presented algorithms is an unexplored avenue for research open for future work. Driving cycles like the WLTP [151] provide scenarios that are more prone to what a consumer vehicle experiences, and the learned controllers would likely be optimal in these driving use cases.

[80] This experiment is part of an earlier campaing published in [81], relying on a different set of hyperparameters.

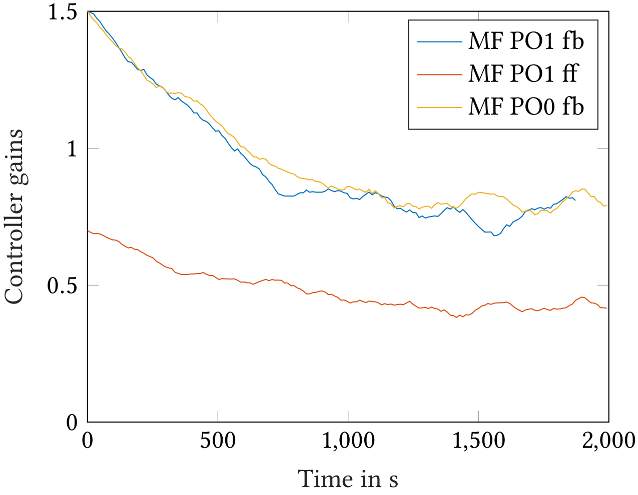


Figure 4.12.: Controller gains during training MF with (marked as "ff") and without feedforward (marked as "fb"). In the experiments the MF algorithm uses a projection on a reference polynomial of order 1 (marked as "PO1") for learning feedforward. In this case, since two coefficients (constant an linear) of the reference polynomial serve as inputs to the policy, two gains are learned, where the on weighting the linear term accounts for the feedforward component. The feedback-only case uses a projection on a constant polynomial (equivalent to no projection at all) and is therefore marked as "PO0". The convergence values for the feedback gains are similar in both cases, albeit higher than in the default scenario with rectangular target variation.

experiment. The learned feedback gain matches the feedback gain of the MF algorithm as Fig. 4.13 shows[81].

The learned controllers are tested to track the sinoid maneuver without exploration noise; track and vehicle are identical to the baseline experiment.

[81] The feedforward gains are not depicted because of the large number of coefficients and different value range.

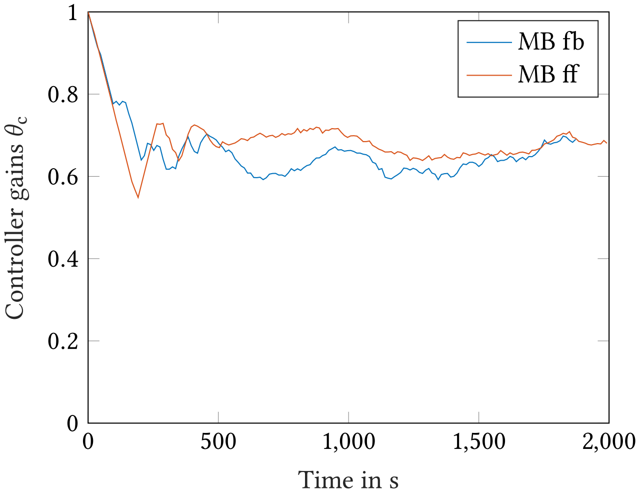


Figure 4.13.: Feedback gains during training of MB algorithm with and without feedforward. The learning progresses in a similar way as in the default scenario, yet the learned gains converge at higher values around 0.7 for both cases.

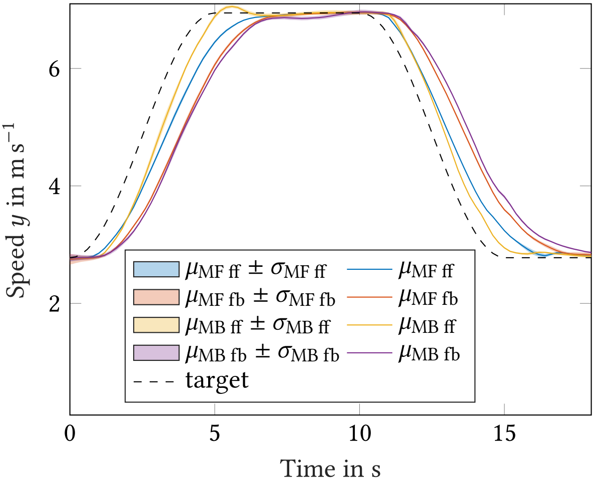


Figure 4.14.: Trained controllers with and without feedforward following the reference maneuver without exploration noise. Both approaches to feedforward learning outperform the feedback-only variants as they significantly reduce lag. The MF feedforward variant is more aggressive than the MB feedforward.

Both approaches yield an improvement in tracking performance as Fig. 4.14 shows[82]. The manuever is performed with less lag by both learned feedforward controllers. The MF controller allows the vehicle speed to lag behind the target more than the MB controller does. However, the latter controller makes the vehicle speed overshoot at the end of the acceleration phase. As the MF feedforward is designed to optimize the reward, it trades accuracy for reduced controller outputs. This is contrast to the MB controller, which is designed to track the target as closely as possible.

Both algorithms provide viable options for learning feedforward control and increase tracking performance in our experiment. This work is among the first ones exploring tracking control using RL in a real-world application. It could be an interesting field for future research to examine the influence of the trajectory during training on the learned feedforward controller.

[82] When training artificial neural networks it is generally avoided to measure the performance of the trained network on the data it was trained on since this would encourage overfitting to the training data, i.e. hamper generalization ability [48, Section 5.2]. This work adheres to this principle, too: While the planned trajectory was the same during learning and evaluation, the trajectory the algorithm learned from significantly differed between training and evaluation due to the addition of virtual trajectory noise (see Section 3.1.2.3) and exploration noise during training.

5. Discussion and Conclusion

This chapter glances over the proposed algorithms from Chapter 3 and the experiment results from Chapter 4. We interpret them and use them to answer the research questions from Chapter 1, point out limitations and finish with a few suggestions for future work.

This work set out to answer three main research questions in Section 1.4, which we re-state here for convenience:

1. How can optimal speed tracking controllers be learned? For this, the capabilities of RL have to be expanded to tracking control and online learning in a real vehicle.
2. How do MB and MF methods compare in this task?
3. How can a feedforward controller be learned in an RL framework?

To adress these, this work proposes two approaches to RL for speed tracking control in a real vehicle. These have to overcome specific challenges that result from the intended application (see Section 2.3): tracking control using RL, a partially observed plant, constrained computational resources, limited learning time and fitness for real-world application.

The proposed MB algorithm is based on a combination of a learning ARX model and policy search. An additional automated feedforward design is based on approximate inversion. The ARX model is estimated from vectors of current and past plant inputs and outputs, which works on the partially observed plant. With a limited number of parameters and a linear model structure the computational effort for parameter estimation is limited. The computational load for the simulation-based controller design can be matched with the available resources depending on batch size and update frequency.

Additionally, this work proposes a MF algorithm based on the actor-critic architecture employing a critic network architecture taylored to a class of partially observed plants that are common in control applications. The MF algorithm is extended with a target compression method that allows to learn an optimal feedforward controller using common actor-critic architectures.

In contrast to prior work that was mostly based on simulation, this work successfully applies both algorithms to a real vehicle for online, closed-loop learning. In the presented experiments both algorithms have proven to perform under various conditions. These include variations of speed and gear, strong disturbances, different kinds of exploration noise and the reward function. Both algorithms consistently converge in an array of experiments, learn a feedforward controller and yield similar results except for the experiments varying exploration and reward signal.

These extensive real-world tests prove that both approaches are valid solutions for learning optimal speed tracking controllers. The results suggest that the proposed MB algorithm tends to be slightly more robust, e.g. to different kinds of exploration noise or discount levels while the MF algorithm converged more quickly. This may, however, be due to our choice of hyperparameters.

When it comes to feedforward control, again, both approaches are viable. However, from an engineering perspective an ideal feedforward, i.e. ideally inverting the plant, may be preferred over an optimal feedforward, i.e. balancing cost of deviation from target against control effort, since the target trajectory is likely already optimized during planning.

By enabling online learning in a real vehicle, this work is a step forward for RL research and control theory. The proposed algorithms have proven to work well within the constraints of computational resources, time and real-world experiment conditions. With these additions to the state of the art, it overcomes several limitations that prevented RL from being applied to tasks outside simulated examples. Algorithms of this capability can take the tedious controller tuning work off an engineer’s shoulders.

Own Publications

[81] Florian Köpf, Luca Puccetti, Christian Rathgeber, and Sören Hohmann. "Reinforcement Learning for Speed Control with Feedforward to Track Velocity Profiles in a Real Vehicle". In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–8. ISBN: 978-1-7281-4149-7. DOI: 10.1109/ITSC45102.2020.9294541. URL: https://ieeexplore.ieee.org/document/9294541 (visited on 02/26/2021).

[82] Florian Köpf, Simon Ramsteiner, Luca Puccetti, Michael Flad, and Sören Hohmann. "Adaptive dynamic programming for model–free tracking of trajectories with time–varying parameters". In: *International Journal of Adaptive Control and Signal Processing* 34.7 (2020), pp. 839–856. ISSN: 0890-6327. DOI: 10.1002/acs.3106.

[118] Luca Puccetti, Florian Köpf, Christian Rathgeber, and Sören Hohmann. "Speed Tracking Control using Online Reinforcement Learning in a Real Car". In: *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*. IEEE, 2020, pp. 392–399. ISBN: 978-1-7281-6139-6. DOI: 10.1109/ICCAR49639.2020.9108051.

[119] Luca Puccetti, Christian Rathgeber, and Sören Hohmann. "Actor-Critic Reinforcement Learning for Linear Longitudinal Output Control of a Road Vehicle". In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 2907–2913. ISBN: 978-1-5386-7024-8. DOI: 10.1109/ITSC.2019.8917113.

[120] Luca Puccetti, Ahmed Yasser, Christian Rathgeber, Andreas Becker, and Sören Hohmann. "Speed Tracking Control Using Model-Based Reinforcement Learning in a Real Vehicle". In: *2021 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 7112021, pp. 1213–1219. ISBN: 978-1-7281-5394-0. DOI: 10.1109/IV48863.2021.9576031.

Bibliography

[1] Alexander Douglas Aberdeen. “Policy-Gradient Algorithms for Partially Observable Markov Decision Processes”. Dissertation. Canberra, Australien: Australian National University, 2003. URL: https://openresearch-repository.anu.edu.au/bitstream/1885/48180/6/02whole.pdf (visited on 06/29/2018).

[2] Bartono Adiprasito. *Fahrzeuglängsführung im Niedergeschwindigkeitsbereich: Zugl.: Braunschweig, Techn. Univ., Diss., 2003.* Vol. 6. Schriftenreihe des Instituts für Fahrzeugtechnik TU Braunschweig. Aachen: Shaker, 2004. ISBN: 3-8322-2367-3.

[3] Dietmar Ahlemann et al. *Digital Auto Report 2020: Navigating through a post-pandemic world.* London, 2020. URL: https://www.strategyand.pwc.com/de/de/studie/2020/digital-auto-report-2020.html (visited on 05/12/2021).

[4] H. Akaike. “A new look at the statistical model identification”. In: *IEEE Transactions on Automatic Control* 19.6 (1974), pp. 716–723. ISSN: 0018-9286. DOI: 10.1109/TAC.1974.1100705.

[5] Ron Amit, Ron Meir, and Kamil Ciosek. “Discount Factor as a Regularizer in Reinforcement Learning”. In: *Proceedings of ICML 2020.* Ed. by PMLR. Proceedings of Machine Learning Research. 2020. URL: https://icml.cc/Conferences/2020/ScheduleMultitrack?event=6021 (visited on 07/19/2021).

[6] Charles W. Anderson, Douglas C. Hittle, Alon D. Katz, and R.Matt Kretchmar. “Synthesis of reinforcement learning, neural networks and PI control applied to a simulated heating coil”. In: *Artificial Intelligence in Engineering* 11.4 (1997), pp. 421–429. ISSN: 09541810. DOI: 10.1016/S0954-1810(97)00004-6.

[7] M. André and U. Hammarström. “Driving speeds in Europe for pollutant emissions estimation”. In: *Transportation Research Part D: Transport and Environment* 5.5 (2000), pp. 321–335. ISSN: 13619209. DOI: 10.1016/S1361-9209(00)00002-X.

[8] Marcin Andrychowicz et al. “Hindsight Experience Replay”. In: *NIPS 2017 Proceedings of Neural Information Processing Systems.* Ed. by MIT Press Cambridge. Cambridge, 2017. URL: `https://proceedings.neurips.cc/paper/2017/hash/453fadbd8a1a3af50a9df4df899537b5-Abstract.html` (visited on 03/04/2021).

[9] Marcin Andrychowicz et al. “Learning dexterous in-hand manipulation”. In: *The International Journal of Robotics Research* 39.1 (2020), pp. 3–20. ISSN: 0278-3649. DOI: `10.1177/0278364919887447`.

[10] Karl J. Aström and Richard M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers.* Second Edition. Pasadena, California: Princeton University Press, 2020. URL: `http://www.cds.caltech.edu/~murray/amwiki/index.php/Main_Page`.

[11] Karl J. Åström and Tore Hägglund. *PID controllers: Theory, design, and tuning.* 2. ed. Research Triangle Park, NC: Instrument Society of America, 1995. ISBN: 1556175167.

[12] Leemon C. Baird III and A. Harry Klopf. *Reinforcement Learning with High-Dimensional, Continuous Actions.* Ed. by WRIGHT LAB WRIGHT-PATTERSON AFB OH. Wright-Patterson Air Force Base, OH 45433-7301, USA. URL: `http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA280844` (visited on 08/17/2017).

[13] Gianluca Baldassarre and Marco Mirolli, eds. *Intrinsically Motivated Learning in Natural and Artificial Systems.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. ISBN: 978-3-642-32374-4. DOI: `10.1007/978-3-642-32375-1`.

[14] Hanna Bellem. *Comfort in Automated Driving: Analysis of Driving Style Preferences in Automated Driving.* 2018. URL: `https://nbn-resolving.org/urn:nbn:de:bsz:ch1-qucosa2-214074` (visited on 03/23/2020).

[15] Felix Berkenkamp and Angela P. Schoellig. “Safe and Robust Learning Control with Gaussian Processes”. In: *2015 European Control Conference (ECC).* IEEE, 2015, pp. 2496–2501. ISBN: 978-3-9524-2693-7. DOI: `10.1109/ECC.2015.7330913`.

[16] Felix Berkenkamp, Matteo Turchetta, Angela P. Schoellig, and Andreas Krause. “Safe Model-Based Reinforcement Learning with Stability Guarantees”. In: *NIPS 2017 Proceedings of Neural Information Processing Systems*. Ed. by MIT Press Cambridge. Cambridge, 2017. URL: https://arxiv.org/pdf/1705.08551.pdf (visited on 10/02/2018).

[17] BMW Group. *Automated Driving at the BMW Group*. 2017. URL: https://www.press.bmwgroup.com/global/photo/compilation/T0271369EN/automated-driving-at-the-bmw-group (visited on 12/09/2020).

[18] BMW Group. *New centre of excellence for autonomous driving. BMW officially opens its autonomous driving campus in Unterschleißheim near Munich*. 2018. URL: https://www.press.bmwgroup.com/global/photo/compilation/T0280021EN/new-centre-of-excellence-for-autonomous-driving-bmw-officially-opens-its-autonomous-driving-campus-in-unterschleissheim-near-munich (visited on 12/09/2020).

[19] Steven J. Bradtke. “Reinforcement Learning Applied to Linear Quadratic Regulation”. In: *Advances in Neural Information Processing Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, pp. 295–302. ISBN: 1-55860-274-7. URL: https://papers.nips.cc/paper/712-reinforcement-learning-applied-to-linear-quadratic-regulation.pdf (visited on 01/25/2018).

[20] Noam Brown and Tuomas Sandholm. “Superhuman AI for multiplayer poker”. In: *Science (New York, N.Y.)* 365.6456 (2019), pp. 885–890. DOI: 10.1126/science.aay2400.

[21] J. A. Butterworth, L. Y. Pao, and D. Y. Abramovitch. “Analysis and comparison of three discrete-time feedforward model-inverse control techniques for nonminimum-phase systems”. In: *Mechatronics* 22.5 (2012), pp. 577–587. ISSN: 09574158. DOI: 10.1016/j.mechatronics.2011.12.006.

[22] Hao-Tien Lewis Chiang, Aleksandra Faust, Marek Fiser, and Anthony Francis. “Learning Navigation Behaviors End-to-End With AutoRL”. In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 2007–2014. DOI: 10.1109/LRA.2019.2899918.

[23] Sungwoo Choi, Brigitte d'Andréa-Novel, Michel Fliess, Hugues Mounier, and Jorge Villagra. "Model-free control of automotive engine and brake for Stop-and-Go scenarios". In: *2009 European Control Conference (ECC)*. Ed. by IEEE. [S.l.]: IEEE, 2009. ISBN: 3952417394.

[24] Mark Cutler and Jonathan P. How. "Autonomous drifting using simulation-aided reinforcement learning". In: *2016 IEEE International Conference on Robotics and Automation, Stockholm, Sweden, May 16th-21st*. Ed. by Allison Okamura and Arianna Menciassi. Piscataway, NJ: IEEE, 2016, pp. 5442–5448. ISBN: 978-1-4673-8026-3. DOI: 10.1109/ICRA.2016.7487756.

[25] Sarah Dean, Horia Mania, Nikolai Matni, Benjamin Recht, and Stephen Tu. "On the Sample Complexity of the Linear Quadratic Regulator". In: *Foundations of Computational Mathematics* 20.4 (2020), pp. 633–679. ISSN: 1615-3375. DOI: 10.1007/s10208-019-09426-y.

[26] Sarah Dean, Stephen Tu, Nikolai Matni, and Benjamin Recht. "Safely Learning to Control the Constrained Linear Quadratic Regulator". In: *2019 American Control Conference (ACC)*. IEEE, 10.07.2019 - 12.07.2019, pp. 5582–5588. ISBN: 978-1-5386-7926-5. DOI: 10.23919/ACC.2019.8814865.

[27] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. "Gaussian Processes for Data-Efficient Learning in Robotics and Control". In: *IEEE transactions on pattern analysis and machine intelligence* 37.2 (2015), pp. 408–423. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2013.218.

[28] S. Dermann and R. Isermann. "Nonlinear distance and cruise control for passenger cars". In: *1995 American Control Conference - ACC'95*. 1995, pp. 3081–3085. DOI: 10.1109/ACC.1995.532083.

[29] dSPACE GmbH. *AutoBox*. URL: https://www.dspace.com/de/gmb/home/products/hw/accessories/autobox.cfm (visited on 11/06/2020).

[30] dSPACE GmbH. *DS1007 PPC Processor Board*. 2014. URL: https://www.dspace.com/de/gmb/home/products/hw/phs_hardware/processor_boards/ds1007.cfm (visited on 11/06/2020).

[31] John C. Duchi, Elad Hazan, and Yoram Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2121–2159. URL: https://dl.acm.org/doi/10.5555/1953048.2021068 (visited on 01/27/2021).

[32] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. *Challenges of Real-World Reinforcement Learning*. URL: http://arxiv.org/pdf/1904.12901v1.

[33] Simon Dylla. *Entwicklung einer Methode zur Objektivierung der subjektiv erlebten Schaltbetätigungsqualität von Fahrzeugen mit manuellem Schaltgetriebe. Development of a method for the objectification of subjectively perceived gear-control quality of manual-shifted vehicles.* 2010. DOI: 10.5445/IR/1000019821.

[34] Thomas Eigel. *Integrierte Längs- und Querführung von Personenkraftwagen mittels Sliding-Mode-Regelung*. Braunschweig: Univ.-Bibl, 2010. ISBN: 978-3-8325-2457-9.

[35] J. Elman. “Finding structure in time”. In: *Cognitive Science* 14.2 (1990), pp. 179–211. ISSN: 03640213. DOI: 10.1016/0364-0213(90)90002-E.

[36] Azim Eskandarian. *Handbook of Intelligent Vehicles*. London: Springer London, 2012. ISBN: 978-0-85729-084-7. DOI: 10.1007/978-0-85729-085-4.

[37] Euro NCAP. *European New Car Assessment Programme: Assessment Protocol*. 2020. URL: https://www.euroncap.com/en/for-engineers/protocols/ (visited on 05/13/2021).

[38] Michael Festner. “Objektivierte Bewertung des Fahrstils auf Basis der Komfortwahrnehmung bei hochautomatisiertem Fahren in Abhängigkeit fahrfremder Tätigkeiten: Grundlegende Zusammenhänge zur komfortorientierten Auslegung eines hochautomatisierten Fahrstils”. PhD thesis. DuEPublico: Duisburg-Essen Publications online, University of Duisburg-Essen, Germany, 2019. DOI: 10.17185/duepublico/70681.

[39] Christodoulos A. Floudas and Panos M. Pardalos. *Encyclopedia of Optimization*. Second Edition. Boston, MA: Springer-Verlag, 2009. ISBN: 9780387747583. DOI: 10.1007/978-0-387-74759-0. URL: http://dx.doi.org/10.1007/978-0-387-74759-0.

[40] Andreas Folkers, Matthias Rick, and Christof Buskens. “Controlling an Autonomous Vehicle with Deep Reinforcement Learning”. In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2019, pp. 2025–2031. ISBN: 978-1-7281-0560-4. DOI: 10.1109/IVS.2019.8814124.

[41] Vincent François-Lavet, Raphael Fonteneau, and Damien Ernst. *How to Discount Deep Reinforcement Learning: Towards New Dynamic Strategies.* URL: http://arxiv.org/pdf/1512.02011v2.

[42] G. P. Frost, M.N Howell, T. J. Gordon, and Q.H Wu. “Dynamic Vehicle Roll Control using Reinforcement Learning”. In: *UKACC International Conference on Control ’96*. Tuscaloosa, Ala.: The @Univ. of Alabama Press, 1996, pp. 1107–1112. ISBN: 0. DOI: 10.1049/cp:19960708.

[43] Scott Fujimoto, Herke Hoof, and David Meger. “Addressing Function Approximation Error in Actor-Critic Methods”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by PMLR. 2016, pp. 1587–1596. URL: http://proceedings.mlr.press/v80/fujimoto18a.html (visited on 02/10/2021).

[44] Marco Galvani. “History and future of driver assistance”. In: *IEEE Instrumentation & Measurement Magazine* 22.1 (2019), pp. 11–16. ISSN: 1094-6969. DOI: 10.1109/MIM.2019.8633345.

[45] Gartner. *2020 Hype Cycle for Connected and Smart Mobility.* 2020. URL: https://www.sae.org/news/2020/09/2020-hype-cycle-for-connected-vehicles-and-smart-mobility.

[46] Jason Gauci et al. *Horizon: Facebook’s Open Source Applied Reinforcement Learning Platform.* URL: http://arxiv.org/pdf/1811.00260v5.

[47] Stefan Germann. *Modellbildung und modellgestützte Regelung der Fahrzeuglängsdynamik.* Düsseldorf: VDI-Verlag, 1997.

[48] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Cambridge, Massachusetts and London, England: MIT Press, 2016. ISBN: 978-0262035613. URL: http://www.deeplearningbook.org/.

[49] Robert Grahn. *Teststrecke und Übungsplatz auf dem BMW Messgelände Aschheim im Ortsteil Neufinsing in Aschheim im Bundesland Bayern, Deutschland.* Ed. by euroluftbild.de, Dirschauer Straße 2, 12623 Berlin.

[50] Ivo Grondman and Robert Babuška. *Online model learning algorithms for actor-critic control.* Delft, 2015. ISBN: 978-94-6186-432-1.

[51] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates". In: *2017 IEEE International Conference on Robotics and Automation (ICRA).* IEEE, 2017, pp. 3389–3396. ISBN: 978-1-5090-4633-1. DOI: 10.1109/ICRA.2017.7989385.

[52] Philippe Guillot and Gilles Millerioux. "Flatness and Submersivity of Discrete-Time Dynamical Systems". In: *IEEE Control Systems Letters* 4.2 (2020), pp. 337–342. DOI: 10.1109/LCSYS.2019.2926374.

[53] Benjamin Gutjahr. *Recheneffiziente Trajektorienoptimierung für automatisierte Fahreingriffe.* Karlsruhe: KIT Scientific Publishing, 2019. ISBN: 9783731509226.

[54] Sehoon Ha, Peng Xu, Zhenyu Tan, Sergey Levine, and Jie Tan. *Learning to Walk in the Real World with Minimal Human Effort.* URL: http://arxiv.org/pdf/2002.08550v3.

[55] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *ICML 2018.* 2018, pp. 1861–1870. URL: http://proceedings.mlr.press/v80/haarnoja18b.html (visited on 02/10/2021).

[56] Tuomas Haarnoja et al. *Soft Actor-Critic Algorithms and Applications.* URL: http://arxiv.org/pdf/1812.05905v2.

[57] Danijar Hafner, Thimothy P. Lillicrap, Jimmy Ba, and Mohammad Norouzi. "Dream to Control: Learning Behaviors by Latent Imagination". In: *8th Conference on Learning Representations.* Ed. by Openreview.net. 2020. URL: https://openreview.net/pdf?id=S1lOTC4tDS (visited on 11/09/2020).

[58] Michael Hafner and John Bales. "Low-Speed Longitudinal Distance Control with Applications to Parking". In: SAE Technical Paper Series. SAE International400 Commonwealth Drive, Warrendale, PA, United States, 2017. DOI: 10.4271/2017-01-0036.

[59] Roland Hafner and Martin Riedmiller. "Reinforcement learning in feedback control". In: *Machine Learning* 84.1-2 (2011), pp. 137–169. ISSN: 0885-6125. DOI: 10.1007/s10994-011-5235-x.

[60] Jean Harb and Doina Precup. *Investigating Recurrence and Eligibility Traces in Deep Q-Networks.* URL: http://arxiv.org/pdf/1704.05495v1 (visited on 04/11/2019).

[61] Matthew Hausknecht and Peter Stone. *Deep Recurrent Q-Learning for Partially Observable MDPs*. URL: http://arxiv.org/pdf/1507.06527v4.

[62] P. I. Hedrick, Datta Godbole, Rajesh Rajamani, and Pete Seiler. *Stop and Go Cruise Control*. URL: https://www.aem.umn.edu/~SeilerControl/Papers/1999/HedrickEtAl_99TR_StopAndGoCruiseControl.pdf (visited on 01/25/2018).

[63] Nicolas Heess, Jonathan Hunt, Thimothy P. Lillicrap, and David Silver. *Memory-based control with recurrent neural networks*. URL: https://arxiv.org/abs/1512.04455 (visited on 06/11/2018).

[64] Bernd Heißing and Hans Jürgen Brandl. *Subjektive Beurteilung des Fahrverhaltens*. 1. Aufl. Vogel-Fachbuch. Würzburg: Vogel, 2002. ISBN: 9783802319037.

[65] Bernd Heißing, Metin Ersoy, and Stefan Gies. "Fahrkomfort". In: *Fahrwerkhandbuch*. Ed. by Bernd Heißing, Metin Ersoy, and Stefan Gies. Wiesbaden: Vieweg+Teubner, 2011, pp. 465–493. ISBN: 978-3-8348-0821-9. DOI: 10.1007/978-3-8348-8168-7_5.

[66] S. Hassan HosseinNia, Ines Tejado, Blas M. Vinagre, Vicente Milanes, and Jorge Villagra. "Low speed control of an autonomous vehicle using a hybrid fractional order controller". In: *2011 2nd International Conference on Control, Instrumentation, and Automation (ICCIA)*. 2011, pp. 116–121. DOI: 10.1109/ICCIAutom.2011.6356641.

[67] M.N Howell and M.C Best. "On-line PID tuning for engine idle-speed control using continuous action reinforcement learning automata". In: *Control Engineering Practice* 8.2 (2000), pp. 147–154. ISSN: 09670661. DOI: 10.1016/S0967-0661(99)00141-0.

[68] Graham Hudson, Alain Leger, Birger Niss, and Istvan Sebestyen. "JPEG at 25: Still Going Strong". In: *IEEE MultiMedia* 24.2 (2017), pp. 96–103. ISSN: 1070-986X. DOI: 10.1109/MMUL.2017.38.

[69] Rolf Isermann and Marco Münchhof. *Identification of Dynamic Systems: An Introduction with Applications*. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2011. ISBN: 978-3-540-78879-9.

[70] Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. "Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by PMLR. 2017.

[71] Max Jaderberg et al. “Human-level performance in 3D multiplayer games with population-based reinforcement learning”. In: *Science (New York, N.Y.)* 364.6443 (2019), pp. 859–865. DOI: 10.1126/science.aau6249.

[72] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. “When to Trust Your Model: Model-Based Policy Optimization”. In: *Advances in Neural Information Processing Systems*. 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/5faf461eff3099671ad63c6f3f094f7f-Abstract.html (visited on 11/11/2020).

[73] Sham Kakade. “A Natural Policy Gradient”. In: *Advances in Neural Information Processing Systems 14 (NIPS 2001)*. 2001. URL: https://proceedings.neurips.cc/paper/2001/file/4b86abe48d358ecf194c56c69108433e-Paper.pdf (visited on 02/10/2021).

[74] Dmitry Kalashnikov et al. *QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation*. URL: http://arxiv.org/pdf/1806.10293v3.

[75] Wolf Dieter Käppler. *Smart Vehicle Handling - Test und Evaluation in der Fahrzeugtechnik*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. ISBN: 978-3-662-46416-8. DOI: 10.1007/978-3-662-46417-5.

[76] Diederik P. Kingma and Jimmy Ba. “ADAM: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations (ICLR) 2015*. Ed. by Openreview.net. 2015. URL: https://arxiv.org/abs/1412.6980 (visited on 01/27/2021).

[77] Bahare Kiumarsi, Frank L. Lewis, Hamidreza Modares, Ali Karimpour, and Mohammad-Bagher Naghibi-Sistani. “Reinforcement -learning for optimal tracking control of linear discrete-time systems with unknown dynamics”. In: *Automatica* 50.4 (2014), pp. 1167–1175. ISSN: 00051098. DOI: 10.1016/j.automatica.2014.02.015.

[78] Bahare Kiumarsi-Khomartash, Frank L. Lewis, Mohammad-Bagher Naghibi-Sistani, and Ali Karimpour. “Optimal tracking control for linear discrete-time systems using reinforcement learning”. In: *Proceedings of the 52nd IEEE Conference on Decision and Control, CDC 2013, December 10-13, 2013, Firenze, Italy*. IEEE, 2013, pp. 3845–3850. ISBN: 978-1-4673-5714-2. DOI: 10.1109/CDC.2013.6760476.

[79] Jens Kober and Jan Peters. “Reinforcement Learning in Robotics: A Survey”. In: *Learning Motor Skills*. Ed. by Jens Kober and Jan Peters. Vol. 97. Springer Tracts in Advanced Robotics. Cham: Springer International Publishing, 2014, pp. 9–67. ISBN: 978-3-319-03193-4. DOI: 10.1007/978-3-319-03194-1_2.

[80] Florian Köpf, Sebastian Ebbert, Michael Flad, and Soren Hohmann. “Adaptive Dynamic Programming for Cooperative Control with Incomplete Information”. In: *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2018, pp. 2632–2638. ISBN: 978-1-5386-6650-0. DOI: 10.1109/SMC.2018.00450.

[81] Florian Köpf, Luca Puccetti, Christian Rathgeber, and Sören Hohmann. “Reinforcement Learning for Speed Control with Feedforward to Track Velocity Profiles in a Real Vehicle”. In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–8. ISBN: 978-1-7281-4149-7. DOI: 10.1109/ITSC45102.2020.9294541. URL: https://ieeexplore.ieee.org/document/9294541 (visited on 02/26/2021).

[82] Florian Köpf, Simon Ramsteiner, Luca Puccetti, Michael Flad, and Sören Hohmann. “Adaptive dynamic programming for model–free tracking of trajectories with time–varying parameters”. In: *International Journal of Adaptive Control and Signal Processing* 34.7 (2020), pp. 839–856. ISSN: 0890-6327. DOI: 10.1002/acs.3106.

[83] Florian Köpf, Johannes Westermann, Michael Flad, and Sören Hohmann. “Adaptive optimal control for reference tracking independent of exo-system dynamics”. In: *Neurocomputing* 405.10 (2020), pp. 173–185. ISSN: 09252312. DOI: 10.1016/j.neucom.2020.04.140.

[84] Gerhard Kreisselmeier. “Struktur mit zwei Freiheitsgraden / Two-Degree-of-Freedom Control Structure”. In: *at - Automatisierungstechnik* 47.6 (1999). ISSN: 0178-2312. DOI: 10.1524/auto.1999.47.6.266.

[85] V. Kučera and C. E. de Souza. “A Necessary and Sufficient Condition for Output Feedback Stabilizability”. In: *Automatica* 31.9 (1995), pp. 1357–1359. ISSN: 00051098. DOI: 10.1016/0005-1098(95)00048-2.

[86] P. R. Kumar and P. P. Varaiya. *Stochastic systems: Estimation, identification, and adaptive control*. Prentice-Hall information and system sciences series. Englewood Cliffs, N.J.: Prentice Hall, 1986. ISBN: 978-0-13-846684-8.

[87] Sampo Kuutti, Richard Bowden, Yaochu Jin, Phil Barber, and Saber Fallah. "A Survey of Deep Learning Applications to Autonomous Vehicle Control". In: *IEEE Transactions on Intelligent Transportation Systems* (2020), pp. 1–22. ISSN: 1524-9050. DOI: 10.1109/TITS.2019.2962338.

[88] Ioan Doré Landau, Rogelio Lozano, Mohammed M'Saad, and Alireza Karimi. *Adaptive Control.* London: Springer London, 2011. ISBN: 978-0-85729-663-4. DOI: 10.1007/978-0-85729-664-1.

[89] Nevena Lazic et al. "Data center cooling using model-predictive control". In: *Advances in Neural Information Processing Systems.* Vol. 31. Red Hook, NY: Curran, 2018, pp. 3814–3823. URL: https://papers.nips.cc/paper/7638-data-center-cooling-using-model-predictive-control (visited on 08/26/2020).

[90] W. Levine and M. Athans. "On the Determination of the Optimal Constant Output Feedback Gains for Linear Multivariable Systems". In: *IEEE Transactions on Automatic Control* 15.1 (1970), pp. 44–48. ISSN: 0018-9286. DOI: 10.1109/TAC.1970.1099363.

[91] Thimothy P. Lillicrap et al. "Continuous Control with Deep Reinforcement Learning". In: *International Conference on Learning Representations.* 2016.

[92] Long-Ji Lin. "Self-improving reactive agents based on reinforcement learning, planning and teaching". In: *Machine Learning* 8.3-4 (1992), pp. 293–321. ISSN: 0885-6125. DOI: 10.1007/BF00992699.

[93] Lennart Ljung. *System Identification: Theory for the User.* 2. ed., 14. printing. Prentice Hall information and system sciences series. Upper Saddle River, NJ: Prentice Hall PTR, 2012. ISBN: 978-0136566953.

[94] Biao Luo, Derong Liu, Tingwen Huang, and Ding Wang. "Model-Free Optimal Tracking Control via Critic-Only Q-Learning". In: *IEEE Transactions on Neural Networks and Learning Systems* 27.10 (2016), pp. 2134–2144. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2016.2585520. URL: https://ieeexplore.ieee.org/document/7509626.

[95] Richard G. Lyons. *Understanding Digital Signal Processing.* 3rd. Safari Tech Books Online. Upper Saddle River, NJ, Toronto, and London: Prentice Hall, 2011. ISBN: 978-0137027415. URL: http://proquest.tech.safaribooksonline.de/9780137028450.

[96] John-Jairo Martinez and Carlos Canudas-de-Wit. “A Safe Longitudinal Control for Adaptive Cruise Control and Stop-and-Go Scenarios”. In: *IEEE Transactions on Control Systems Technology* 15.2 (2007), pp. 246–258. ISSN: 1063-6536. DOI: 10.1109/TCST.2006.886432.

[97] Robert Mayr. *Regelungsstrategien für die automatische Fahrzeugführung: Längs- und Querregelung, Spurwechsel- und Überholmanöver*. Berlin u.a.: Springer, 2001. ISBN: 3-540-67518-3.

[98] G. Meier, G. Roppenecker, and Ch. Wurmthaler. “Automatische Fahrzeug-Längsführung mittels Trajektorien-Folgeregelung: Ein modulares Konzept zur Trennung zwischen Fahrzeug- und Fahrer-Spezifika”. In: *Steuerung und Regelung von Fahrzeugen und Motoren - AUTOREG 2004*. VDI-Berichte. Düsseldorf: VDI-Verl., 2004. ISBN: 3180918284.

[99] Andreas Michalka. *Ein modellbasiertes Gesamtkonzept zur Längsdynamiksteuerung und -regelung von Parallelhybridfahrzeugen*. neue Ausg. Vol. 1. FAU Studien aus der Elektrotechnik. Erlangen: FAU University Press, 2015. ISBN: 978-3-944057-25-5.

[100] Vicente Milanes, Jorge Villagra, Joshué Perez, and Carlos Gonzalez. “Low-Speed Longitudinal Controllers for Mass-Produced Cars: A Comparative Study”. In: *IEEE Transactions on Industrial Electronics* 59.1 (2012), pp. 620–628. ISSN: 0278-0046. DOI: 10.1109/TIE.2011.2148673.

[101] Manfred Mitschke and Henning Wallentowitz. *Dynamik der Kraftfahrzeuge*. 5., überarb. u. erg. Aufl. VDI-Buch. Wiesbaden: Springer Vieweg, 2014. ISBN: 978-3-658-05068-9. DOI: 10.1007/978-3-658-05068-9. URL: http://dx.doi.org/10.1007/978-3-658-05068-9.

[102] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533. ISSN: 1476-4687. DOI: 10.1038/nature14236.

[103] Hamidreza Modares and Frank L. Lewis. “Linear Quadratic Tracking Control of Partially-Unknown Continuous-Time Systems Using Reinforcement Learning”. In: *IEEE Transactions on Automatic Control* 59.11 (2014), pp. 3051–3056. ISSN: 0018-9286. DOI: 10.1109/TAC.2014.2317301.

[104] Thomas M. Moerland, Joost Broekens, and Catholijn M. Jonker. *Model-based Reinforcement Learning: A Survey*. URL: http://arxiv.org/pdf/2006.16712v2.

[105] Jorge J. Moré. “The Levenberg-Marquardt Algorithm: Implementation and Theory”. In: *Numerical Analysis*. Ed. by G. A. Watson. Berlin, Heidelberg: Springer, 1978, pp. 105–116. ISBN: 978-3-540-35972-2. DOI: 10.1007/BFb0067700.

[106] Kevin Patrick Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive computation and machine learning series. Cambridge, Mass.: MIT Press, 2012. ISBN: 978-0-262-01802-9.

[107] Andrew Y. Ng, H. Jin Kim, Michael I. Jordan, and Shankar Sastry. “Autonomous Helicopter Flight via Reinforcement Learning”. In: *Advances in Neural Information Processing Systems 16 (NIPS 2003)*. 2003, pp. 779–806. URL: https://proceedings.neurips.cc/paper/2003/hash/b427426b8acd2c2e53827970f2c2f526-Abstract.html (visited on 02/25/2021).

[108] OpenAI et al. *Dota 2 with Large Scale Deep Reinforcement Learning*. URL: http://arxiv.org/pdf/1912.06680v1.

[109] Brian Paden, Michal Cap, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. “A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles”. In: *IEEE Transactions on Intelligent Vehicles* 1.1 (2016), pp. 33–55. ISSN: 2379-8858. DOI: 10.1109/TIV.2016.2578706.

[110] Thomas W. Parks and Charles S. Burrus. *Digital filter design*. Topics in digital signal processing. Chichester: Wiley, 1987. ISBN: 978-0471828969.

[111] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training Recurrent Neural Networks”. In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Proceedings of Machine Learning Research. PMLR, 2013. URL: http://arxiv.org/pdf/1211.5063v2.

[112] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. “Curiosity-driven Exploration by Self-supervised Prediction”. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by PMLR. Vol. 70. 2017, pp. 2778–2787. URL: http://proceedings.mlr.press/v70/pathak17a.html (visited on 02/15/2021).

[113] Leonid Peshkin, Nicolas Meuleau, and Leslie Kaelbling. “Learning Policies with External Memory”. In: *Machine Learning*. Ed. by Ivan Bratko. San Francisco, Calif.: Kaufmann, 1999. ISBN: 1558606122. URL: http://arxiv.org/pdf/cs/0103003v1 (visited on 03/21/2019).

[114] Matthias Plappert et al. "Parameter Space Noise for Exploration". In: *6th Conference on Learning Representations (ICLR 2018)*. 2018. URL: `https://openreview.net/forum?id=ByBAl2eAZ` (visited on 08/26/2020).

[115] Athanasios S. Polydoros and Lazaros Nalpantidis. "Survey of Model-Based Reinforcement Learning: Applications on Robotics". In: *Journal of Intelligent & Robotic Systems* 86.2 (2017), pp. 153–173. ISSN: 0921-0296. DOI: `10.1007/s10846-017-0468-y`.

[116] Harald Proff et al. *2021 Global Automotive Consumer Study: Global focus countries*. Ed. by Deloitte. 2021. URL: `https://www2.deloitte.com/de/de/pages/consumer-industrial-products/articles/global-automotive-consumer-study.html` (visited on 05/12/2021).

[117] Harald Proff et al. *2022 Global Automotive Consumer Study: Key Findings: Global Focus Countries*. Ed. by Deloitte. 2022. URL: `https://www2.deloitte.com/de/de/pages/consumer-industrial-products/articles/global-automotive-consumer-study.html` (visited on 06/24/2022).

[118] Luca Puccetti, Florian Köpf, Christian Rathgeber, and Sören Hohmann. "Speed Tracking Control using Online Reinforcement Learning in a Real Car". In: *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*. IEEE, 2020, pp. 392–399. ISBN: 978-1-7281-6139-6. DOI: `10.1109/ICCAR49639.2020.9108051`.

[119] Luca Puccetti, Christian Rathgeber, and Sören Hohmann. "Actor-Critic Reinforcement Learning for Linear Longitudinal Output Control of a Road Vehicle". In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 2907–2913. ISBN: 978-1-5386-7024-8. DOI: `10.1109/ITSC.2019.8917113`.

[120] Luca Puccetti, Ahmed Yasser, Christian Rathgeber, Andreas Becker, and Sören Hohmann. "Speed Tracking Control Using Model-Based Reinforcement Learning in a Real Vehicle". In: *2021 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 7112021, pp. 1213–1219. ISBN: 978-1-7281-5394-0. DOI: `10.1109/IV48863.2021.9576031`.

[121] Ning Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural Networks* 12.1 (1999), pp. 145–151. ISSN: 08936080. DOI: `10.1016/s0893-6080(98)00116-6`.

[122] Tobias Radke. *Energieoptimale Längsführung von Kraftfahrzeugen durch Einsatz vorausschauender Fahrstrategien: Zugl.: Karlsruhe, Karlsruher Institut für Technologie (KIT), Diss., 2013.* Vol. 19. Karlsruher Schriftenreihe Fahrzeugsystemtechnik / Institut für Fahrzeugsystemtechnik. Karlsruhe: KIT Scientific Publishing, 2013. ISBN: 9783731500698.

[123] Christian Rathgeber. *Trajektorienplanung und -folgeregelung für assistiertes bis hochautomatisiertes Fahren.* Technische Universität Berlin, 2016. URL: `http://dx.doi.org/10.14279/depositonce-5506` (visited on 01/13/2020).

[124] Martin Riedmiller, Mike Montemerlo, and Hendrik Dahlkamp. "Learning to Drive a Real Car in 20 Minutes". In: *Proceedings of the Frontiers in the convergence of bioscience and information technologies.* Ed. by Daniel Howard. [Piscataway, NJ]: IEEE, 2007, pp. 645–650. ISBN: 978-0-7695-2999-8. DOI: `10.1109/FBIT.2007.37`.

[125] H. H. Rosenbrock. "An Automatic Method for Finding the Greatest or Least Value of a Function". In: *The Computer Journal* 3.3 (1960), pp. 175–184. ISSN: 0010-4620. DOI: `10.1093/comjnl/3.3.175`.

[126] Thomas Rückstieß, Frank Sehnke, Tom Schaul, Daan Wierstra, Yi Sun, and Jürgen Schmidhuber. "Exploring Parameter Space in Reinforcement Learning". In: *Paladyn, Journal of Behavioral Robotics* 1.1 (2010), p. 1213. DOI: `10.2478/s13230-010-0002-4`.

[127] Tim Salimans and Diederik P. Kingma. "Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks". In: *Advances in Neural Information Processing Systems 29 (NIPS 2016).* Ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett. Advances in Neural Information Processing Systems. 2016. URL: `http://arxiv.org/pdf/1602.07868v3`.

[128] Claude Sammut and Geoffrey I. Webb, eds. *Encyclopedia of Machine Learning.* Boston, MA: Springer US, 2010. ISBN: 978-0-387-30768-8. DOI: `10.1007/978-0-387-30164-8`.

[129] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. "Universal Value Function Approximators". In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning.* Microtome Publishing, 2015. URL: `http://proceedings.mlr.press/v37/schaul15.pdf` (visited on 02/19/2018).

[130] John Schulman, Sergey Levine, Pieter Abbeel, Michael I. Jordan, and Philipp Moritz. "Trust Region Policy Optimization". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by PMLR. 2015, pp. 1889–1897. URL: http://proceedings.mlr.press/v37/schulman15.html (visited on 02/10/2021).

[131] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. *Proximal Policy Optimization Algorithms*. URL: http://arxiv.org/pdf/1707.06347v2.

[132] P. Shakouri, A. Ordys, D. S. Laila, and M. Askari. "Adaptive Cruise Control System: Comparing Gain-Scheduling PI and LQ Controllers". In: *IFAC Proceedings Volumes* 44.1 (2011), pp. 12964–12969. ISSN: 14746670. DOI: 10.3182/20110828-6-IT-1002.02250.

[133] Wenjie Shi, Shiji Song, Cheng Wu, and C. L. Philip Chen. "Multi Pseudo Q-Learning-Based Deterministic Policy Gradient for Tracking Control of Autonomous Underwater Vehicles". In: *IEEE Transactions on Neural Networks and Learning Systems* 30.12 (2019), pp. 3534–3546. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2018.2884797.

[134] Konstantin Shirokinskiy, Wolfgang Bernhard, and Stephan Keese. *Advanced Driver Assistance Systems: A Ubiquitous Technology for The Future of Vehicles*. Ed. by Roland Berger. 2021. URL: https://www.rolandberger.com/en/Insights/Publications/Advanced-Driver-Assistance-Systems-A-ubiquitous-technology-for-the-future-of.html (visited on 05/13/2021).

[135] Gary Silberg, Angelika Huber-Straßer, Peter Schalk, Aline Dodd, and Bernd Oppold. *Global Automotive Executive Survey 2020: Say goodbye to one global market - recognize increasing localization, accelerated by COVID-19*. Ed. by KPMG. 2020. URL: https://automotive-institute.kpmg.de/GAES2020/ (visited on 05/12/2021).

[136] David Silver, Guy Lever, Nicolas Heess, Thomas Degirs, Daan Wierstra, and Martin Riedmiller. "Deterministic Policy Gradient Algorithms". In: *Proceedings of Machine Learning Research*. Ed. by Eric P. Xing and Tony Jebara. PMLR, 2014. URL: http://proceedings.mlr.press/v32/silver14.pdf (visited on 09/08/2017).

[137] David Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science (New York, N.Y.)* 362.6419 (2018), pp. 1140–1144. DOI: 10.1126/science.aar6404.

[138] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529.7587 (2016), pp. 484–489. ISSN: 1476-4687. DOI: 10.1038/nature16961.

[139] David Silver et al. “Mastering the game of Go without human knowledge”. In: *Nature* 550.7676 (2017), pp. 354–359. ISSN: 1476-4687. DOI: 10.1038/nature24270.

[140] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. “Learning Without State-Estimation in Partially Observable Markovian Decision Processes”. In: *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 284–292. ISBN: 9781558603356. DOI: 10.1016/B978-1-55860-335-6.50042-8.

[141] *Specifications BMW 7 Series Sedan: valid from 11/2015*. Munich, 2015-09-16. URL: https://pressclub.bmwgroup.net/global/article/detail/T0235242EN/specifications-bmw-7-series-sedan-valid-from-11/2015 (visited on 10/03/2022).

[142] Robert A. Stine. “Model Selection Using Information Theory and the MDL Principle”. In: *Sociological Methods & Research* 33.2 (2004), pp. 230–260. ISSN: 0049-1241. DOI: 10.1177/0049124103262064.

[143] Yi Sun, Faustino Gomez, and Jürgen Schmidhuber. “Planning to Be Surprised: Optimal Bayesian Exploration in Dynamic Environments”. In: *Artificial General Intelligence*. Ed. by David Hutchison et al. Vol. 6830. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 41–51. ISBN: 978-3-642-22886-5. DOI: 10.1007/978-3-642-22887-2_5.

[144] Richard S. Sutton. “Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming”. In: *Machine Learning Proceedings 1990*. Elsevier, 1990, pp. 216–224. ISBN: 9781558601413. DOI: 10.1016/B978-1-55860-141-3.50030-4.

[145] Richard S. Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. 2nd. Adaptive computation and machine learning. Cambridge, MA and London: The MIT Press, 2018. ISBN: 9780262039246.

[146] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. In: *NIPS’99 Proceedings of the 12th International Conference on Neural Information Processing Systems*. Ed. by MIT Press Cambridge. Cambridge: MIT Press, 1999, pp. 1057–1063. URL:

https://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf (visited on 01/08/2018).

[147] Richard S. Sutton, Satinder P. Singh, and David A. McAllester. "Comparing Policy-Gradient Algorithms". 2000. URL: https://pdfs.semanticscholar.org/beef/9a0f27e4ca54eb5c5311f6ac90d90fa88f12.pdf.

[148] Haoran Tang et al. "#Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning". In: *NIPS 2017 Proceedings of Neural Information Processing Systems.* Ed. by MIT Press Cambridge. Vol. 30. Cambridge, 2017. URL: http://papers.nips.cc/paper/6868-exploration-a-study-of-count-based-exploration-for-deep-reinforcement-learning (visited on 08/26/2020).

[149] Inés Tejado, Vicente Milanés, Jorge Villagrá, Jorge Godoy, Hassan HosseinNia, and Blas M. Vinagre. "Low Speed Control of an Autonomous Vehicle by Using a Fractional PI Controller". In: *IFAC Proceedings Volumes* 44.1 (2011), pp. 15025–15030. ISSN: 14746670. DOI: 10.3182/20110828-6-IT-1002.01108.

[150] Stephan Terwen. *Vorausschauende Längsregelung schwerer Lastkraftwagen: Zugl.: Karlsruhe, Univ., Diss., 2009.* Print on demand. Vol. 6. Schriften des Instituts für Regelungs- und Steuerungssysteme, Karlsruher Institut für Technologie. Karlsruhe and Hannover: KIT Scientific Publ and Technische Informationsbibliothek u. Universitätsbibliothek, 2010. ISBN: 978-3-86644-481-2. URL: http://edok01.tib.uni-hannover.de/edoks/e01fn12/626362342.pdf.

[151] UNECE Transport Division, ed. *United Nations Global Technical Regulation.* Geneva, Switzerland. URL: https://wiki.unece.org/display/trans/Latest+GTR+15.

[152] Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-Learning". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence and the Twenty-Eighth Innovative Applications of Artificial Intelligence Conference.* Vol. 6. Palo Alto, California: AAAI Press, 2016, pp. 2094–2100. ISBN: 978-1-57735-760-5. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389 (visited on 02/08/2021).

[153] Hado Philip van Hasselt. *Insights in reinforcement learning: Formal analysis and empirical evaluation of temporal-difference learning.* Utrecht: Universiteit Utrecht, 2011. ISBN: 978-90-39354964.

[154] Herke van Hoof, Daniel Tanneberg, and Jan Peters. "Generalized exploration in policy search". In: *Machine Learning* 106.9-10 (2017), pp. 1705–1724. ISSN: 0885-6125. DOI: 10.1007/s10994-017-5657-1.

[155] Oriol Vinyals et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782 (2019), pp. 350–354. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1724-z.

[156] Peter Waldmann and Daniel Niehues. "Der BMW TrackTrainer: automatisiertes Fahren im Grenzbereich auf der Nürburgring Nordschleife". In: *Tagung Sicherheit durch Fahrerassistenz.* 2010. URL: http://www.ftm.mw.tum.de/uploads/media/38_waldmann.pdf (visited on 07/05/2017).

[157] Jian Wang, Xin Xu, Daxue Liu, Zhenping Sun, and Qingyang Chen. "Self-Learning Cruise Control Using Kernel-Based Least Squares Policy Iteration". In: *IEEE Transactions on Control Systems Technology* 22.3 (2014), pp. 1078–1087. ISSN: 1063-6536. DOI: 10.1109/TCST.2013.2271276.

[158] Shaochen Wang, Yuan Pu, Shangtong Yang, Xin Yao, and Bin Li. "Boltzmann Exploration for Deterministic Policy Optimization". In: *Neural Information Processing.* Ed. by Haiqin Yang, Kitsuchart Pasupa, Andrew Chi-Sing Leung, James T. Kwok, Jonathan H. Chan, and Irwin King. Vol. 12533. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 214–222. ISBN: 978-3-030-63832-0. DOI: 10.1007/978-3-030-63833-7_18.

[159] Ziyu Wang et al. "Sample Efficient Actor-Critic with Experience Replay". In: *5th International Conference on Learning Representations.* Ed. by Openreview.net. 2017. URL: https://openreview.net/forum?id=HyM25Mqel (visited on 02/10/2021).

[160] Daniel Watzenig and Martin Horn. *Automated Driving.* Cham: Springer International Publishing, 2017. ISBN: 978-3-319-31893-6. DOI: 10.1007/978-3-319-31895-0.

[161] Pawel Wawrzyński. "Control Policy with Autocorrelated Noise in Reinforcement Learning for Robotics". In: *International Journal of Machine Learning and Computing.* Ed. by Lin Huang. Vol. 5. 2015. URL: http://www.ijmlc.org/vol5/489-A16.pdf (visited on 08/26/2020).

[162] Greg Wayne et al. *Unsupervised Predictive Memory in a Goal-Directed Agent.* URL: http://arxiv.org/pdf/1803.10760v1.

[163] Hermann Winner, Stephan Hakuli, Felix Lotz, and Christina Singer. *Handbook of Driver Assistance Systems.* Cham: Springer International Publishing, 2016. ISBN: 978-3-319-12351-6. DOI: 10.1007/978-3-319-12352-3.

[164] Dongbin Xiu. *Numerical methods for stochastic computations: A spectral method approach.* Princeton, N.J: Princeton University Press, 2010. ISBN: 9780691142128. DOI: 10.2307/j.ctv7h0skv. URL: http://www.jstor.org/stable/10.2307/j.ctv7h0skv.

[165] Oded Yechiel, Gal Israeli, and Hugo Guterman. "Direct Adaptive Control Using a Neuro-evolutionary Algorithm for Vehicle Speed Control". In: *2018 IEEE International Conference on the Science of Electrical Engineering in Israel (ICSEE).* IEEE, 2018, pp. 1–5. ISBN: 978-1-5386-6378-3. DOI: 10.1109/ICSEE.2018.8646302. URL: https://ieeexplore.ieee.org/document/8646302 (visited on 08/24/2020).

[166] Runsheng Yu, Zhenyu Shi, Chaoxing Huang, Tenglong Li, and Qiongxiong Ma. "Deep reinforcement learning based optimal trajectory tracking control of autonomous underwater vehicle". In: *2017 36th Chinese Control Conference (CCC).* IEEE, 2017, pp. 4958–4965. ISBN: 978-988-15639-3-4. DOI: 10.23919/ChiCC.2017.8028138.

[167] Wu Yuhuai, Elman Mansimov, Roger B. Grosse, Shun Liao, and Jimmy Ba. "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation". In: *NIPS 2017 Proceedings of Neural Information Processing Systems.* Ed. by MIT Press Cambridge. Cambridge, 2017. URL: https://papers.nips.cc/paper/2017/hash/361440528766bbaaaa1901845cf4152b-Abstract.html (visited on 02/10/2021).

[168] Baohe Zhang et al. "On the Importance of Hyperparameter Optimization for Model-based Reinforcement Learning". In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics.* Ed. by PMLR. Proceedings of Machine Learning Research. 2021, pp. 4015–4023. URL: https://proceedings.mlr.press/v130/zhang21n.html (visited on 10/02/2022).

[169] Guanjie Zheng et al. "DRN". In: *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*. Ed. by Pierre-Antoine Champin, Fabien Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis. New York, New York, USA: ACM Press, 2018, pp. 167–176. ISBN: 9781450356398. DOI: 10.1145/3178876.3185994.

[170] Qingze Zou and Santosh Devasia. "Preview-Based Stable-Inversion for Output Tracking of Linear Systems". In: *Journal of Dynamic Systems, Measurement, and Control* 121.4 (1999), pp. 625–630. ISSN: 0022-0434. DOI: 10.1115/1.2802526.

A. Appendix

A.1. Optimizer

Several algorithms have been presented to solve optimization problems. This section quickly goes over the most popular gradient-based algorithms and introduces the Levenberg-Marquardt algorithm that is used throughout this work. At the end of this section we present step size limitation, an extension that can be applied to most optimizers.

Gradient Descent Gradient descent is a basic iterative optimization algorithm that consists of taking steps $\Delta\theta_t$ in the opposite direction of the gradient ${}^{\mathrm{d}L}/{}_{\mathrm{d}\theta}$, often with a step size proportional to the gradient norm:

$$\Delta\theta_t = -\alpha \nabla L\big|_{\theta_t}. \tag{A.1}$$

Here, t is a discrete index denoting the optimization step. The scalar α is commonly referred to as learning rate. The subsequent parameter estimate θ_{t+1} consists of the old estimate θ_t and the gradient step $\Delta\theta_t$:

$$\theta_{t+1} = \theta_t + \Delta\theta_t. \tag{A.2}$$

This algorithm depends on the hyperparameter α. A too small choice may require many optimization steps and therefore slow convergence, while a too high value can make the optimization process divergent, i.e. diverge from the optimum. The effect of the parameter depends on the optimization problem. Stochastic optimization may make the algorithm behave erratically if the gradient is affected by noise. In ideal conditions[83], gradient descent can solve problems that are linear-in-parameters in a single step and is therefore considered a first-order optimization technique.

[83] Ideal conditions include optimal choice in step size, noise-free and non-stochastic optimization.

Gradient Descent Based Algorithms The downsides of gradient descent were adressed by two extensions to the algorithm:

- AdaGrad [31],[48, Section 8.5.1] and RMSProp [48, Section 8.5.2] try to choose an individual **learning rate** for each entry in the gradient vector, therefore alleviating slow learning while avoiding divergence as far as possible.
- **Momentum** [121], [48, Section 8.3.2] is a popular technique to enhance robustness to noise in the gradient and local optima using a running average over past parameter steps.

Both techniques are combined in the popular adaptive moment estimation (ADAM) [76], [48, Section 8.5.3] optimizer. It counters the shortcomings of gradient descent optimization to some extent at low computational cost and is therefore considered state of the art in machine learning applications.

Second Order Optimization While ADAM may exhibit performance beyond regular first-order optimization, in some cases second-order optimization may still outperform it [48, Section 4.3.1]. Second-order optimization is based on Newton's method which relies on a second-order Taylor series expansion of the loss function to compute the parameter update $\Delta\theta_t$.

The Gauß-Newton algorithm makes use of the fact that for least-squares loss functions the required Hessian for the approximation can be constructed using the Jacobian [39].

The Levenberg-Marquardt (LM) algorithm [105], [48, Section 8.6.1] adds regularization to the (Gauß-) Newton step, effectively scaling between (Gauß-) Newton step and (small) gradient descent steps:

$$\Delta\theta_t = -\left[J_L^\top J_L + \lambda I\right]^{-1} \nabla L\bigg|_{\theta_t} . \tag{A.3}$$

The regularization factor λ is adapted in accordance with the success of the last step in the optimization: if the loss increases, the step is discarded, λ is increased (decreasing the step size and making the step more prone to a small gradient descent step). If the loss decreases, the step is accepted and λ is decreased, making the next step closer to a Gauß-Newton step.

The matrix inversion is expensive for large matrices, but may be feasible for applications with a comparably low number of variables to optimize.

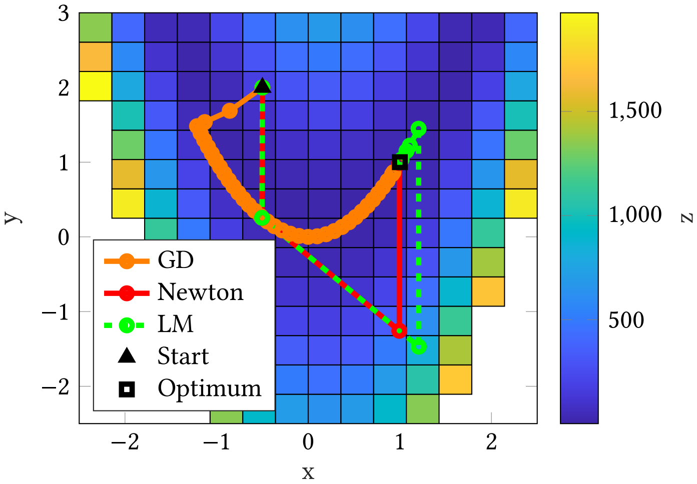


Figure A.1.: Example for trajectories of optimization algorithms finding the minimum of the Rosenbrock function [125]. While gradient descent with a learning rate of 0.0005 is not divergent in this example, it takes 8225 steps to get within a radius of 0.05 around the optimum at $\binom{1}{1}$. Newton's method and Levenberg-Marquardt both converge in less than 10 iterations.

An example for the behavior of exemplary implementations of the presented optimizers is given in Fig. A.1, where they are applied to the difficult to optimize Rosenbrock function. The figure shows that second order optimizers converge to a proximity of the optimum in significantly less steps.

Step Size Limitation Erroneous gradients may lead to misguided optimizer steps in stochastic optimization[84], introducing variance in the optimization process. In the worst case a single (possibly large) erroneous step may cause the entire optimization process to diverge. A possible countermeasure could be to limit the maximum step size an optimizer is allowed to take. While other methods exist, in this work we rely on norm clipping [111], i.e. we rescale the step $\Delta\theta_t$ if its norm $|\Delta\theta_t|$ exceeds a threshold $g_{\max}$ to ${}^{g_{\max}}/_{|\Delta\theta_t|}\Delta\theta_t$.

[84] This may be an effect of unfortunate sampling of minibatches from noisy data.

A.2. Motivation for Initializing the MB RL Feedforward Controller in its Steady State

Section 3.2.2 states that the MB RL Feedforward Controller was initialized in its steady state. This section provides background to this decision by laying out the available choices and their impact on the system during learning.

During learning, the feedforward controller has to be reset after every update to its parameters, because its internal state may not be meaningful with the feedforward dynamics after an update. With updates occuring every 600 ms, the reaction of the feedforward after initialization frequently affects the vehicle. The choice of initialization method is therefore an important one.

For initializing any simulated dynamic system the state can be chosen arbitrarily or filled with zeros - in both cases and indepentently of the input fed to the system its output generally cannot be predicted. In the context of learning longitudinal control this may amount to extreme controller commands that may trigger safety thresholds and deactivate the controller or cause drastic braking or acceleration. This behavior would lead to learning an ill-fitting model which in turn results in a poor controller.

This work therefore chooses to initialize the feedforward controller in its steady state. At this state, the output of the system remains constant - with the learned model forced to have a pole at 1, the steady state gain of the inverted system is 0, and thus the output in steady state is also 0. When following a constant target during learning, the feedforward is not contributing to the controller output. Periodic resets to its steady state following parameter updates therefore have no impact on its (non-existent) output. If the trajectory to follow varies over time, the feedforward contribution is non-zero. Periodically resetting its state, and thus its output to zero therefore can cause oscillating behavior during learning. Since resets do not occur once learning is deactivated, these oscillations were tolerated during learning, where they aided exploration.

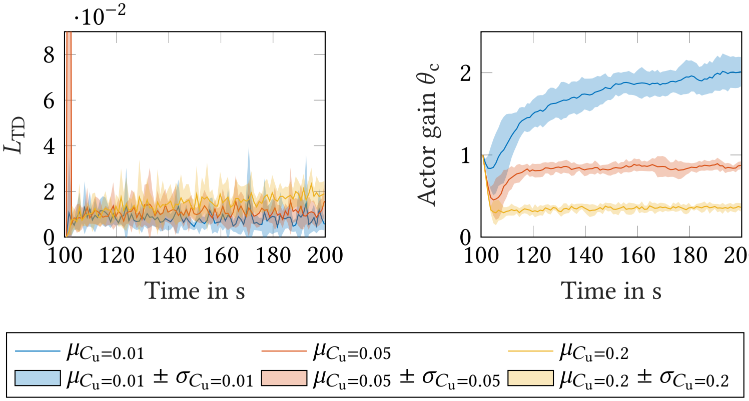


Figure A.2.: Averages and standard deviation intervals for 10 training runs with different values for reward parameter C_u. In classic control parameter C_u is used to tune the aggressivity of the controller. While lower values for C_u tend to lower the TD error, this has and adverse effect on policy learning: it increases variance and slows learning. Within this learning setup, the parameter C_u can therefore not be varied freely and the engineer once more has to trade bias for variance, if an aggressive controller is desired.

A.3. A Simulation Study on Hyperparameter Influence

Here, the complete results from the simulation study introduced in Section 4.1 are presented.

This part begins by varying the hyperparameters defining the optimality criterion for the controller, then it shows the effect of parameters for the optimization in actor and critic before presenting a study on the effects of noise both for exploration and in the environment. A graphical overview of the presented experiments can be found in Fig. 4.2. A tabular overview of the influences of the individual parameters is given in Table 4.1.

A.3.1. Hyperparameters in the Optimality Criterion

For discount factor γ, see Section 4.1 and Fig. 4.3.

While low values for parameter C_u encourage the agent to learn more aggressive controllers, it slightly increases variance (see actor gain in Fig. A.2) in the learning process and therefore makes it less stable. At first glance it seems counterintuitive that low values for C_u simultaneously increase variance in the actor gain and decrease TD error levels. However, lower TD errors are a direct consequence of lowering the impact of the action on state values, thus reducing the error potential from misadaptation of the critic. The actor gains exhibit higher variance regardless of that, since it is learned from the policy gradient which is derived from aforementioned lowered impact. Since the advantage component in the critic is reduced while the overall noise level remains equal, the resulting gradient is less accurate. The outcome can be seen in the actor gain progression in Fig. A.2: at lower C_u values learning is slower and affected by higher variance.

The parameter C_u is considered a tuning factor for how aggressive the controller shall be in classic optimal control. In vehicle experiments we found values around 0.025 to be the lower feasible limit. Values below this required excessive choices in other hyperparameters to achieve stable learning in the real vehicle.

A.3.2. Hyperparameters for the Optimization Steps

While LM provides quick learning, this comes at the cost of a somewhat noisy behavior. It tends to take big steps even in the proximity of the optimum in the presence of noise. This introduces variance in the controller parameter estimation. If a restrictive maxnorm is applied in the optimizer for the actor, this effect can be dampened at the cost of learning speed (see actor gain in Fig. A.3). On the contrary, if the limit does not constrain the actor optimizer, this may allow the actor to move faster than the critic can track it or cross the stability boundary, leading to failure of the learning process. This can be seen from our simulated experiment in Fig. A.3: with a very open maxnorm limit for the actor, not only TD errors become large, but it leads to several failed training runs. In the vehicle application, a restrictive maxnorm in the actor is therefore mandatory to ensure stability. In order to benefit from quick learning, a slightly higher limit is used when training begins and decayed during the learning process until it reaches a lower boundary.

In the context of critic optimization, the maxnorm constraint reverses its effect on learning stability: a liberal limit on the maxnorm generally increases

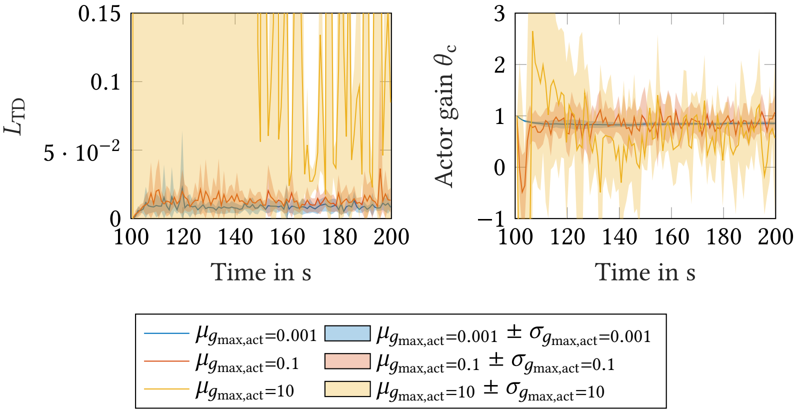


Figure A.3.: Average and standard deviation interval over 10 training runs for multiple values for actor maxnorm constraint. 6 of the training runs with maxnorm 10 failed and were subsequently excluded from the plot. High actor maxnorm limits can cause instability due to the critic not being able to track the value changes due to policy evolution. This can be seen from very high TD errors and failed test runs with actor maxnorm 10, and high variance in the actor gain θ_c. Lower values may slow learning. The actor maxnorm therefore trades off speed against variance and stability.

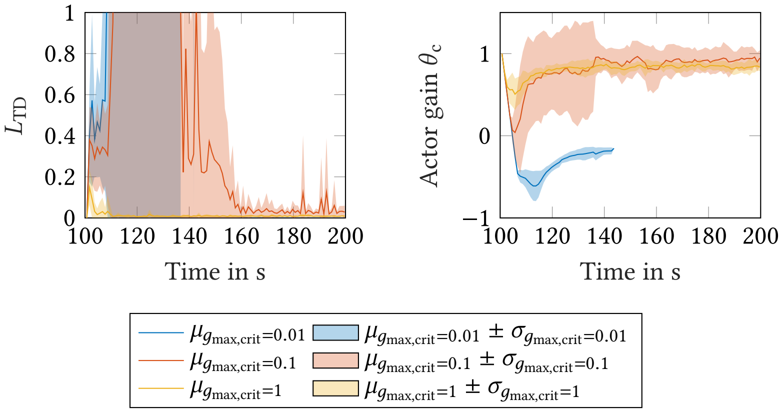


Figure A.4.: Critic maxnorm variation with 10 experiments per maxnorm value. The plot shows average and standard deviation interval for each maxnorm limit. While using maxnorm 1, every experiment run was stable, 1 run was unstable with maxnorm 0.1 and all 10 runs were broken off with maxnorm 0.01. This simulated experiment shows that high maxnorm limits aid stability in learning.

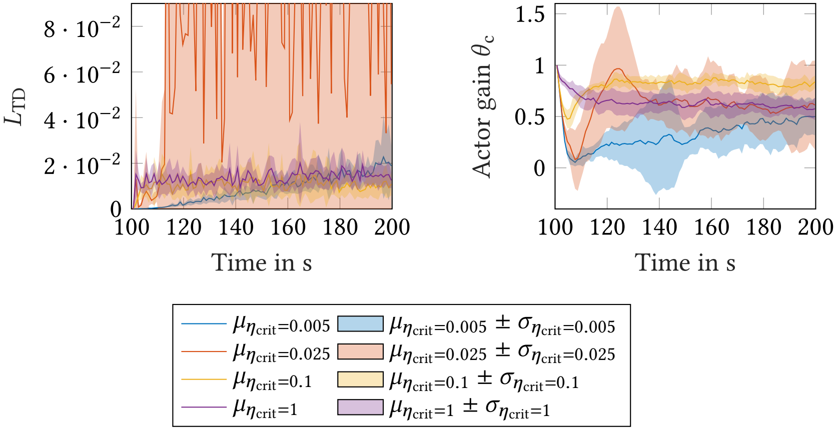


Figure A.5.: Average and standard deviation interval for critic target update rate variation. Very low values for target update rate make it harder for the critic to track the state-action value with high fidelity, therefore resulting in higher variance in both actor gain and TD error, thus slowing learning at very low target update rates. A target update rate of 1 results in fastest convergence, but may introduce bias due to overestimation of state-action values (see subsection 2.2.4, not visible from graph due to variance and slow learning).

stability, since it allows to quickly learn value changes resulting from a shift in the controller. This is visible from an opposite perspective in the TD error in Fig. A.4: the lowest critic maxnorm first causes the TD error to explode, then the runs are terminated due to instability. In the vehicle experiments it was observed that learning without any limit would become unstable again in some cases (not seen in the simulated experiments). However, regarding noise its effect is fully parallel to the actor: lower maxnorm constraints reduce variance, which generally enhances both learning speed and precision in the actor. In the experiments the critic optimizer was loosely constrained, yet not entirely left without a maxnorm limit to avoid (rare) cases of instability due to large erroneous parameter changes in the critic. Over the course of the experiment, the limit was slowly restrained to additionally reduce variance.

While target networks were originally introduced to aid stability in the learning process (see subsection 2.2.4), they can have a similar effect as slowing the critic optimizer: at very low target update rates the critic may fail to track the actor's learning progress. While [152] claim that target networks may help reduce bias in certain cases, no clear conclusion can be drawn from

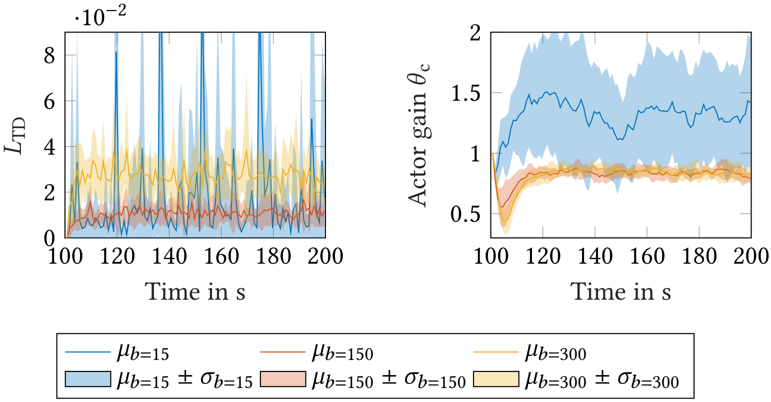


Figure A.6.: Average and standard deviation for training runs with different batch sizes. With the smallest batch size $b = 15$ variance was increased both in TD error and actor gain. A batch size of $b = 150$ yielded lower TD errors that $b = 300$, but similar results on the actor gain. Albeit the difference in TD error may be explained with overfitting to a small batch size, it is considered a negligible risk.

this experiment (see actor gain in A.5), mainly because learning was slowed due to high variance and therefore terminated before convergence (e.g. for $\eta_{\text{crit}} = 0.005$) or because of too high variance in the steady state ($\eta_{\text{crit}} = 0.025$). In the vehicle experiments an intermediate value of $\eta_{\text{crit}} = 0.1$ was chosen to not slow learning down too much while retaining some of the stabilizing effect.

Big batches help avoiding local minima and yield better gradient estimates which results in less variance in the learned policy, which can be seen best in the TD error plot in Fig. A.6: low batch sizes[85] yield low TD errors in some cases, but are stricken with high variance, suggesting that the critic approximator overfits to the small provided batches, but fails to match state transitions in other batches. Higher batch sizes reduce this effect, keeping a higher average TD error but with lower variance. Very small batch sizes may even compromise stability, while very large batch sizes bring little benefit, which is visible in the actor gains from Fig A.6. Batch size can therefore be seen as a way to balance between low variance and fast computation, i.e. quick learning (within the hardware limits).

[85] We used batches of identical size for training actor and critic.

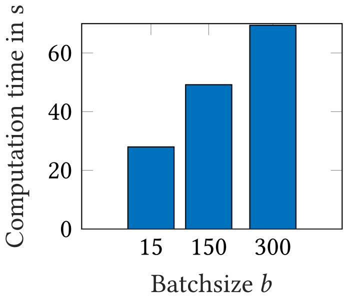


Figure A.7.: Average computation time for different batch sizes for 10 simulated training runs each. Logging and environment dynamics are included in the computation time measurement but can be considered as negligible. The analysis was performed on a Laptop containing an *Intel Core i5-8350* paired with 16 GB RAM using *Matlab R2018b* and *Windows 10* Build 17134. The setup differs both in hardware and in software from vehicle experiments and therefore reacts differently to the batch size, but the general trend towards higher computation times is valid in both cases. With faster computation times, the learning interval for the vehicle tests can be shortened, allowing for faster convergence and therefore shorter experiment duration.

However, higher batch sizes come at the cost of higher computation time[86] as Fig. A.7 shows. A batch size of $b = 150$ was chosen to balance precision in learning with low computational load.

A.3.3. Variation of Algorithm Architecture Elements

The number of steps for the TD backup (see Section 2.2.2) is often referred to as an example for a bias-variance trade-off (see e.g. [60]), i.e. more steps for backup should reduce bias in the learned policy at the cost of higher variance. The simulation experiments show that a higher number of steps for backup does significantly increase the TD error and slightly affects the learned gains (see A.8). We choose not to use more than one step for backup in the Bellman equation since the slight change in learned gain is not worth increasing the variance by orders of magnitude.

[86] Note that the hardware used in the vehicle differs from the one used in this comparison. Additionally, while this comparison relies on interpreter-based *Matlab* code while compiled generated code was employed in the vehicle experiment. The comparison can therefore only indicate a trend.

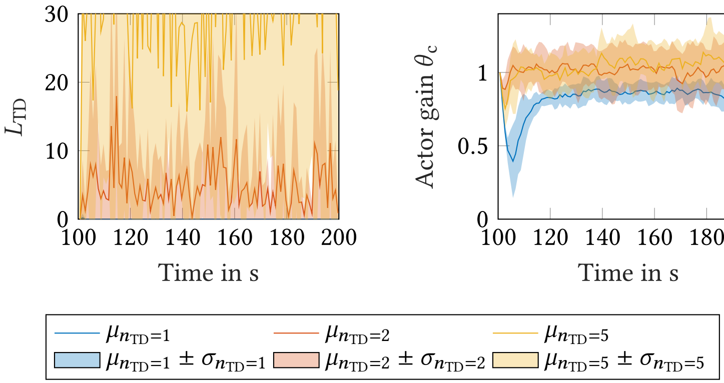


Figure A.8.: Average and standard deviation for training runs with different amounts of steps for backup. More steps for backup significantly increase variance in the TD error (plot for 1 step barely visible at the lower end of the plot), slightly change them average learned controller gain but add variance there, too.

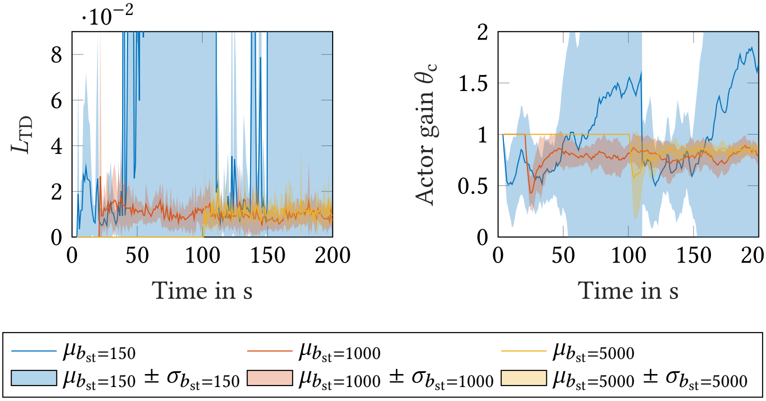


Figure A.9.: Experience storage size variation. The configuration with the smallest buffer size crossed the instability threshold once, but tends in that direction towards the end of the training run. Filling a large experience storage may take longer, but yields repeatably stable learning with little variance afterwards.

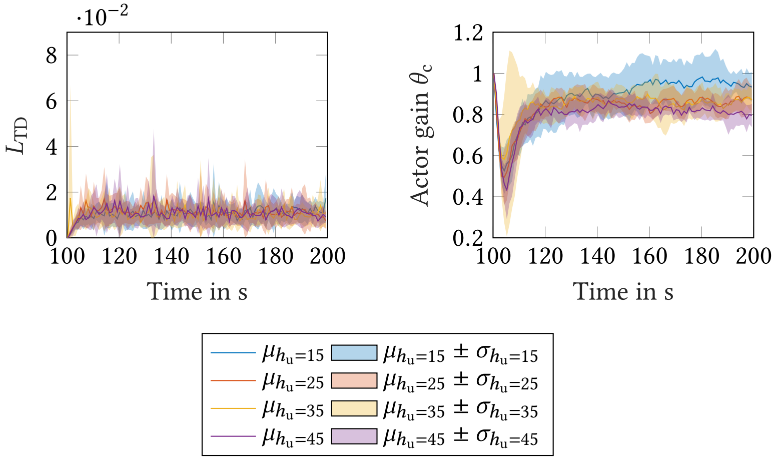


Figure A.10.: Mean and standard deviation interval for different FIR filter length settings. Short filters tend towards higher gains, but this effect vanishes towards higher filter lengths. There is no effect on the TD error.

The authors of [91] recommend using a large experience storage. The next experiment shows why: If the buffer is limited to the batch size, the agent may start to learn early on, but fail soon after. With a large experience storage, learning is repeatably stable. In the vehicle experiments, stable learning was achieved with a low amount of variance using the largest setting (5000 time steps). This helps increase robustness against short disturbances like bumps, ramps or lane changes. Experience storage size therefore can be used to increase stability and decrease variance at the expense of some waiting time before the learning process starts, i.e. effective learning speed.

In [119] we suggested to choose the length h_u of the FIR filter according to the impulse response of the system of interest. Fig. A.10 supports this: while all configurations have similar TD errors, shorter filter length settings yield higher gains. Filters with length equal to or greater 35 yield similar gains. When looking at the unit impulse response in Fig. A.11 it can be seen that the output is constant at around 35 time steps. Using no filter at all can increase variance up to the point of instability [119], but too short filters can yield biased results.

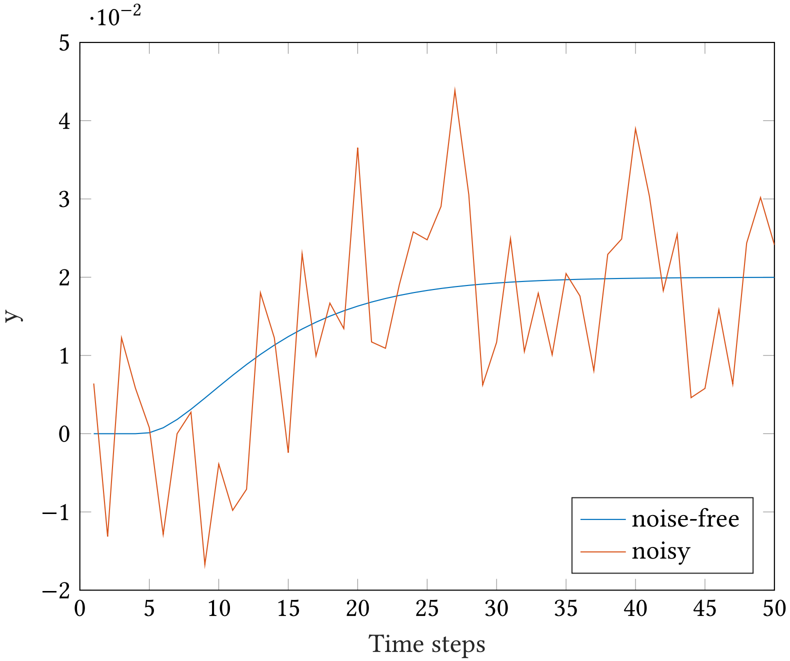


Figure A.11.: Response to a unit impulse for example system (1.3) with and without noise. Note that the time scale is in discrete time steps, not seconds. The initial response is flat due to delay. After roughly 35 time steps the output is roughly constant again.

A.3.4. Exploration Noise

Fig. A.12 shows the influence of exploration noise amplitude. While higher amplitudes increase the TD error, they benefit policy learning by making it quicker and more precise. If the amplitude is too low, the exploration noise may fail to excite the system, thus yielding data that is not rich enough to meaningfully fit the state-action value function to. This causes high variance in the learning process, increasing the risk of unstable learning. Very high amplitudes on the other hand enhance the learning process only slightly and may not be applicable to real-world systems without risking to damage the plant.

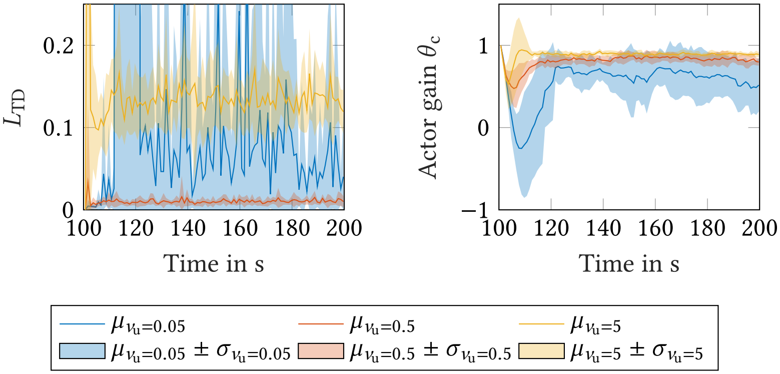


Figure A.12.: Mean and standard deviation of TD error and actor gain for training runs with different exploration noise amplitudes. Intermediate values for exploration noise perform best for TD-error values, TD error variance and actor gain variance. Very low values fail to excite the system, making it harder to learn without overfitting. The intermediate noise amplitude provides the agent with rich data to learn from, and further increasing the noise amplitude brings little benefit for learning: it increases the TD error, but only slightly increases learning performance in the actor gain. Very high amplitudes may not be feasible in real-world systems.

Despite the objective of exciting the system with exploration noise, learning should generally occur around the envisioned operation area of the controller to be learned. Since exploration noise is only added during learning, it may skew the resulting policy and should therefore be used carefully.

Exploration noise amplitude needs to balance stability and variance against bias while staying within the limits of the system.

Holding a value for the exploration noise for more than one time step can be used to shift the frequency spectrum towards lower frequencies. Depending on the system, this may not only prevent it from being harmed, but also enhance excitation. Holding for no more than one time step in this experiment results in higher variance than holding it for 50 time steps, i.e. 1 s, as can be seen from the actor gain evolution in Fig A.13. Longer hold phases deteriorate the learning performance again. The real-world vehicle has low-pass characteristics and may suffer from excessive wear if too high frequency signals are used as an input. We chose an intermediate value of 50 time steps for our experiments.

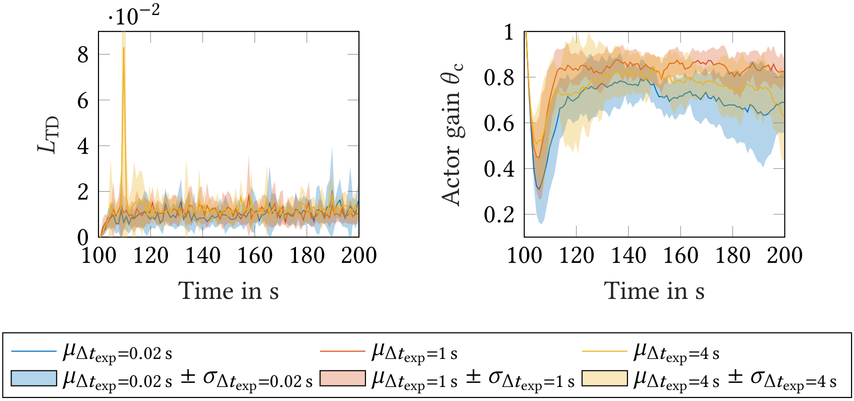


Figure A.13.: Exploration noise sampling time variation plotted with averages and standard deviation intervals. The sampling time allows to scale the signal between white noise and a step-like signal by sampling the random component used for exploration at a lower frequency. A hold factor of 1 is equivalent to sampling the exploration noise at each controller time step of 20 ms, a hold factor of 200 keeps the random part constant for 200 time steps (equivalent to 4 s) before resampling it. High hold factor values therefore emphasize low frequencies. The optimal hold factor depends on the system characteristics. In this example a hold factor of 1 or 200 results in higher variance in the actor gain compared to a hold factor of 50 (equivalent to 1 s).

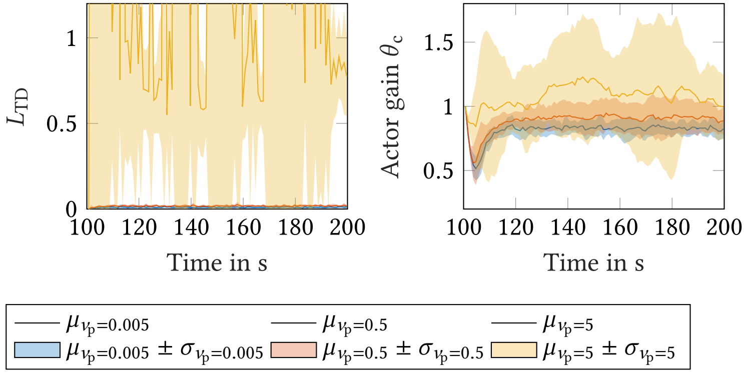


Figure A.14.: Average and standard deviation for TD error and actor gain in virtual trajectory noise amplitude size variation. Adding a random component was proposed to enhance learning speed for feedforward components, but in this case only increases variance and bias.

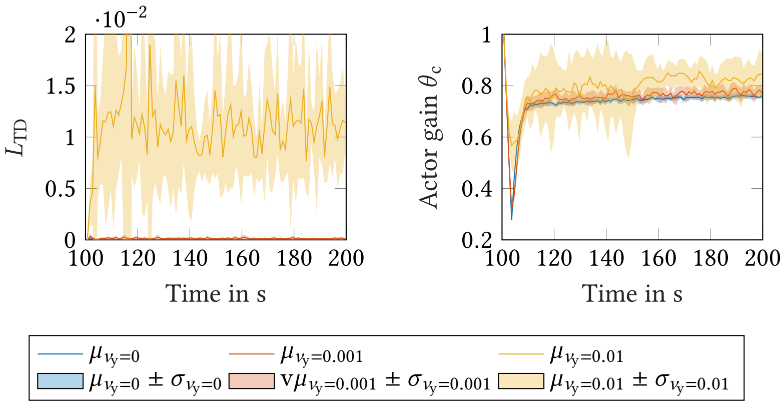


Figure A.15.: Effect of environment noise levels on TD error and actor gain. The higher the noise level in the environment, the higher the TD error becomes and the higher the variance in actor gains becomes.

Adding additional noise on the trajectory was proposed by [81] in order to accelerate learning of feedforward components in the controller, but only adds bias for this simple policy as Fig. A.14 shows. This is because it shifts the target in the training data to be more diverse for off-policy learning. This, however makes the distribution of training data differ from the actual use case, which may make the controller be suboptimal for the envisioned scenario. When learning a policy without a feedforward component virtual trajectory noise was therefore abandoned.

A.3.5. Environment Dynamics and Noise

The proposed algorithm is affected by environment noise as Fig. A.15 shows[87]. The noise level is reflected in the variance of the learned gain and the TD error level. This experiment only shows that it is desirable to have low levels of noise in the environment. In real-world systems the amount of noise generally cannot be influenced.

[87] The noise level chosen in for the linear system 1.3 was chosen to have a comparable amplitude as the measurement noise on the measurements in the vehicle setup, but this was not based on a thorough analysis, e.g. by Fourier transformation.

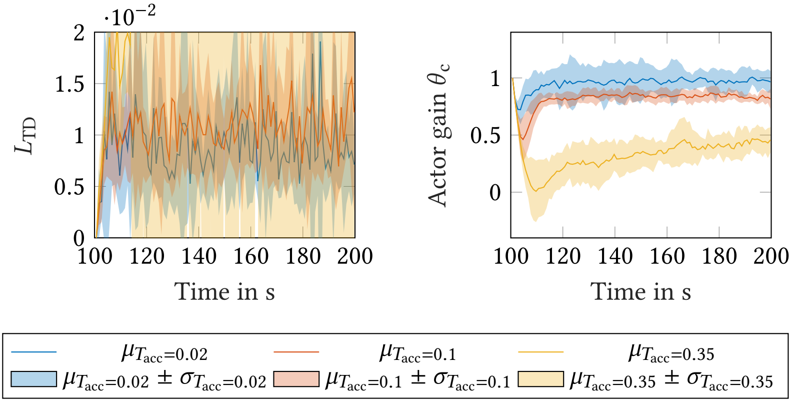


Figure A.16.: Effect on TD error and actor gain of environment time constant. The slower the system dynamics are, i.e. the harder it is to excite the system, the slower the learning process and the higher the variance.

The next simulated example presents how different dynamics affect the learning process in Fig. A.16. For the algorithm it seems to be easier to learn from comparably fast dynamics: with smaller time constants T_{acc}, the learning process converges earlier. This suggests that slow systems are harder to learn from for an RL agent. An integrator would pose the limiting case here: it can be seen as having an infinite time constant, and would thus be hardest to learn[88].

A.4. Hyperparameters Used in Experiments

In Table A.1 and its continuations we a provide tabular hyperparameter overview for each experiment.

[88] An RL agent generally cannot learn a policy if the policy it is currently training on is unstable in combination with the system since the value function has infinite values, i.e. the critic would diverge. For brief instants an unstable policy can be tolerated if the current policy is unstable, since the critic only slowly tracks the policy. For systems that cannot be stabilized with the policy structure, the learning process must diverge.

[89] The derivative of the value used for bootstrapping was included. See footnote 21 on page 32 for more details.

Table A.1.: Hyperparameters MF

Hyperparameter	Symbol	Set A	Set B
Intervals			
Controller sample time	Δt_{c}	0.02 s	0.02 s
Learning interval actor	$\Delta t_{\mathrm{l,act}}$	0.02 s	0.08 s
Learning interval critic	$\Delta t_{\mathrm{l,crit}}$	0.02 s	0.04 s
Exploration noise sample time	Δt_{exp}	0.02 s	0.02 s
Dimensions			
Preview length	h_{y}	–	1
Reference Polynomial Order	n_{p}	0	0
Experience Storage Capacity	b_{st}	1000	500
Number of appended actions	h_{u}	20	35
Batch size critic	b_{crit}	300	100
Batch size actor	b_{act}	150	100
Actor Optimization			
Actor optimizer type		SGD	LM
Actor learning rate	α_{act}	0.05	–
Actor initial maxnorm	$g_{\mathrm{max,act,0}}$	0.001	0.1
Actor minimum maxnorm	$g_{\mathrm{max,act,min}}$	0.001	0.1
Actor maxnorm decay rate	β_{act}	1	1
Critic Optimization			
Critic TD gradient type		double[89]	single
Steps for TD backup	n_{TD}	1	1
Method to eliminate FIR DOF		fix par.	weightnorm
Critic optimizer type		LM	LM
Critic learning rate	α_{crit}	–	–
Critic initial maxnorm	$g_{\mathrm{max,crit,0}}$	5	0.15
Critic minimum maxnorm	$g_{\mathrm{max,crit,min}}$	5	0.15
Critic maxnorm decay rate	β_{crit}	1	1
Critic target update rate	η_{crit}	1	1
Learning Goal			
Discount factor	γ	1	0.95
Reward weight control error	C_{y}	1	1
Reward weight control effort	C_{u}	0.1	0.1
Noise Configuration			
Distribution exploration noise		uniform	uniform
Exploration noise amplitude	ν_{u}	0.2	2
Distribution target noise		–	–
Standard deviation target noise	ν_{p}	–	–

Table A.1.: Hyperparameters MF (ctd.)

Hyperparameter	Symbol	Set C	Set D
Intervals			
Controller sample time	Δt_{c}	0.02 s	0.02 s
Learning interval actor	$\Delta t_{\mathrm{l,act}}$	2 s	0.6 s
Learning interval critic	$\Delta t_{\mathrm{l,crit}}$	1 s	0.6 s
Exploration noise sample time	Δt_{exp}	3 s	2 s
Dimensions			
Preview length	h_{y}	1	1, 2
Reference Polynomial Order	n_{p}	0	0, 1
Experience Storage Capacity	b_{st}	200	5000
Number of appended actions	h_{u}	35	35
Batch size critic	b_{crit}	100	200
Batch size actor	b_{act}	100	200
Actor Optimization			
Actor optimizer type		LM	LM
Actor learning rate	α_{act}	–	–
Actor initial maxnorm	$g_{\mathrm{max,act,0}}$	0.01	0.01
Actor minimum maxnorm	$g_{\mathrm{max,act,min}}$	0.01	0.01
Actor maxnorm decay rate	β_{act}	1	1
Critic Optimization			
Critic TD gradient type		single	single
Steps for TD backup		1	1
Method to eliminate FIR DOF		weightnorm	weightnorm
Critic optimizer type		LM	LM
Critic learning rate	α_{crit}	–	–
Critic initial maxnorm	$g_{\mathrm{max,crit,0}}$	0.1	10
Critic minimum maxnorm	$g_{\mathrm{max,crit,min}}$	0.1	10
Critic maxnorm decay rate	β_{crit}	1	1
Critic target update rate	η_{crit}	1	1
Learning Goal			
Discount factor	γ	0.85	0.95
Reward weight control error	C_{y}	1	1
Reward weight control effort	C_{u}	0.025	0.1
Noise Configuration			
Distribution exploration noise		rectangle	uniform
Exploration noise amplitude	ν_{u}	$0.7\,\mathrm{m\,s^{-2}}$	$1\,\mathrm{m\,s^{-2}}$
Distribution target noise		–	gaussian
Standard deviation target noise	ν_{p}	–	1

Table A.1.: Hyperparameters MF (ctd.)

Hyperparameter	Symbol	Set E
Intervals		
Controller sample time	Δt_{c}	0.02 s
Learning interval actor	$\Delta t_{\mathrm{l,act}}$	0.6 s
Learning interval critic	$\Delta t_{\mathrm{l,crit}}$	0.6 s
Exploration noise sample time	Δt_{exp}	1.5 s
Dimensions		
Preview length	h_{y}	1
Reference Polynomial Order	n_{p}	0
Experience Storage Capacity	b_{st}	5000
Number of appended actions	h_{u}	35
Batch size critic	b_{crit}	150
Batch size actor	b_{act}	150
Actor Optimization		
Actor optimizer type		LM
Actor learning rate	α_{act}	–
Actor initial maxnorm	$g_{\mathrm{max,act,0}}$	0.01
Actor minimum maxnorm	$g_{\mathrm{max,act,min}}$	0.001
Actor maxnorm decay rate	β_{act}	0.995
Critic Optimization		
Critic TD gradient type		single
Steps for TD backup		1
Method to eliminate FIR DOF		weightnorm
Critic optimizer type		LM
Critic learning rate	α_{crit}	–
Critic initial maxnorm	$g_{\mathrm{max,crit,0}}$	10
Critic minimum maxnorm	$g_{\mathrm{max,crit,min}}$	0.25
Critic maxnorm decay rate	β_{crit}	0.999
Critic target update rate	η_{crit}	0.1
Learning Goal		
Discount factor	γ	0.95
Reward weight control error	C_{y}	1
Reward weight control effort	C_{u}	0.05
Noise Configuration		
Distribution exploration noise		uniform
Exploration noise amplitude	ν_{u}	$0.5\,\mathrm{m\,s^{-2}}$
Distribution target noise		–
Standard deviation target noise	ν_{p}	–

Table A.2.: Hyperparameters MB Algorithm from [120].

Hyperparameter	Symbol	Value
Intervals		
Controller sample time	Δt_c	0.02 s
Learning interval actor	$\Delta t_\mathrm{l,act}$	0.6 s
Learning interval critic	$\Delta t_\mathrm{l,crit}$	0.6 s
Exploration noise sample time	Δt_exp	1.5 s
Dimensions		
Preview length	h_y	1
Experience Storage Capacity	b_st	5000
Batch size critic	b_crit	200
Batch size actor	b_crit	20
Model order	n_m	20
Controller Optimization		
Actor optimizer type		LM
Initial Maxnorm controller	$g_\mathrm{max,c,0}$	0.015
Decay factor maxnorm controller	β_c	1
Minimum maxnorm controller	$g_\mathrm{max,c,min}$	0.015
Model Optimization		
Number of time steps to predict		1
Initial Maxnorm model	$g_\mathrm{max,m,0}$	0.1
Decay factor maxnorm model	β_m	0.995
Minimum maxnorm model	$g_\mathrm{max,c,min}$	0.001
Learning Goal		
Discount factor	γ	0.95
Reward weight control error	C_y	1
Reward weight control effort	C_u	0.05
Noise Configuration		
Distribution exploration noise		uniform
Exploration noise amplitude	ν_u	$0.5\,\mathrm{m\,s^{-2}}$

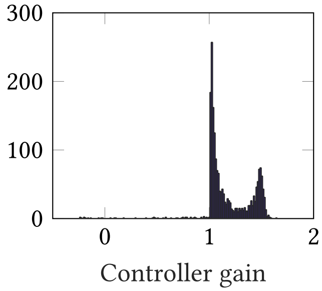


Figure A.17.: Histogram of optimal output gain for 2000 randomly chosen initial states within the unit cube from truncated policy search. The distribution sports two discrete maxima and spreads over a wide range of values. The optimal gain strongly depends on the initial state.

A.5. A Simulation Study on Optimal Output Control Gains

While for the fully observed linear case the optimal controller is valid throughout the entire state-space, this is not the case for optimal output control. This section aims to illustrate this by computing the optimal output feedback gain for a noise-free variant of the system (1.3) using (truncated) policy search (see subsections 2.2.3 and 3.2.2).

First the optimal output feedback gain is computed for 2000 randomly distributed initial states within the unit cube. Fig. A.17 shows that the optimal gain is multimodal and varies over a wide range depending on the initial state.

Next, the initial state is sampled over grids in the coordinate planes. Fig. A.18 shows strong dependence of the optimal output feedback gain over the initial state position in the state space.

This implies that even in the best case a learned gain can only be considered optimal in a close proximity of the conditions it was learned in. For the controller to be learned within this work it can therefore be assumed that the optimal controller depends on the distribution of states seen during training, which is affected by external factors from the experiment, e.g. road slope, or from the target trajectory and exploration noise.

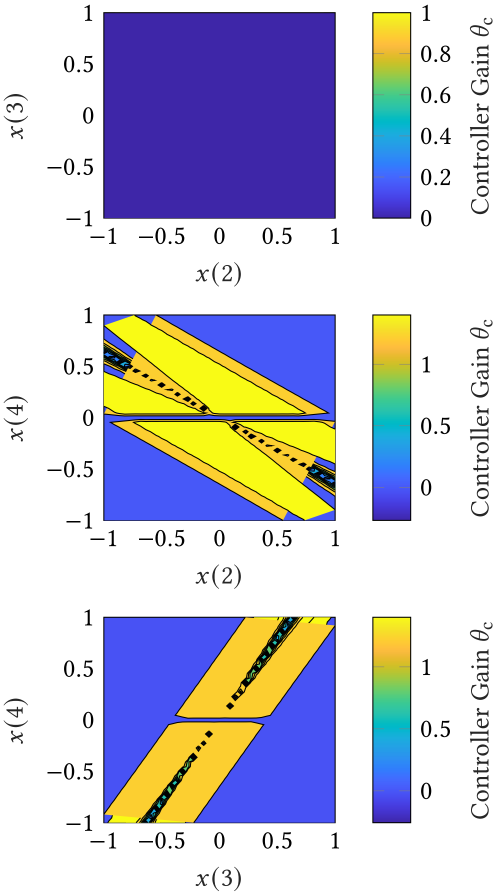


Figure A.18.: Optimal output gain over initial state in the coordinate planes; delay initialized with zero. The optimal gain in the $x(2)$-$x(3)$-plane is zero throughout, but assumes different values along trajectories in the $x(2)$-$x(4)$-plane and $x(3)$-$x(4)$-plane.

A.6. Validation of Example Gains in the Real Car

This section aims to understand if the controller chosen by the RL algorithm is plausible. For this, the learning goal of the controller is approximated using data of test runs using controllers with a series of different gains. The results suggest that the controller may be close to optimal. However, this estimation relies on several approximations that are necessary for feasibility but limit precision. Therefore these estimations cannot be considered as proof for optimality, but can instead provide some insights from recorded trajectories that make the algorithm's decision plausible.

First, the approximation of the controller's learning goal (2.7) is introduced. To this end, (2.7) is expanded to

$$H = \int \hat{V}^{\pi}(s) \int(s) \mathrm{d}s. \tag{A.4}$$

However, it is difficult to simultaneously obtain $\hat{V}^{\pi}(s)$ and $\int(s)$ from experimental data:

- The goal is to recreate the value estimation of our RL algorithm. Due to the data manipulation proposed in Section 3.1.2, the state value is estimated as if the assumed trajectory was followed on an infinite horizon. In this case, this would be a constant target speed $\hat{y}$ and no exploration noise. The ideal experiment would therefore be to start the controller in a random starting condition and leave it activated without exploration noise forever, which is not feasible, especially with a large number of initial conditions to test. Additionally, it might be difficult to reach the desired starting point in the real-world experiment. The observed state distribution in such an experiment would starkly deviate from what the controller observes during learning.
- The state distribution during learning results from excitation with exploration noise and changing setpoints. In this scenario, however, it is impossible to generate the ideal trajectories necessary for computing state values, since the controller actions taken are off-policy.

An intermediate experiment is therefore conducted: The output controller is used without noise to follow the step-like trajectory[90], balancing excitation from setpoint changes with settling periods in which the target remains constant. At least 10 setpoint change cycles are recorded for an array of control gains. The target and the resulting vehicle trajectories are given in

[90] The same type of trajectory was used during our training experiments.

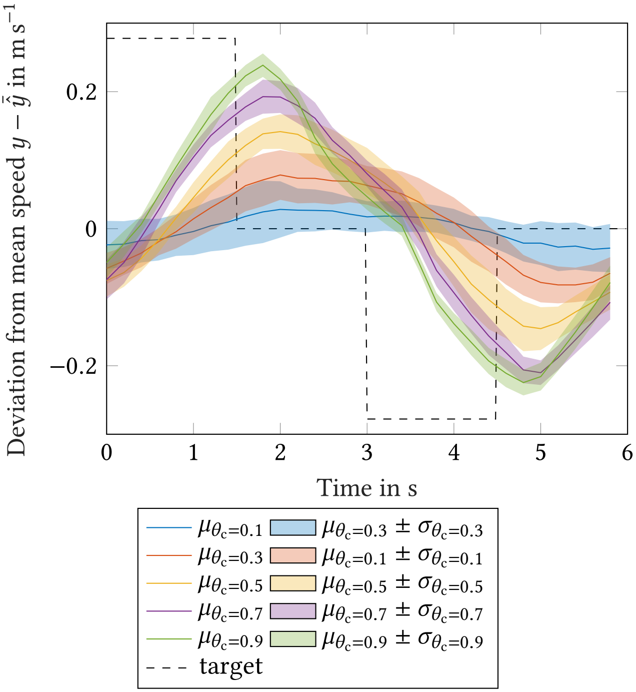


Figure A.19.: Average μ and standard deviation interval $\mu \pm \sigma$ of speed measurements for different controller gains following the trajectory used for training. Lower gains θ_c tend to make the vehicle stay close to the average speed $\bar{\hat{y}}$, higher gains make the vehicle deviate from it to follow the variation Δy.

Fig. A.19.

From this data, the controller values (A.4) are approximated for an array of controller gains. The continuous integral in (A.4) is replaced with a finite sum over weighted state values. For this, multiple approximations are taken:

1. Instead of considering the entire multi-dimensional state space, only the control error is considered, which is divided into $n = 26$ bins. Due to the limited data from the experiments, considering the entire space S would not yield a meaningful distribution.

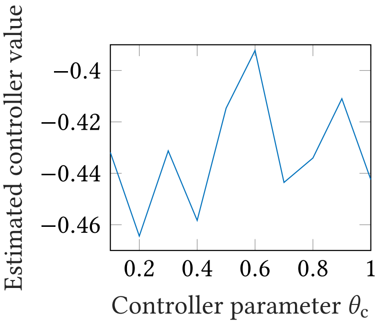


Figure A.20.: Estimated truncated controller values from trajectories averaged over experienced state distribution. According to this estimation the optimal controller is close to 0.5, suggesting that the learned controllers are optimal.

2. The state value is approximated by an interpolation of the truncated state value at the geometrical mean each interval. Truncation of the infinte sum defining the state value (see Section 2.2.1) is necessary for feasibility. The sum is truncated after 51 elements. This value is again a compromise: a longer series gives a more precise estimation of the state value[91], but limits the number of state values that can be computed from a recorded trajectory[92]. To dampen variance from the limited sample size and ensure the availability of a state value for each bin we apply a gaussian filter to the curve of state values over control error and interpolate the state value at the geometrical mean of each interval.

3. The probability density $\int$ is approximated with a quantized relative frequency of values within these intervals, which can be obtained from experimental data.

According to this estimation (see Fig. A.20), the optimal controller is close to 0.5, suggesting that the gain learned by both algorithms in the baseline experiment is close to optimal. However, we cannot take this result as proof

[91] With a discount factor of $\gamma = 0.9$, the accumulated discount is small: $\gamma^{51} < 0.005$.

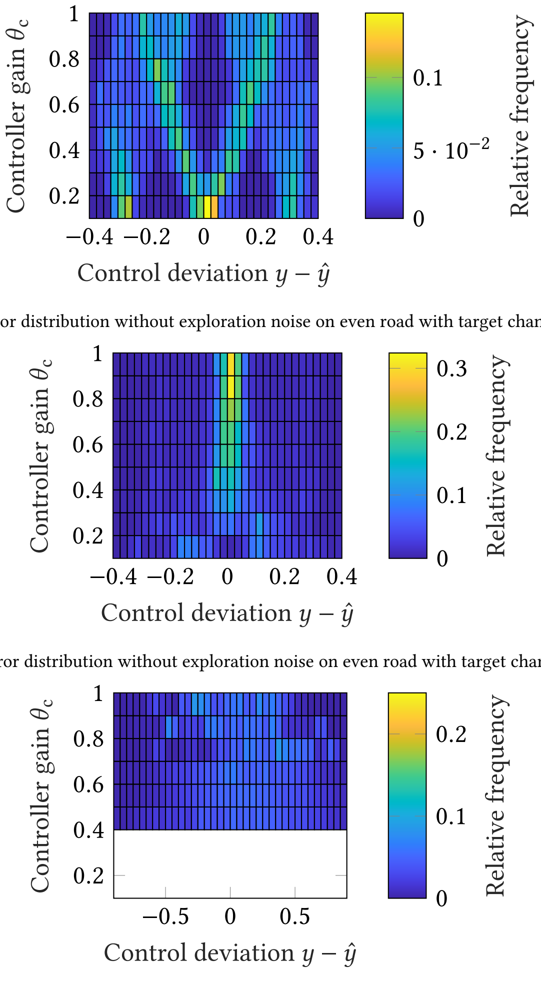
[92] A trajectory to compute a valid state value from must not contain a setpoint change. Due to the repeating target trajectory, the portion of states a state value can be computed for is limited.

due to the multitude of sources for inaccuracy.

Instead a few strategies for optimizing the learning objective can be pointed out from the obtained trajectories that can help to understand the algorithm's choice.

The control error distribution for the validation experiment gives insight to the different strategies to balance when following this varying target using output control. Since the target changes very frequently, trying to follow it closely, i.e. using a high controller gain, may not be beneficial for the control error distribution (see Fig. A.21a). While high gains yield a distribution containing both high and low control errors, keeping the speed constant and using almost no control effort, i.e. a low controller gain, yields three distinct maxima in the control error distribution: since the target assumes three different values, but the speed stays mostly constant, the control error is either close to 0 or equal to the variation of the target speed. Note that this strategy works best if disturbances are low (cf. section 4.3). Controllers in the mid range trade these effects off, seldom reaching a control error close to 0, but also avoiding very high control errors. This effect is also visible in the average state measurements in Fig. A.19. Fig. A.21 shows that exploration noise spreads the experienced control errors over a wider band while changing the control target less frequently has the opposite effect.

It is therefore not possible to prove the optimality of the learned gain from this experiment, but it appears plausible that trading off control effort against control error favors an intermediate gain in line with the algorithm's choice.

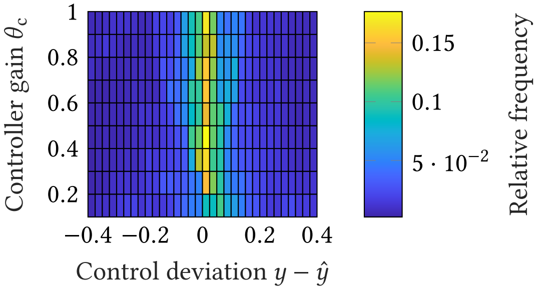


(a) Control error distribution without exploration noise on even road with target change every 1.5 s.

(b) Control error distribution without exploration noise on even road with target change every 10 s.

(c) Control error distribution with exploration noise on even road with target change every 1.5 s.

Figure A.21.: Control error distributions over gain for example use cases. The experienced control error is strongly influenced by the choice of target trajectory, the presence of disturbances and exploration noise.



(d) Control error distribution without exploration noise and high disturbances but constant target.

Figure A.21.: (ctd.) Control error distributions over gain for example use cases. The experienced control error is strongly influenced by the choice of target trajectory, the presence of disturbances and exploration noise.

Luca **Puccetti**

Control engineer with background in automotive engineering and experience in reinforcement learning.

- **Project Manager for Development of HMI Components**
  Mar 2021 - today
  BMW AG
- **PhD Self Learning Longitudinal Control for Vehicles**
  Jun 2017 – Mar 2021
  BMW AG/Karlsruhe Institute of Technology
- **Master's Degree in Automotive Engineering**
  Oct 2014 – Mar 2017
  Technical University of Munich
- **Bachelor's Degree in Automotive Engineering**
  Oct 2011 – Dec 2014
  Technical University of Munich
- **Internships**
  Jan – Jul 2011
  BMW AG
  Knorr Bremse
- **Military Service**
  Jul 2010 – Dec 2010
- **Abitur**
  Jun 2010

E-Mail
luca.puccetti@bmw.de
Phone
+151 601 38427