Advanced Seminar Control Theory

Machine Learning for Optimization-Based Design of Stabilizing Feedback Laws for Nonlinear Dynamical Systems

Max Pritzkoleit, Patrick Rüdiger
Institute of Control Theory
Faculty of Computer Engineering
TU Dresden
{max.pritzkoleit, patrick.ruediger}@mailbox.tu-dresden.de

Abstract

We compare the linear-quadratic regulator (LQR), a classical method of linear control theory, with model based and model free machine learning methods for optimization based design of stabilizing feedback laws for nonlinear dynamical systems. We test the presented methods on a wheeled inverted pendulum mobile robot and show that they are capable of stabilizing the system in an unstable equilibrium.

Keywords: *Machine Learning, Reinforcement Learning, Imitation Learning, Neural-Fitted Q-Iteration (NFQ), Optimal Control, LQR, Nonlinear Dynamical Systems*

1. INTRODUCTION

In the winter semester 2017/18 we were part of the advanced seminar in the diploma degree program in electrical engineering at TU Dresden, Germany and studied methods from machine learning and their application in control theory.

In the last decade, huge progress has been achieved in different research areas through the application of machine learning [1]. With this background in mind, it seems promising to investigate how some of these methods can be applied to control theoretic problems.

The two methods we studied come from the areas of model-free reinforcement learning and imitation learning. In both cases the goal is to stabilize a nonlinear dynamical system through optimization based feedback design in an unstable equilibrium.

The method of model-free reinforcement learning has been chosen, because it does not use a model of the controlled dynamical system. Therefore modeling of the dynamics, which can be a time consuming part of feedback design can be neglected. Nethertheless for our simulation study a model had to be derived. If we had applied the algorithm to a real system, this would not have been necessary.

The second method combines methods from supervised learning and trajectory optimization. The goal is to design a stabilizing feedback law in the form of an evaluable function, which approximates the results of an offline trajectory optimization for different initial conditions with sufficient accuaracy.

We compare the methods we applied to a classical LQR design on a wheeled inverted pendulum, a segway-like robot in terms of controller performance and implementation complexity. All studies have been carried out in simulation using *Python*.

In section 2 we lay out the theoretical background of the studied methods to describe them in section 3 and compare them to each other in section 4.

2. THEORETICAL BACKGROUND

2.1. OPTIMAL CONTROL

Given is an unconstrained optimization problem of the following form

$$\min_{\mathbf{u}(\cdot)} J(\mathbf{u}) = k\left(\mathbf{x}_\mathrm{f}\right) + \int_{t=t_0}^{t_\mathrm{f}} l\left(\mathbf{x},\mathbf{u}\right) dt \tag{1}$$

with state $\mathbf{x}(t) \in \mathbb{R}^n$, control $\mathbf{u}(t) \in \mathbb{R}^m$, arbitrary initial state $\mathbf{x}_0$, free end state $\mathbf{x}_\mathrm{f}$ and fixed final time t_f. The state trajectory is constrained by the system dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u})$ at every point in time $t \in [t_0, t_\mathrm{f}]$. The total cost J in (1) is composed of a final cost $k\left(\mathbf{x}_\mathrm{f}\right)$ and an arbitrary integral part $l\left(\mathbf{x},\mathbf{u}\right)$.

The problem at hand is a dynamic optimization problem. The system dynamics can be included in the cost function by using a Lagrange multiplier $\boldsymbol{\lambda}(t) \in \mathbb{R}^n$:

$$\bar{J}(\mathbf{u}) = k\left(\mathbf{x}_\mathrm{f}\right) + \int_{t=t_0}^{t_\mathrm{f}} \Big(l\left(\mathbf{x},\mathbf{u}\right) + \boldsymbol{\lambda}^\mathrm{T}(\mathbf{f}(\mathbf{x},\mathbf{u}) - \dot{\mathbf{x}}) \Big)\, dt \tag{2}$$

Application of the calculus of variations to $\bar{J}$ and introduction of the Hamiltonian $H(\mathbf{x},\mathbf{u},\boldsymbol{\lambda}) = l\left(\mathbf{x},\mathbf{u}\right) + \boldsymbol{\lambda}^\mathrm{T}\mathbf{f}(\mathbf{x},\mathbf{u})$ leads to

the necessary conditions of optimality for the optimal control problem defined in (1). [2, p. 89ff.]:

$$\dot{\mathbf{x}} = \frac{\partial H}{\partial \boldsymbol{\lambda}} \tag{3a}$$

$$\dot{\boldsymbol{\lambda}} = -\frac{\partial H}{\partial \mathbf{x}} \tag{3b}$$

$$0 = \frac{\partial H}{\partial \mathbf{u}} \tag{3c}$$

Additionaly to the initial conditions

$$\mathbf{x}(t_0) = \mathbf{x}_0 \tag{4a}$$

in the case of a free final state with fixed end time a final condition has to be fulfilled:

$$\boldsymbol{\lambda}(t_\mathrm{f}) = \frac{\partial k\,(\mathbf{x}_\mathrm{f})}{\partial \mathbf{x}_\mathrm{f}} \tag{4b}$$

By taking the necessary conditions of optimality (3) and the final condition (4) a boundary value problem (BVP) can be constructed. The solutions to this BVP are potentially[1] optimal trajectories $\mathbf{x}^*(t)$, $\boldsymbol{\lambda}^*(t)$ and $\mathbf{u}^*(t)$ on $[t_0, t_\mathrm{f}]$.

2.1.1. Potentially optimal trajektories for nonlinear dynamical systems. By solving condition (3c) for $\mathbf{u}$, $\mathbf{u}$ can be described as a function of $\mathbf{x}$

$$\mathbf{u} = \boldsymbol{\phi}(\mathbf{x}, \boldsymbol{\lambda}) \tag{5}$$

Substituting (5) in (3a) and (3b) leads to a BVP in $\mathbf{x}$ and $\boldsymbol{\lambda}$. In general this problem can only be solved numerically, but solution methods are quite complex [2, p. 120ff.]. The solution are trajectories of $\mathbf{x}^*(t)$ and $\boldsymbol{\lambda}^*(t)$. A potentially optimal control trajectory results from substitution of $\mathbf{x}^*(t)$ and $\boldsymbol{\lambda}^*(t)$ in (5).

2.1.2. Optimal state-feedback for linear time-invariant (LTI) systems with quadratic cost. For linear time-invariant systems of the form $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$ with $(\mathbf{A}, \mathbf{B})$ stabilizable, $l\,(\mathbf{x}, \mathbf{u}) = \mathbf{x}^\mathrm{T}\mathbf{Q}\mathbf{x} + \mathbf{u}^\mathrm{T}\mathbf{R}\mathbf{u}$ with symmetric, positive definite weight matrices $\mathbf{Q}$ and $\mathbf{R}$ with $(\mathbf{A}, \mathbf{C})$ detectable, where $\mathbf{Q} = \mathbf{C}^\mathrm{T}\mathbf{C}$ and $t_\mathrm{f} \to \infty$ and therefore $\mathbf{x}(t_\mathrm{f}) = 0$ and $k\,(\mathbf{x}_\mathrm{f}) = \mathbf{x}_\mathrm{f}^\mathrm{T}\mathbf{S}\mathbf{x}_\mathrm{f} = 0$, the optimal control $\mathbf{u}^*(t)$ for all t is given by the optimal feedback law

$$\mathbf{u}^*(t) = -\mathbf{K}^*\mathbf{x}(t), \tag{6}$$

with the optimal feedback matrix

$$\mathbf{K}^* = \mathbf{R}^{-1}\mathbf{B}^\mathrm{T}\mathbf{P}, \tag{7}$$

where $\mathbf{P}$ is the positive semidefinite solution of the algebraic Riccati equation[2, p. 102ff.]:

$$\mathbf{P}\mathbf{A} + \mathbf{A}^\mathrm{T}\mathbf{P} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^\mathrm{T}\mathbf{P} + \mathbf{Q} = 0. \tag{8}$$

Such a controller is called linear-quadratic regulator (LQR) and widely used in practice.

[1] Trajectories that fullfill the necessary conditions of optimality in (3) are called „potentially optimal“. Optimality of these trajectories may not be given, because the conditions in (3) are not guaranteed to be sufficient.

With the described limitations in mind follow the existence and uniqueness of the optimal state feedback, which drives the system from arbitrary initial condtions to the equilibrium (origin). Whereas the more general case in subsubsection 2.1.1 only leads to a optimal control trajectory for certain initial conditions. For the solution of the ARE, efficient solvers are available.

2.2. REINFORCEMENT LEARNING

Reinforcement learning is a subfield of machine learning, besides supervised and unsupervised learning [3]. It is theoretically and historically related with optimal control theory and can be viewed as direct adaptive optimal control [4].

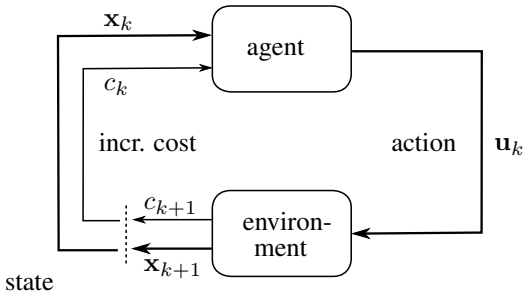

Figure 1. The agent-environment interaction of reinforcement learing [3, Fig. 3.1]

2.2.1. Agent-environment interaction. In reinforcement learning an agent interacts with an environment (Figure 1). At time step k the environment is in state $\mathbf{x}_k$ and the agent can take an action (control) $\mathbf{u}_k \in \mathcal{U}$. This results in a new state $\mathbf{x}_k \to \mathbf{x}_{k+1}$ and an incremental cost $c_{k+1} \in \mathbb{R}$ signal. The goal of the agent is to find the optimal action or control sequence $\mathbf{u}_0^*, \mathbf{u}_1^*, ..., \mathbf{u}_T^*$ from an initial state $\mathbf{x}_0 \in \mathcal{X}$ to an end state $\mathbf{x}_T \in \mathcal{X}$, which minimizes the cost-to-go $V_k(\mathbf{x}_k) = \sum_{i=0}^{T} \gamma^i c_{k+i+1}$ with $\gamma \in [0, 1]$. This framework can be modeled as a Markov decision process (MDP).

A MDP is a tuple $\mathcal{M} = \langle \mathcal{X}, \mathcal{U}, \mathcal{T}, \mathcal{J}, \gamma \rangle$, where $\mathcal{X}$ is a finite state space, $\mathcal{U}$ is a finite action space, $\mathcal{T}$ is a transition model and $\mathcal{J}$ a cost function. The transition model holds the markov property, which states, that the next state $\mathbf{x}_{k+1}$ at time step $k+1$ only depends on state $\mathbf{x}_k$ and action $\mathbf{u}_k$ at the current time step k.

The MDP can be solved using the Bellman equation:

$$V(\mathbf{x}_k) = c_k + \gamma V(\mathbf{x}_{k+1}), \quad \gamma \in [0, 1]. \tag{9}$$

2.2.2. Q-Learning. To augment the concept of value functions, an advantage function $A(\mathbf{x}, \mathbf{u}) : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ can be introduced that holds information about the value of taking an action in a certain state. Adding A to V results in the action-value function Q:

$$Q(\mathbf{x}, \mathbf{u}) = V(\mathbf{x}) + A(\mathbf{x}, \mathbf{u}) : \mathcal{X} \times \mathcal{U} \to \mathbb{R}, \tag{10a}$$

which returns the value of a given state-action pair $(\mathbf{x}, \mathbf{u})$:

$$Q(\mathbf{x}, \mathbf{u}) = \mathbb{E}[V(\mathbf{x}) | \mathbf{x} = \mathbf{x}_k, \mathbf{u} = \mathbf{u}_k,]. \tag{10b}$$

The greedy policy

$$\pi(\mathbf{x}) = \arg\min_{\mathbf{u}^*} Q(\mathbf{x}, \mathbf{u}^*), \tag{11}$$

leads to the optimal action $\mathbf{u}^*$, which minimizes Q. If the agent knows the optimal action-value function Q^*, the MDP can be solved by using the greedy policy.

With out knowing the transition probabilies $\mathcal{T}$ of the environment, Q^* has to be estimated from data. In Q-learning the agent tries to approximate Q^* by interacting with the environment. Therefore so called Q-updates are taken by the agent:

$$\begin{aligned} Q(\mathbf{x}_k, \mathbf{u}_k) &:= (1-\alpha) Q(\mathbf{x}_k, \mathbf{u}_k) \\ &\quad + \alpha [c_k + \gamma \underbrace{\min_{\mathbf{u}} Q(\mathbf{x}_{k+1}, \mathbf{u})}_{V(\mathbf{x}_{k+1})}], \quad \alpha, \gamma \in [0,1]. \end{aligned} \tag{12}$$

The learning rate α determines how the new and old Q-value are blended. In a stochastic environment the value should be set lower, because different states $\mathbf{x}_{k+1}$ can occur for a tuple $(\mathbf{x}_k, \mathbf{u}_k)$. In a deterministic environment $\alpha \equiv 1$, therefore (12) simplifies to (9):

$$Q(\mathbf{x}_k, \mathbf{u}_k) := c_k + \gamma \min_{\mathbf{u}} Q(\mathbf{x}_{k+1}, \mathbf{u}), \tag{13}$$

$$:= c_k + \gamma V(\mathbf{x}_{k+1}). \tag{14}$$

This value-iteration converges to Q^* in a deterministic setting and finite state space $\mathcal{X}$, if the equation is applied to all $\mathbf{x} \in \mathcal{X}$ simultaniously.

In tabular Q-learning [5] a value in a table is assigned to every state-action pair. To solve the MDP, the Q-updates have to be executed until convergence for every state-action pair. This leads to limitations in applicability of the algorithm to large state and action spaces. One way out of this dilemma is approximation of Q, e.g. using a neural network [6], as described in 3.3 or other function approximators, like polynomial functions [7, p. 30] tile-coding [8] or gaussian processes [9].

2.3. ARTIFICIAL NEURAL NETWORKS (ANN)

Through the application of ANNs huge progress has been made in image- and natural language processing [10], as well as in robotics [11]. Some of these progresses attracted attention outside the scientific communitiy. This is especially true for applications of deep neural networks in reinforcement learning. [1],[12]

2.3.1. Multilayer perceptron (MLP). A certain type of ANN is the MLP (Figure 3), which is composed of simple computation units, the perceptrons (Figure 2). A perceptron can be viewed as a strongly simplified model of a neuron. A perceptron Figure 2 has an input vector $\mathbf{x} = (x_1, x_2, ..., x_n)^{\mathrm{T}}$, a weight vector $\mathbf{w}_j = (w_{1,j}, w_{2,j}, ...w_{n,j})^{\mathrm{T}}$, a bias term b, as well an activation function $f_{\mathrm{act.}}$ (e.g. $\tanh$). The output y_i is computed by:

$$y_j = f_{\mathrm{act.}}(\mathbf{w}_j^{\mathrm{T}} \mathbf{x} + b)$$

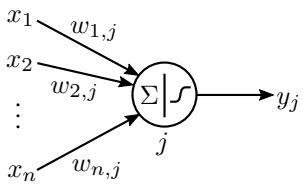

Figure 2. Perceptron

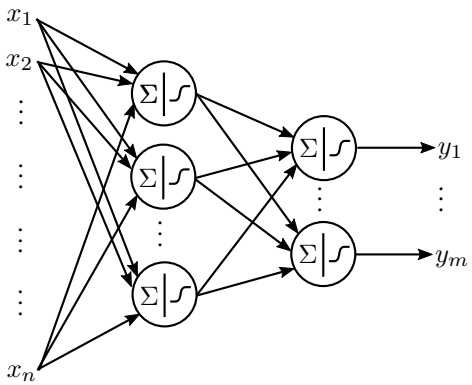

Figure 3. Multilayer perceptron

2.3.2. MLPs as function approximators. Three layer MLPs are universal function approximators and can approximate arbitrary nonlinear continuous multidimensional functions with arbitrary accuracy. This property has been proven for MLPs with linear input and output layers and certain activation functions in the hidden layer. [13], [14] In practice an appropriate count of perceptrons and layers is needed as well as algorithms to tweek the weights of the model. These algorithms are in general based on backpropagation, a gradient-based algorithm that updates the weights to minimize an error function. [15]

There exist powerful *Python*-packages for machine learning with MLPs. Tasks of function approximation can be done quite easy using the API *Keras* [16], that is a part of *Tensorflow* [17]. This requires an appropriate data set for training.

3. STUDIED METHODS

3.1. APPROXIMATION OF OPTIMAL STATE FEEDBACK (optSF)

This section deals with the optimization-based design of nonlinear static MLP state feedbacks for stabilization of nonliear dynamical systems in an unstable equlibrium. Therefore a limited set of optimal trajectories with the method described in subsubsection 2.1.1 is computed. These trajectories are than used to approximate an optimal state feedback using supervised learning. This approach can be viewed as imitation learning. A continuous function $\mathbf{K}_{\mathrm{MLP}} : \mathbf{x} \mapsto \mathbf{u}^*$ that maps states to optimal controls should be approximated. The nonlinear dynamics are given and there are no disturbances. The cost function and the system at hand are considerd to be of a kind such that the necessary conditions of a optimality (3) are sufficient and $\mathbf{K}_{\mathrm{MLP}}$ exists.

3.1.1. Solution of the BVP. The *Python*-package *SciPy* [18] contains the function *integrate.solve_bvp* that implements a solver for the BVP in subsubsection 2.1.1. After the computation of the optimal trajectory the fullfilment of the conditions of optimality as well as boundary conditions are checked.

3.1.2. Training data set. For the problem at hand, optimal trajectories of state and control from different initial conditions are used as a training data set. Therefore, we use a sufficent count of N samples from $\mathbf{X}_0 \sim \mathcal{U}(\mathbf{x}_{0\,\min}, \mathbf{x}_{0\,\max})$ and $\mathbf{x}_{0\,\min}, \mathbf{x}_{0\,\max} \in \mathbb{R}^n$. $\mathcal{A} = [\mathbf{x}_{0\,\min}, \mathbf{x}_{0\,\max}]$ is the set of all admissible deviations. For growing $\mathcal{A}$ and state dimensions, N has to grow even faster to have enough samples for a good approximation.

3.2. LQR

For stabilizing nonlinear dynamical systems in an unstable equilibrium for the application of the LQR design a linearization of the system at the equilibrim is needed. Through the weight matrices the closed loop behaviour can be influenced in terms of state deviation from the equlibrium and control effort. To compute the optimal feedback matrix the solution of the ARE is needed. An efficent solver is *linalg.solve_continuous_are* from the *Python*-package *SciPy*. The computation of a feedforward filter is not neccessary.

The controller performance should decrease for increasing deviations from the equlibirum. Possible state and control constraints are not taken into account explicitly in the design of the controller.

3.3. NEURAL FITTED Q-ITERATION (NFQ)

An algorithm that tackles the problems of Q-learing subsubsection 2.2.2 is NFQ [6]. The algorithm uses an MLP as a function approxiamtor of the action-value function Q. It uses iterative updates of this Q-network to obtain an approximate solution of Q^*. The algorithm was chosen for our example system, because the authors in [6] and in [19] showed, that NFQ can be applied to systems like the inverted pendulum. The learned controllers showed good performance.

3.3.1. Learning process. The learning process is composed of episodes. In every epsiode the state of the environment is set to $\mathbf{x}_0 \in \mathcal{X}_0 \subseteq \mathcal{X}$. The agent can now interact with the environment for T steps and obtains incremental costs$c_j, j \in [0,T]$. For every transition j to $j+1$ a tuple (u_j, x_j, x_{j+1}, c_j) is stored in a data set $\mathcal{D}$. This data set is the memory of the agent.

An episdoe is terminated, if $k = T$ or a terminal state $\mathbf{x}^- \in \mathcal{X}^- \subseteq \mathcal{X}$ is reached. If a state is terminal is determined by the cost function:

$$c_j(\mathbf{x}_j, \mathbf{u}_j) = \begin{cases} 0 & \text{for } \mathbf{x}_j \in \mathcal{X}^+, \\ 1 & \text{for } \mathbf{x}_j \in \mathcal{X}^-, \\ 0.01 & \text{else.} \end{cases} \tag{15}$$

This cost function also defines the goal states $\mathbf{x}^+ \in \mathcal{X}^+ \subseteq \mathcal{X}$ that the agent has to reach as fast as possible to minimize the cost of the episode. The decision to constrain the output of the cost function to $[0,1]$ arises from the activation function of the ANN, which is a sigmoid function. It can only output values in this interval.

3.3.2. Training process of the MLP. At the end of an episode out of every transition $(u_j, x_j, x_{j+1}, c_j) \in \mathcal{D}$ an input and output pair for training of the Q-network is generated. The input of the network is $(\mathbf{x}_j, \mathbf{u}_j)$. Using (12) the forward pass of the Q-network leads to the desired output y_i for a given input:

$$y_j = \begin{cases} 1 & \text{for } \mathbf{x}_{j+1} \in \mathcal{X}^-, \\ c_j + \gamma \min\limits_{\mathbf{u}} \hat{Q}_n(\mathbf{x}_{j+1}, \mathbf{u}; \theta) & \text{else.} \end{cases} \tag{16}$$

The generated data set is used for training. Learning can be accelerated by introducing artificial state transitions that force the output of the network in the target state $\mathcal{X}^+$ to 0. The generalization of the network speeds up the convergence of the algorithm. [20, p. 10] The computational effort of computing

initialize data set $\mathcal{D}$;
initialize MLP $\hat{Q}(\mathbf{x}, \mathbf{u}; \theta_0) \leftarrow \theta_0 \sim \mathcal{U}(-0.5, 0.5)$;
for *n episodes* **do**
  $\mathbf{x}_k := \mathbf{x}_0 \in \mathcal{X}_0$;
  for $k = 0, T$ **do**
    agent takes action:
    $\mathbf{u}_k = \pi(\mathbf{x}_k)$ - (ϵ-greedy);
    simulation of environment:
    $\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k)$;
    computation of costs:
    $c_k(\mathbf{x}_k, \mathbf{u}_k)$;
    store transition:
    $(\mathbf{u}_k, \mathbf{x}_k, \mathbf{x}_{k+1}, c_k)$ in $\mathcal{D}$;
    compute training data:
    $y_j =$ $\begin{cases} 1 & \text{for } \mathbf{x}_{j+1} \in \mathcal{X}^-, \\ c_j + \gamma \min\limits_{\mathbf{u}} \hat{Q}(\mathbf{x}_{j+1}, \mathbf{u}; \theta_n) & \text{else.} \end{cases}$;
    $\mathbf{x}_k := \mathbf{x}_{k+1}$;
  end
  training of the MLP:
  $\theta_{n+1} := argmin_\theta \sum_j (y_j - \hat{Q}(\mathbf{x}_j, \mathbf{u}_j; \theta_n))^2$;
end

Algorithm 1: NFQ

$\mathbf{u}$ through

$$\min_{\mathbf{u}_k} \hat{Q}_n(\mathbf{x}_{j+1}, \mathbf{u}_k, \theta)$$

increases with the number of actions. In [12] the Q-network has an output for every possible action. Another approach is optimization of the Q-network [21, p.84] which leads to a continous action version of NFQ.

4. COMPARISON

The three proposed methods are now applied on the example system. We compare them in terms of controller performance, starting from different initial conditions.

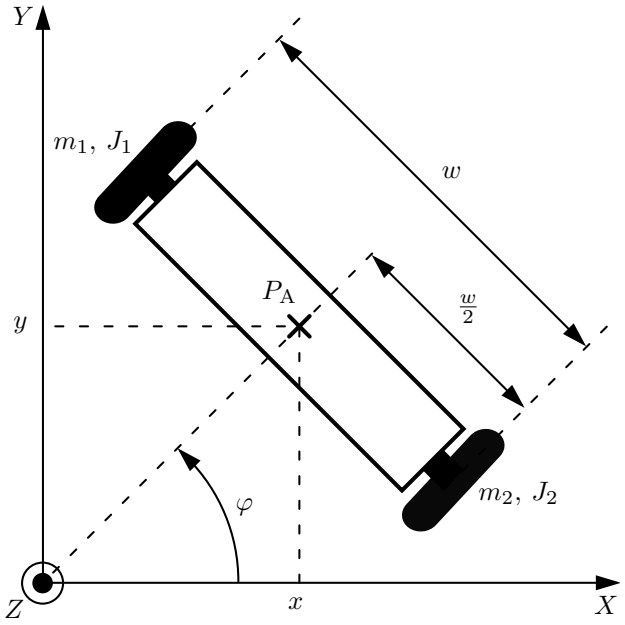

Figure 4. Example system (top view)

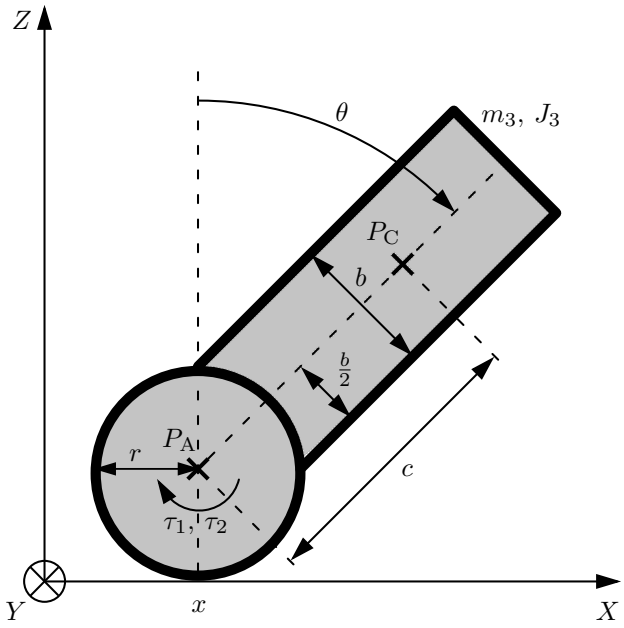

Figure 5. Example system (side view)

4.1. EXAMPLE SYSTEM

We consider a rigid body system with two wheels an a pendulum (Figure 4 and 5). The system variables are the deviation of the axle center P_A to the origin $d = \sqrt{x^2 + y^2}$, rotation angle around the vertical axis φ, angle of the pendulum θ and the induced wheel torques $\tau_1,\ \tau_2$. The system parameters are the masses of the rigid bodies $m_1 = m_2 =: m_w,\ m_3 =: m_p$, moments of inertia in terms of center of mass of the rigid bodies $J_1 = J_2 =: J_w,\ J_3 =: J_p$ with center of mass of the pendelum P_C, gravity g and geometric variables $r_1 = r_2 =: r,\ b,\ c$ and w. To limit the complexity of the system, the control variables are set to $u := \tau := \tau_1 = \tau_2$. This leads to $\varphi = 0$ and $d = x$ and a reduction of the state and control dimensions.

The state is $\mathbf{x} := (x_1, x_2, x_3, x_4)^T := (x, \theta, \dot{x}, \dot{\theta})^T$. The equations of motion are derived form the potential and kinetic energy of the system using the Lagrange-formalism. From those equations a nonlinear state space model can be obtained:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, u) = \begin{pmatrix} \begin{pmatrix} x_3 \\ x_4 \end{pmatrix} \\ \mathbf{M}^{-1}(x_2) \cdot (\mathbf{F}(u) - \mathbf{C}(x_2, x_4) - \mathbf{K}(x_2)) \end{pmatrix}.$$

The system is underactuated and has an unstable equilibrium at $\mathbf{x}_{RL} = (x_{RL},\ 0,\ 0,\ 0)^T$ with $x_{RL} \in \mathbb{R}$. The goal state is set to the equilibrium with $x_{RL} = 0$.

Remark that the nonholonomic constraint $|\theta| < \frac{\pi}{2}$ is neglected since it can be explicitly treated in the controller design as a state constraint. The control is constraint by a torque of 5 Nm.

4.2. SIMULATIONEXPERIMENTS

To evaluate the controller performances of the described methods, the stabilization from 7 different initial conditions of the robot is compared based on the cost (1). With $l(\mathbf{x}, \mathbf{u}) = \mathbf{x}^T\mathbf{Q}\mathbf{x} + Ru^2$ and $k(\mathbf{x}_f) = \mathbf{x}_f^T\mathbf{S}\mathbf{x}_f$. This was used for the design of an optimal state feedback (subsection 3.1) and LQR. For the NFQ-design the discrete cost function (15) was used. The learning curves in Figure 6 show, that the terminal state

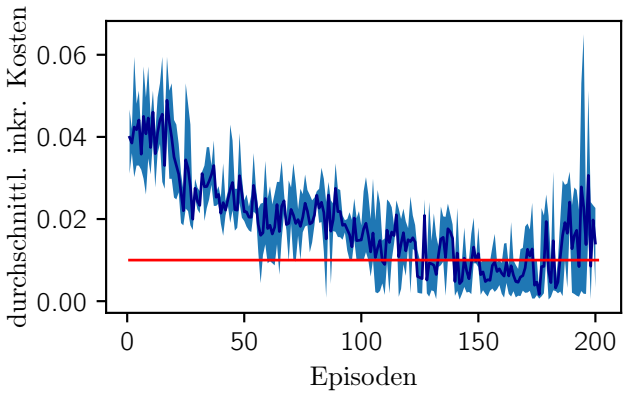

Figure 6. Learning curves of 200 episodes (blue = average over all learning processses, red = above: episode terminated)

$\mathcal{X}^-$ is not reached anymore after around 50 episodes. After 150 episodes the agent drives the system to the desired state $\mathcal{X}^+$ with high probability. But the algorithm can also diverge, which can be seen at the end of the learning curve.

4.3. RESULTS

Table 1 shows the results of the simulation experiments. The initial conditions of $\dot{x}_0$ and $\dot{\theta}_0$ were always set to 0. The lowest total cost is marked bold. Figure 7 shows the control and state trajectories for LQR, optZRF and NFQ for deviation 2. In Figure 8 the control and state trajectories for LQR and optZRF for deviation 6 are shown. All methods can stabilize

Table 1. Total cost of for different initial conditions

#	x_0/m	θ_0/rad	J_{LQR}	J_{optZRF}	J_{NFQ}
1	0	$\frac{\pi}{9}$	1,047	**1,046**	106,243
2	0	$\frac{2\pi}{9}$	7,329	**6,931**	79,720
3	0,3	0	**0,578**	0,579	39,310
4	0,7	0	**3,122**	**3,122**	44.684
5	0,5	$\frac{\pi}{6}$	7,644	**7,499**	70,778
6	0,5	$\frac{\pi}{4}$	30,916	**17,384**	-
7	0	$\frac{\pi}{3}$	-	**44,886**	-

the system in the unstable equilibrium in initial conditions 1 to 5.

In terms of total cost, optSF showed the best controller performance, with an exception for small deviations in x. For small deviations of x the difference between LQR and optSF is very small. For deviations of the angle of the pendulum, it shows a much better performance than LQR. This can be seen Figure 8. For inital condition 7 onlyoptSF leads to stabilization.

The suboptimality of LQR was expected due to the linearization of the system dynamics. THe high performance of optSF is interesting, considering how many optimal trajectories where computed. It is also remarkable that initial condition 7 was not in the training data. The high cost of the NFQ-algorithm (Table 1) comes from the different cost function of the agent and the discrete action space. But it should not be mentioned that this algorithm does not explicitly use the robots dynamics model.

5. CONCLUSION AND OUTLOOK

The implementation of NFQ and optSF is straight forward and is simplified through the availabilty of software packages like *Keras* and *TensorFlow*. In optSF the computation of the optimal control trajectories is more computatinally expensive than the supervised laerning part. Working with NFQ it was not trivial to choose appropriate actions for the agent, which makes it hard to compare to the other two methods. Another degree of freedom in the design are the hyperparameters (section 5) of the algorithm which are not simple to tune. [19] was a huge help.

The expactation, that NFQ can easily be applied to different systems without much tuning effort was not confirmed.

An algorithm, that deals with the problems of NFQ is Deep-Deterministic Policy-Grandient (DDPG) [22], where a second policy network is used. It can be viewed as an extension to NFQ and was tested by the authors on different system, e.g. cart-pole. Another advantage besides the continuous action is that the algorithm is more general.

The proposed method for the design of optimal state feedbacks has proven to be a powerful approach for the example system. Compared to the LQR-design, the implementation complexity is enormous and only useful if a system should be stabilized for large deviations.

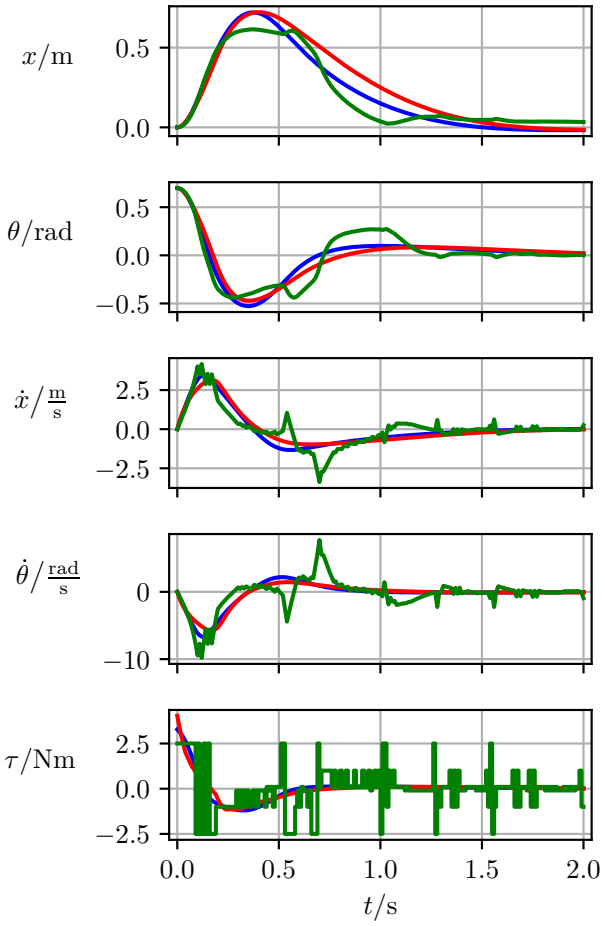

Figure 7. State and control trajectories (LQR, NFQ, optSF) für Auslenkung 2 (Table 1)

It is an open research question if the method can be used on more complex system in terms of state and control dimensions, as well as nonlinearities. This is true from a theoretic persepective with a view on conditions of optimality as well as from the application side concerning the BVP solver.

In the calculation of optSF control constraints could be treated explicitly. The adapted optimality condition is know as PONTRJAGINs maximum principle [2, p.110ff.]. Furthermore, it could be studied how the mezhod could be used to design a MLP-feedforward-filter.

The described methods show, that methods from machine learning can be used in control theory and developments in this field should be observed.

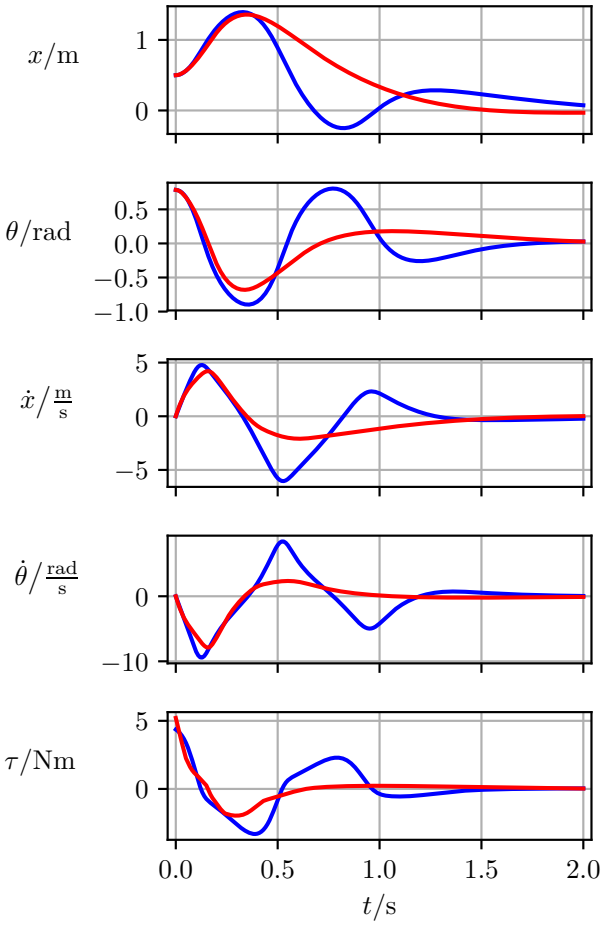

Figure 8. State and control trajectories (LQR , optSF) for deviation 6 (Table 1)

REFERENCES

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge", *Nature*, vol. 550, no. 7676, p. 354, 2017. DOI: 10.1038/nature24270.

[2] K. Graichen, *Methoden der optimierung und optimalen steuerung, Wintersemester 2017/2018*, Institut für Mess–, Regel– und Mikrotechnik, Uni Ulm, 2017. [Online]. Available: https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.110/Downloads/Vorlesung/Optimierung/Skript/Skript_MOOS_WS1718.pdf.

[3] R. S. Sutton and A. B. Barto, *Reinforcement Learning: An Introduction*. The MIT Press Cambridge, Massachusetts, 1998.

[4] R. S. Sutton, A. G. Barto, and R. J. Williams, "Reinforcement learning is direct adaptive optimal control", *IEEE Control Systems*, vol. 12, no. 2, pp. 19–22, 1992. DOI: 10.1109/37.126844.

[5] C. J. Watkins and P. Dayan, "Q-learning", *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992. DOI: 10.1007/BF00992698.

[6] M. Riedmiller, "Neural fitted q-iteration - first experiences with a data efficient reinforcement learning method", in *Machine Learning: ECML 2005. ECML 2005. Lecture Notes in Computer Science*, vol. 3720, 2005. DOI: 10.1007/11564096_32.

[7] C. Szepesvári, "Algorithms for reinforcement learning", *Synthesis lectures on artificial intelligence and machine learning*, vol. 4, no. 1, pp. 1–103, 2010. DOI: 10.2200/S00268ED1V01Y201005AIM009.

[8] C. G. Atkeson and J. C. Santamaria, "A comparison of direct and model-based reinforcement learning", in *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, IEEE, vol. 4, 1997, pp. 3557–3564. DOI: 10.1109/ROBOT.1997.606886.

[9] S. Kamthe and M. P. Deisenroth, "Data-efficient reinforcement learning with probabilistic model predictive control", 2017. arXiv: 1706.06491.

[10] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio", *arXiv preprint arXiv:1609.03499*, 2016.

[11] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey", *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013. DOI: 10.1177/0278364913495721.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning", *Nature*, vol. 518, no. 7540, p. 529, 2015. DOI: 10.1038/nature14236.

[13] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators", *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989. DOI: 10.1016/0893-6080(89)90020-8.

[14] S. Sonoda and N. Murata, "Neural network with unbounded activation functions is universal approximator", *Applied and Computational Harmonic Analysis*, vol. 43, no. 2, pp. 233–268, 2017. DOI: 10.1016/j.acha.2015.12.005.

[15] S. Ruder, "An overview of gradient descent optimization algorithms", 2016. arXiv: 1609.04747.

[16] F. Chollet *et al.*, *Keras*, https://keras.io, 2015.

[17] *TensorFlow*, 2015. [Online]. Available: https://www.tensorflow.org/.

[18] E. Jones, T. Oliphant, P. Peterson, *et al.*, *SciPy: Open source scientific tools for Python*, 2001–. [Online]. Available: http://www.scipy.org/.

[19] M. Riedmiller, "10 steps and some tricks to set up neural reinforcement controllers", in *Neural networks: tricks of the trade*, Springer, 2012, pp. 735–757.

[20] ——, "A framework for rl learning controllers", 2008. [Online]. Available: http://people.csail.mit.edu/russt/iros2008_workshop_talks/Riedmiller.pdf.

[21] R. Hafner, *Dateneffiziente selbstlernende neuronale regler*, 2009. [Online]. Available: https://d-nb.info/998931659/34.

[22] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning", 2015. arXiv: 1509.02971.

APPENDIX

EXAMPLE SYSTEM

Derivation of the nonlinear state space model.
Generalized coordinates:

$$\mathbf{q} = (q_1, q_2)^{\mathrm{T}} = (x, \theta)^{\mathrm{T}}$$

Kinetic energy T and potential energy U:

$$T(\mathbf{q}, \dot{\mathbf{q}}) = \frac{1}{2}\left(2m_{\mathrm{w}} + 2\frac{J_{\mathrm{w}}}{r^2} + m_{\mathrm{p}}\right)\dot{x}^2 + m_{\mathrm{p}} c\cos(\theta)\dot{\theta}\dot{x} + \frac{1}{2}\left(J_{\mathrm{p}} + m_{\mathrm{p}} c^2\right)\dot{\theta}^2$$

$$U(\mathbf{q}, \dot{\mathbf{q}}) = m_{\mathrm{p}} g c\cos(\theta)$$

LAGRANGE-function:

$$L(\mathbf{q}, \dot{\mathbf{q}}) = T(\mathbf{q}, \dot{\mathbf{q}}) - U(\mathbf{q}, \dot{\mathbf{q}})$$

Equations of motion:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L(\mathbf{q}, \dot{\mathbf{q}})}{\partial \dot{q}_j}\right) - \frac{\partial L(\mathbf{q}, \dot{\mathbf{q}})}{\partial q_j} = F_j,\ j = 1, 2$$

$$F_1 = 2\frac{\tau}{r},\ F_2 = 0$$

Equations of motion in matrix form:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{K}(\mathbf{q}) = \mathbf{F}$$

$$\mathbf{M}(\mathbf{q}) = \begin{pmatrix} 2m_{\mathrm{w}} + 2\frac{J_{\mathrm{w}}}{r^2} + m_{\mathrm{p}} & m_{\mathrm{p}} c\cos(\theta) \\ m_{\mathrm{p}} c\cos(\theta) & J_{\mathrm{p}} + m_{\mathrm{p}} c^2 \end{pmatrix},\ \mathbf{F} = \begin{pmatrix} 2\frac{\tau}{r} \\ 0 \end{pmatrix}$$

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{pmatrix} -m_{\mathrm{p}} c\sin(\theta)\dot{\theta}^2 \\ 0 \end{pmatrix},\ \mathbf{K}(\mathbf{q}) = \begin{pmatrix} 0 \\ -m_{\mathrm{p}} g c\sin(\theta) \end{pmatrix}$$

Nonlinear state space model (explicit):

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, u) = \begin{pmatrix} x_3 \\ x_4 \\ \frac{r}{2}\frac{c^2 m_{\mathrm{p}}^2 g r\sin(2x_2) - 2(J_{\mathrm{p}} + c^2 m_{\mathrm{p}})(c m_{\mathrm{p}} r x_4^2\sin(x_2) + 2u)}{c^2 m_{\mathrm{p}}^2 r^2\cos^2(x_2) - (J_{\mathrm{p}} + c^2 m_{\mathrm{p}})(2J_{\mathrm{w}} + r^2(m_{\mathrm{p}} + 2m_{\mathrm{w}}))} \\ c m_{\mathrm{p}}\frac{r(c m_{\mathrm{p}} r x_4^2\sin(x_2) + 2u)\cos(x_2) - g(2J_{\mathrm{w}} + r^2(m_{\mathrm{p}} + 2m_{\mathrm{w}}))\sin(x_2)}{c^2 m_{\mathrm{p}}^2 r^2\cos^2(x_2) - (J_{\mathrm{p}} + c^2 m_{\mathrm{p}})(2J_{\mathrm{w}} + r^2(m_{\mathrm{p}} + 2m_{\mathrm{w}}))} \end{pmatrix}$$

Model parameters.

$$m_{\mathrm{w}} = 0,5\ \mathrm{kg},\ m_{\mathrm{p}} = 3,0\ \mathrm{kg},\ g = 9,81\frac{\mathrm{m}}{\mathrm{s}^2},$$
$$b = 0,2\ \mathrm{m},\ c = 0,3\ \mathrm{m},\ r = 0,05\ \mathrm{m},$$
$$J_{\mathrm{w}} = \frac{1}{2} m_{\mathrm{w}} r^2,\ J_{\mathrm{p}} = \frac{1}{12} m_{\mathrm{p}}\left(b^2 + c^2\right)$$

IMPLEMENTATION

Weight matrices.

$$\mathbf{Q} = \mathrm{diag}\left(8,7\ \tfrac{1}{\mathrm{m}}\quad 8,7\ \tfrac{1}{\mathrm{rad}}\quad 10^{-5}\ \tfrac{\mathrm{s}}{\mathrm{m}}\quad 10^{-5}\ \tfrac{\mathrm{s}}{\mathrm{rad}}\right),$$
$$R = 4,8\ \frac{1}{\mathrm{Nm}},\ \mathbf{S} = \mathrm{diag}\left(8,7\ \tfrac{1}{\mathrm{m}}\quad 8,7\ \tfrac{1}{\mathrm{rad}}\quad 4\ \tfrac{\mathrm{s}}{\mathrm{m}}\quad 3\ \tfrac{\mathrm{s}}{\mathrm{rad}}\right)$$

Parameters of optSF (subsection 3.1).
Trainingsdatensatz:

$$N = 100,\ \mathbf{x}_{0\ \max} = \left(1\ \mathrm{m},\ \frac{\pi}{4}\ \mathrm{rad},\ 1\ \frac{\mathrm{m}}{\mathrm{s}},\ 1\frac{\mathrm{rad}}{\mathrm{s}}\right)^{\mathrm{T}},$$

$$\mathbf{x}_{0\ \min} = -\mathbf{x}_{0\ \max},\ t_{\mathrm{f}} = 3\ \mathrm{s},\ \text{Schrittweite: } 10\ \mathrm{ms}$$

MLP: input dimension: 4; output dimension: 1; 20 perceptrons in the input layer with rectified-linear-unit (ReLU) as activation function, no hidden layer, one perceptron in the linear output layer, 121 weights (with bias), *Keras*-loss: *mean_ squared_ error*, *Keras*-optimizer: *nadam*

MLP training: cycles: 30, batch: 32, validation data set: 10 % of the training data set, appproximation error after training: $< 10^{-3}$

Parameters of NFQ (subsection 3.3).
MLP: input dimension: $\dim(\mathbf{x}) + dim(\mathbf{u}) = 5$; output dimension: $dim(\hat{Q}) = 1$; the two hidden layers have 20 perceptrons each and tanh as activation function, in the output layer the sigmoid function is used. 561 wights in total, *TensorFlow* loss function: *mean_ squared_ error*, *TensorFlow* optimizer: *tf.train.AdamOptimizer()*

MLP training: cycles: 300, batch: ≤ 15.000, step size: $0,001$, weight init.: uniform distribution $[-0,5;\ 0,5]$

Table 2. Learning process parameters

episodes n	200
length T	6 s
Δt	10 ms
γ	0,99
ϵ	0,1
ϵ_{exp}	0,98
τ	$\left(-2,5 \quad -1,0 \quad -0,1 \quad 0,1 \quad 1,0 \quad 2,5\right)\mathrm{Nm}$
$\mathcal{X}^0$	$\left([-0,5,\quad 0,5]\mathrm{m} \quad [-0,3,\quad 0,3]\mathrm{rad} \quad * \quad *\right)$
$\mathcal{X}^+$	$\left([-0,05,\quad 0,05]\mathrm{m} \quad [-0,03,\quad 0,03]\mathrm{rad} \quad * \quad *\right)$
$\mathcal{X}^-$	$\left([-1,\quad 1]\mathrm{m} \quad [-\frac{\pi}{2},\quad \frac{\pi}{2}]\mathrm{rad} \quad * \quad *\right)$