Institute of Electrical Engineering and Information Technology
Paderborn University
Department of Power Electronics and Electrical Drives
Prof. Dr.-Ing. Joachim Böcker

Master Thesis

Combined Current Control and System Identification of a PMSM with Neural Ordinary Differential Equations

by

Marvin Meyer

Supervisor: Wilhelm Kirchgässner and Maximilian Schenke
Thesis Nr.: MA129
Filing Date: November 24, 2022

Declaration of Authorship

I declare that I have authored this thesis independently, that I have not used other than the declared sources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources. This paper was not previously presented to another examination board and has not been published.

Paderborn, 24.11.2022 

Date Signature

Abstract

This thesis deals with current control and system identification of a permanent magnet synchronous machine. To learn a data-driven controller for current control, a model of the motor is required. Such model can be constructed with the utilization of neural ordinary differential equations. Through this approach, expert knowledge can be used in combination with learnable parameters to identify the motor parameters. With the help of this model, a neural network can then be trained to control the current. The proposed approach will be compared with both model-based and model-free methods for current control.

Contents

1 Introduction

In recent years, the permanent magnet synchronous motor (PMSM) has become increasingly popular for industrial and automotive applications. The appeal results primarily from a broad spectrum of advantages: high efficiency, good control accuracy, high torque and power density, low rotor inertia and high life expectancy. This motor rotates synchronously the stator's rotating magnetic field and has permanent magnets instead of windings in the rotor, which maintain a constant magnetic field. The permanent magnets have to meet high requirements, especially with regard to the specific energy density, which results in a higher acquisition cost. An alternative design of the electric motors is certainly worthwhile from an ecological point of view, since the rare earths in particular have a poor carbon footprint. However, as raw material prices are expected to decrease in the coming years, and in combination with the beneficial properties of the PMSM, it is expected that the PMSM drive system will continue to gain popularity.

In order for the PMSM drive system to achieve the best possible steady-state and dynamical performance, various control strategies have been developed. One of the model based methods is the field-oriented control (FOC). The classical FOC is implemented by simple proportional-integral controllers (PI controllers), with the proportional and integration part adjusted to the plant. The FOC uses a mathematical model of the motor. However, modeling complex systems is not always possible without simplifications. In the case of the PMSM, i.e., these can be parasitic effects that occur in the plant but are not taken into account in the model.

Modern control approaches utilize the progress made in the field of machine learning (ML). A branch of this is reinforcement learning (RL), which has gained popularity in recent years due to many remarkable results [1][2]. These control algorithms use function approximators, such as artificial neural networks (ANNs), to develop a control strategy, referred to as policy. Instead of requiring a plant model for their design, they determine an appropriate policy based on interactions with the plant. Since a policy is learned on the basis of data, it can be better adapted to the given plant.

Both approaches are at the different edges of the spectrum, that measures the exploitation of available system information. The goal in this work is to use as much expert knowledge about the plant as possible, but also to take advantage of the adaptability of ML approaches.

In this thesis, a novel approach to current control using ML is presented utilizing the concept of neural ordinary differential equations (NODEs) [3]. With the help of this approach, models can be learned from data in a supervised manner. On the one hand, an internal plant model is learned, which is to approximate the behavior of the plant. Here, expert knowledge of the plant can be used to improve the internal plant model. On the other hand, based on the internal plant model, a policy is learned, which takes over the control of the plant. Both components, the internal plant model and the policy, are represented by ANN and can be learned simultaneously. This approach uses both expert knowledge of the plant and ML's function approximation capabilities to obtain the best possible controller.

The thesis is structured as follows. In chapter 2, the theoretical fundamentals are presented. This includes the considered drive system and its corresponding model. In addition to the classic methods of control, also data-driven methods are explained. In chapter 3, the NODE approach will be elaborated. The current control with the help of the NODE is specified, which is extended by making use of the NODEs' capability for system identification. Chapter 4 deals with the evaluation of the experiments. First, the setting of the experiments is explained in more detail. This is followed by the results and discussion of those experiments. Chapter 5 summarizes the work and provides an outlook.

2 Theoretical Fundamentals

This chapter deals with the basics of the work in order to provide a better understanding of the later topics. The first section deals with the basics of the drive system. The components and the modeling of such a system will be discussed in more detail. After that, concepts of the system identification are given. The basics of mechanical learning are then discussed, first covering neural networks (NNs) and followed by RL.

2.1 Electric Drive

A modern drive system consists of four essential components: A power supply, the inverter, as well as the motor which is connected to the load [4]. The power supply's task is to provide the inverter with a constant voltage on the input side. The inverter uses the control signals from the controller to ensure that a voltage is applied to the motor through which a target current is then obtained. The voltage can be adjusted variably by the inverter using modulation methods. For the vast majority of drive-related tasks, the electrical energy is converted into mechanical energy, the so-called motor operating mode. In the opposite case, the system is in generator mode. Fig. 2.1 shows a simple drive system.

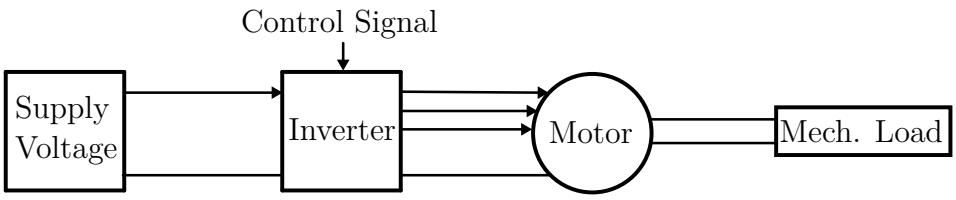


Fig. 2.1: Structure of the drive system.

Before this section is devoted to the modeling of the motor, the inverter and the type of motor used in the work will be introduced. Subsequently, a basic classical control method will be presented.

2.1.1 Inverter

The voltage-source inverter (VSI) converts the DC link voltage into an AC voltage [4]. This is solved by a three-phase bridge circuit composed of half-bridges for each phase, shown conceptually in Fig. 2.2.

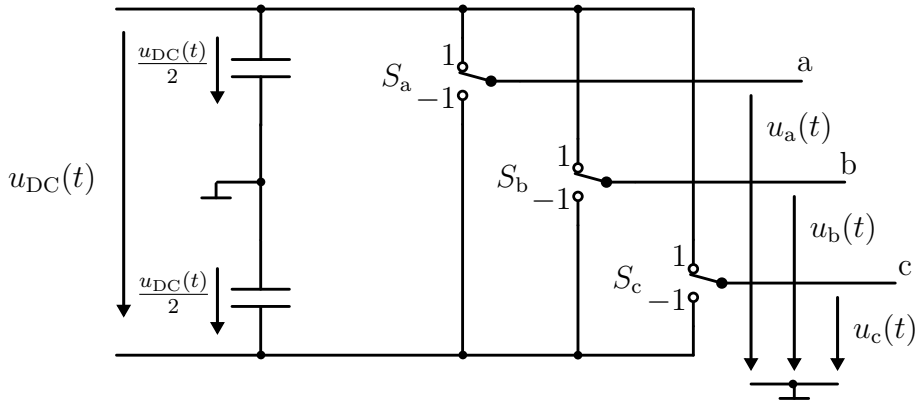


Fig. 2.2: Conceptual structure of a three-phase VSI.

The input voltage $u_{DC}(t)$ is applied across the two identically assumed capacitance. Between line a and ground the potential difference causes a voltage of $u_a(t)$. In dependence of the switch S_a the voltage can alternate between $\pm\frac{u_{DC}(t)}{2}$. Whether $u_a(t)$ is at the high or low potential is determined by the controller, which can change the switch position with the signal $s_a(t)$. Usually the switch is realized in hardware by paired metaloxide semiconductor field-effect transistors (MOSFETs) or insulated-gate bipolar transistors (IGBTs), as shown in Fig. 2.3. It depends on the application which of the two technologies are used.

Each phase $i \in \{a,b,c\}$ can be controlled by a separate signal $s_i(t)$ to generate the necessary phase shift of 120° between the voltages $u_i(t)$. In addition, the frequency can be controlled via the operating speed of the switches [5].

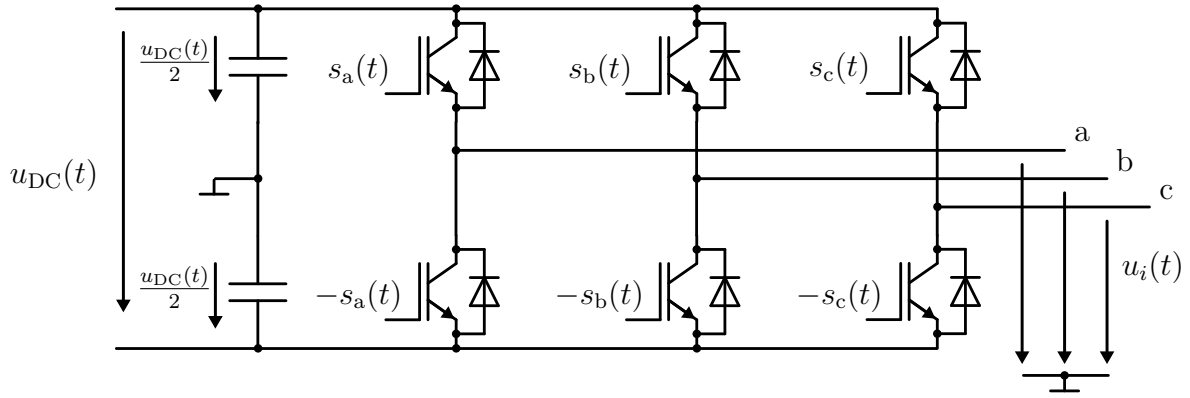


Fig. 2.3: Structure of the B6 bridge.

In Fig. 2.3 a three-phase VSI with IGBTs is shown. The voltages can be converted in vector notation to the following equation:

$$\begin{pmatrix} u_\mathrm{a}(t) \\ u_\mathrm{b}(t) \\ u_\mathrm{c}(t) \end{pmatrix} = \frac{u_\mathrm{DC}(t)}{2} \begin{pmatrix} s_\mathrm{a}(t) \\ s_\mathrm{b}(t) \\ s_\mathrm{c}(t) \end{pmatrix} \tag{2.1}$$

2.1.2 Pulse Centering

To calculate the input voltage of the motor, the physical voltage limits are first examined, which follow from the DC supply voltage and the inverter. The maximum voltage of the inverter is thus one half of the positive or negative supply voltage. Projecting this onto the abc-coordinates results in two stripes for each of the coordinates as boundaries. Due to the phase offset of 120°, the intersection points can be used to form a regular hexagon, as shown in Fig. 2.4. The maximum line-to-line voltage available at the motor therefore results from the input DC voltage. However, the modulation range can be extended by subtracting the zero component

$$u_0(t) = \frac{1}{2} \left(\max[u_\mathrm{a}^*(t), u_\mathrm{b}^*(t), u_\mathrm{c}^*(t)] + \min[u_\mathrm{a}^*(t), u_\mathrm{b}^*(t), u_\mathrm{c}^*(t)] \right) \tag{2.2}$$

so that the string voltages are increased:

$$\begin{aligned} u_\mathrm{a}^{**}(t) &= u_\mathrm{a}^*(t) - u_0(t) \\ u_\mathrm{b}^{**}(t) &= u_\mathrm{b}^*(t) - u_0(t) \\ u_\mathrm{c}^{**}(t) &= u_\mathrm{c}^*(t) - u_0(t). \end{aligned} \tag{2.3}$$

This gives the maximum possible modulation range for the line-to-line voltages

$$\frac{|u_\mathrm{ab,bc,ca}(t)|}{u_\mathrm{DC}(t)} \leq 1, \tag{2.4}$$

which is represented by the outer hexagon in Fig. 2.4.

2.1.3 Permanent Magnet Synchronous Machine

This thesis deals with PMSMs. These consists of two parts, the stator and the rotor, whose cross-section is illustrated in Fig. 2.5.

The non-moving part of the motor is called the stator. Using the three-phase AC coming from the inverter and the arrangement of the coils, the necessary rotating magnetic field is built up. The frequency of the voltage can thus be used to set the rotation speed of the rotor. Conversely, the rotor is the moving part of a PMSM. The necessary constant magnetic field is generated by permanent magnets, which are integrated in the rotor. The

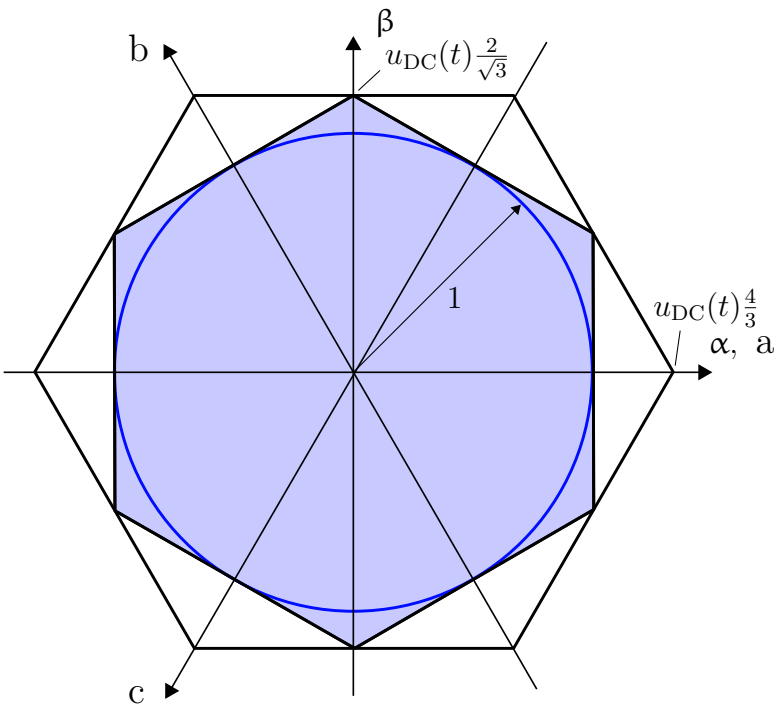


Fig. 2.4: Visualization of the voltage constraints forming two hexagons [6].

design determines how many pole pairs are involved. For the purpose of convenience, only one pole pair is shown in the figure. In steady-state, the rotor and stator flux vectors are synchronized and rotate with the same angular velocity [6].

2.1.4 Modeling

For machines with rotating fields, it is common to use two coordinate systems to simplify the calculation. The first one is the stator-fixed coordinate system. Opposed to this, the other coordinate system is rotor-fixed. Since the system has three phases $j \in \{\mathrm{a,b,c}\}$, the following transformations are applied to the voltage u_j, the current i_j, and the magnetic flux Ψ_j, respectively.

The αβ- and abc-coordinate systems share the same origin, where the α-axis overlaps with the a-axis and the β-axis is perpendicular to the α-axis (Compare Fig. 2.7). The transformation can be performed with the help of the following equations:

$$\begin{pmatrix} x_\alpha(t) \\ x_\beta(t) \end{pmatrix} = \boldsymbol{T}_{23} \begin{pmatrix} x_\mathrm{a}(t) \\ x_\mathrm{b}(t) \\ x_\mathrm{c}(t) \end{pmatrix} \tag{2.5}$$

where

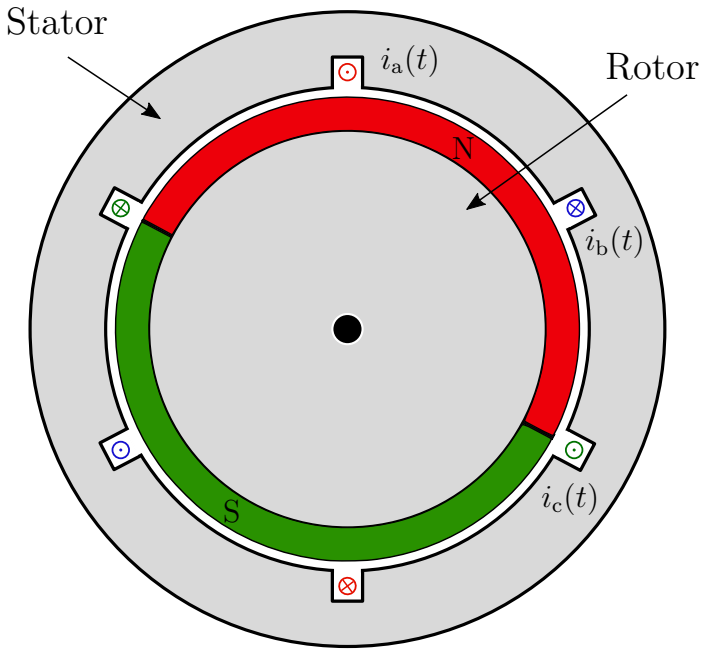
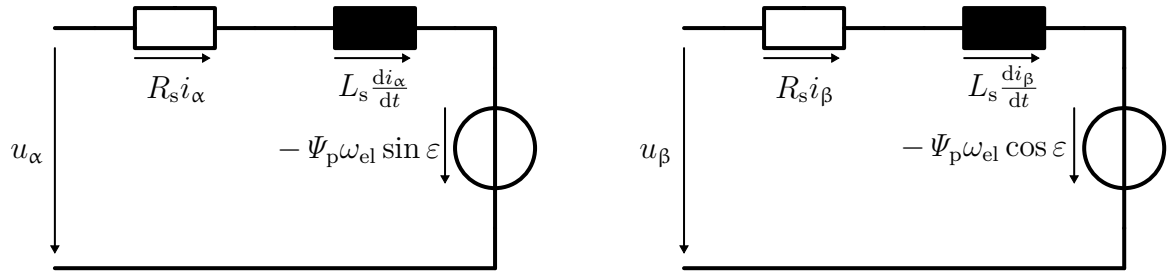


Fig. 2.5: Cross section of a PMSM with only one pole pair. This is a simplification to better distinguish the individual elements. Usually PMSMs contain several pool pairs.

$$\boldsymbol{T}_{23} = \begin{pmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ 0 & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \end{pmatrix}. \tag{2.6}$$

Applying this transformation to the system results in the equivalent circuit diagrams of the two components shown in Fig. 2.6.



(a) α-axis **(b)** β-axis

Fig. 2.6: The equivalent circuit diagrams for the PMSM in αβ-coordinates.

R_s is the winding resistance and L_s the inductance of the winding system. Generally, the inductances in the two lines are different. For simplicity, the inductances are assumed

to be equal. Different inductances will be taken into account later when the rotor-fixed coordinate system is introduced. The induced voltage of the permanent magnet results from the flux Ψ_p and the sine or cosine of the angle ε between rotor and stator, as well as the derivative of the angle, the angular velocity ω_el. This results in the two differential equations (DEs):

$$\begin{aligned} u_\alpha(t) &= R_\mathrm{s} i_\alpha(t) + L_\mathrm{s} \frac{\mathrm{d}i_\alpha(t)}{\mathrm{d}t} - \Psi_p \omega_\mathrm{el} \sin\varepsilon \\ u_\beta(t) &= R_\mathrm{s} i_\beta(t) + L_\mathrm{s} \frac{\mathrm{d}i_\beta(t)}{\mathrm{d}t} - \Psi_p \omega_\mathrm{el} \cos\varepsilon \end{aligned} \tag{2.7}$$

The voltages $u_\alpha(t)$ and $u_\beta(t)$, as well as the currents $i_\alpha(t)$ and $i_\beta(t)$ result from the previously discussed transformation. In order to get rid of the dependence on ε, i.e., the rotor position in relation to the stator, the second coordinate system is introduced.

The dq-coordinate system is fixed to the rotor. Therefore, the d-axis is placed across the pole pair and the q-axis again orthogonal to it. Since the rotor flux vector follows the field of the stator, there is a direct connection between the coordinate systems, whereby the angle ε indicates the twisting of the systems.

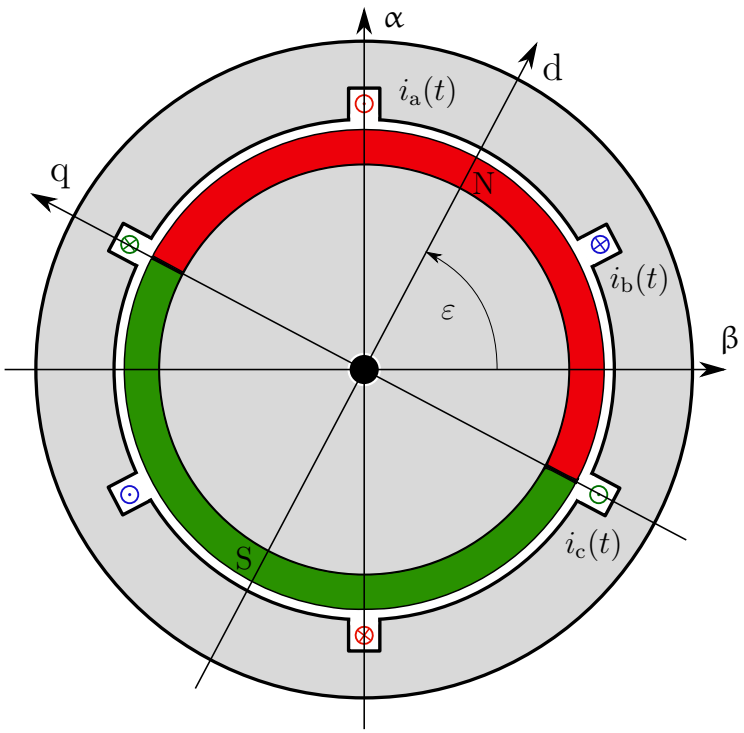


Fig. 2.7: Illustration of the different coordinate systems within the PMSM.

The projection between the systems is described by the Park-Transformation with the rotation matrix $\boldsymbol{Q}(\varepsilon)$:

$$\boldsymbol{Q}(\varepsilon) = \begin{pmatrix} \cos\varepsilon & -\sin\varepsilon \\ \sin\varepsilon & \cos\varepsilon \end{pmatrix} \tag{2.8}$$

Hence, the following relationship can be stated:

$$\begin{pmatrix} x_\alpha(t) \\ x_\beta(t) \end{pmatrix} = \boldsymbol{Q}(\varepsilon) \begin{pmatrix} x_\mathrm{d}(t) \\ x_\mathrm{q}(t) \end{pmatrix} \tag{2.9}$$

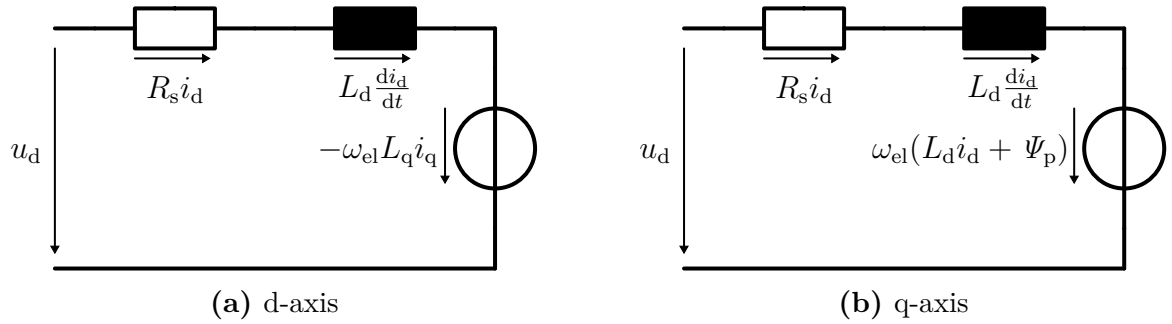
Applying this transformation to the Eq. (2.7) yields the ordinary differential equations (ODEs) in dq-coordinates:

$$\begin{aligned} u_\mathrm{d}(t) &= R_\mathrm{s} i_\mathrm{d}(t) + L_\mathrm{d} \frac{\mathrm{d}i_\mathrm{d}(t)}{\mathrm{d}t} - \omega_\mathrm{el} L_\mathrm{q} i_\mathrm{q} \\ u_\mathrm{q}(t) &= R_\mathrm{s} i_\mathrm{q}(t) + L_\mathrm{q} \frac{\mathrm{d}i_\mathrm{q}(t)}{\mathrm{d}t} + \omega_\mathrm{el} \left(L_\mathrm{d} i_\mathrm{d} + \Psi_\mathrm{p} \right) \\ \frac{\mathrm{d}\varepsilon(t)}{\mathrm{d}t} &= \omega_\mathrm{el} \end{aligned} \tag{2.10}$$

The torque results to

$$T = \frac{3}{2} p \left(\Psi_\mathrm{p} + \left(L_\mathrm{d} - L_\mathrm{q} \right) i_\mathrm{d} \right) i_\mathrm{q} \tag{2.11}$$

and is therefore dependent on both currents, but no longer on the angle. The scaling factor $\frac{3}{2}$ takes into account the conversion from a three- to a two-phase motor model. In Fig. 2.8 the equivalent circuit is shown.



(a) d-axis **(b)** q-axis

Fig. 2.8: The equivalent circuit diagrams for the PMSM in dq-coordinates.

2.1.5 Field-Oriented Control with PI-Controller

The FOC is one of the widely used classical control methods [7][8][9]. As described above, three-phase sinusoidal voltages are applied to the stator terminals. The aim of the FOC is to calculate the required voltages in field-oriented quantities, i.e., in the dq-coordinates, and then to convert these back to the stator-fixed coordinates in order to apply them to the motor. The advantage is that the previously sinusoidal quantities in the abc-coordinate system become constant quantities in the dq-coordinate system and thus a simpler control policy can be established as shown in Eq. (2.11).

Given a desired signal, the reference $r(t)$ and a system output $y(t)$, a control error $e(t)$ can be determined by subtracting the two values. The control of the plant is performed via linear PI controller. Based on the control error and the control policy, given by the PIC, a control variable $u(t)$ can be determined. A simple control loop is shown in Fig. 2.9.

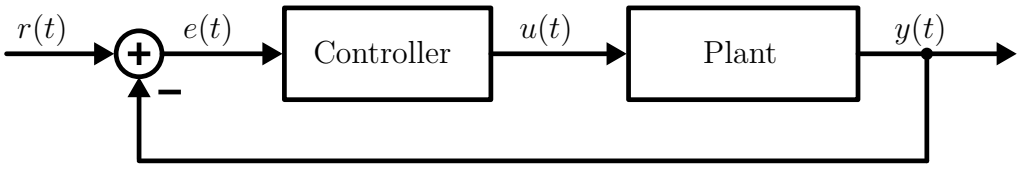


Fig. 2.9: Standard control loop.

Like the name indicates, the PI controller consists of two parts: the proportional and the integral component. The proportional part ensures a fast control when the error increases, whereas the integral part ensures that the steady state error is reduced. The control variable $u(t)$ is calculated as

$$u(t) = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(\tau)\mathrm{d}\tau = K_\mathrm{p} \left(e(t) + \frac{1}{T_\mathrm{N}} \int_0^t e(\tau)\mathrm{d}\tau \right). \tag{2.12}$$

The factors K_p and K_i represent the proportional and integral parts of the controller. K_i is the quotient of the proportionality factor and integration time T_N, which must be designed by the user. In order to apply two independent PI controller for the d and q components of the current, additional calculations are required. Figure Fig. 2.10 shows the controller structure for the FOC with PI controller.

To decouple the equation Eq. (2.10), the error of both currents is given to separate PI controllers. Furthermore the voltages are pre-controlled with

$$\begin{aligned} u_\mathrm{d}^0 &= -\omega_\mathrm{el} L_\mathrm{q} i_\mathrm{q} \\ u_\mathrm{q}^0 &= \omega_\mathrm{el} (L_\mathrm{d} i_\mathrm{d} + \Psi_\mathrm{p}), \end{aligned} \tag{2.13}$$

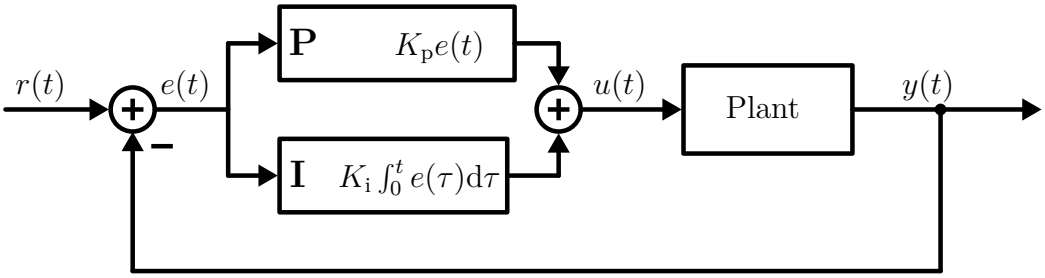


Fig. 2.10: Contol loop with PI controller.

yielding the DEs

$$\begin{aligned} \Delta u_\mathrm{d}^* &= R_\mathrm{s} i_\mathrm{d} + L_\mathrm{d} \frac{\mathrm{d}i_\mathrm{d}}{\mathrm{d}t} \\ \Delta u_\mathrm{q}^* &= R_\mathrm{s} i_\mathrm{q} + L_\mathrm{q} \frac{\mathrm{d}i_\mathrm{q}}{\mathrm{d}t} \end{aligned} \tag{2.14}$$

with

$$\begin{aligned} u_\mathrm{d}^* &= u_\mathrm{d}^0 + \Delta u_\mathrm{d}^* \\ u_\mathrm{q}^* &= u_\mathrm{q}^0 + \Delta u_\mathrm{q}^*. \end{aligned} \tag{2.15}$$

Another point that must be taken into account when converting the voltage into the stator-fixed coordinates is the angular lead. This is given by

$$\Delta\varepsilon = 1.5\tau\omega_\mathrm{el} \tag{2.16}$$

where τ is the sampling interval of the system. Due to the discretization and the time delay caused by the pulse width modulation, the angular lead is used to compensate for both effects. Another problem that arises due to the described limitations in Section 2.1.2 is the wind-up effect. This occurs when the control error is integrated although the input variable is already in the constraint. The integral part of the controller increases because it cannot be applied, which leads to the feedback into the control loop being invalid. This proportion decreases after the sign of the control error changes, but the reduction of the error requires time and therefore the control variable has to remain in the limit longer than necessary. This decreases the dynamics and can also cause instability. To prevent this problem the integral part of the controller can be activated only when the input variable is not in the constraint.

2.1.6 Simulation Environment

Since real test rigs are expensive and the realization of an experiment is time-consuming, a simulation of the electric drive is used in this work. A Python package for the simulation of

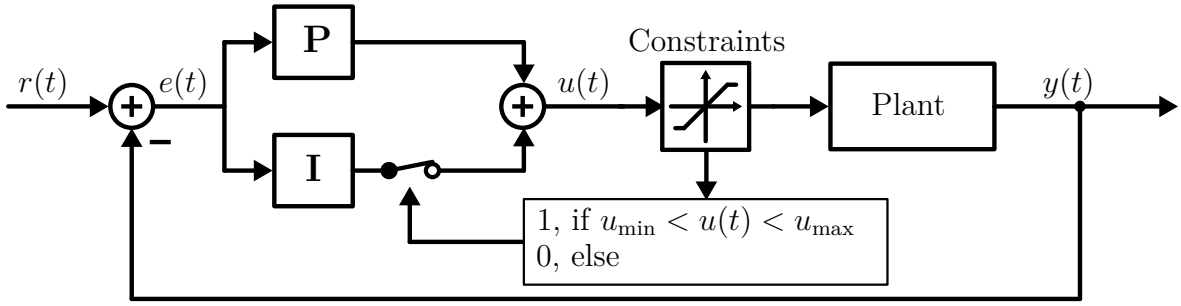


Fig. 2.11: PI controller with anti-wind-up.

electric drives, with all its above described components, is the gym-electric-motor (GEM) toolbox [10]. Furthermore, the toolbox provides classical controllers but the API can be used to apply own algorithms. Fig. 2.12 visualizes the various components of the GEM environment.

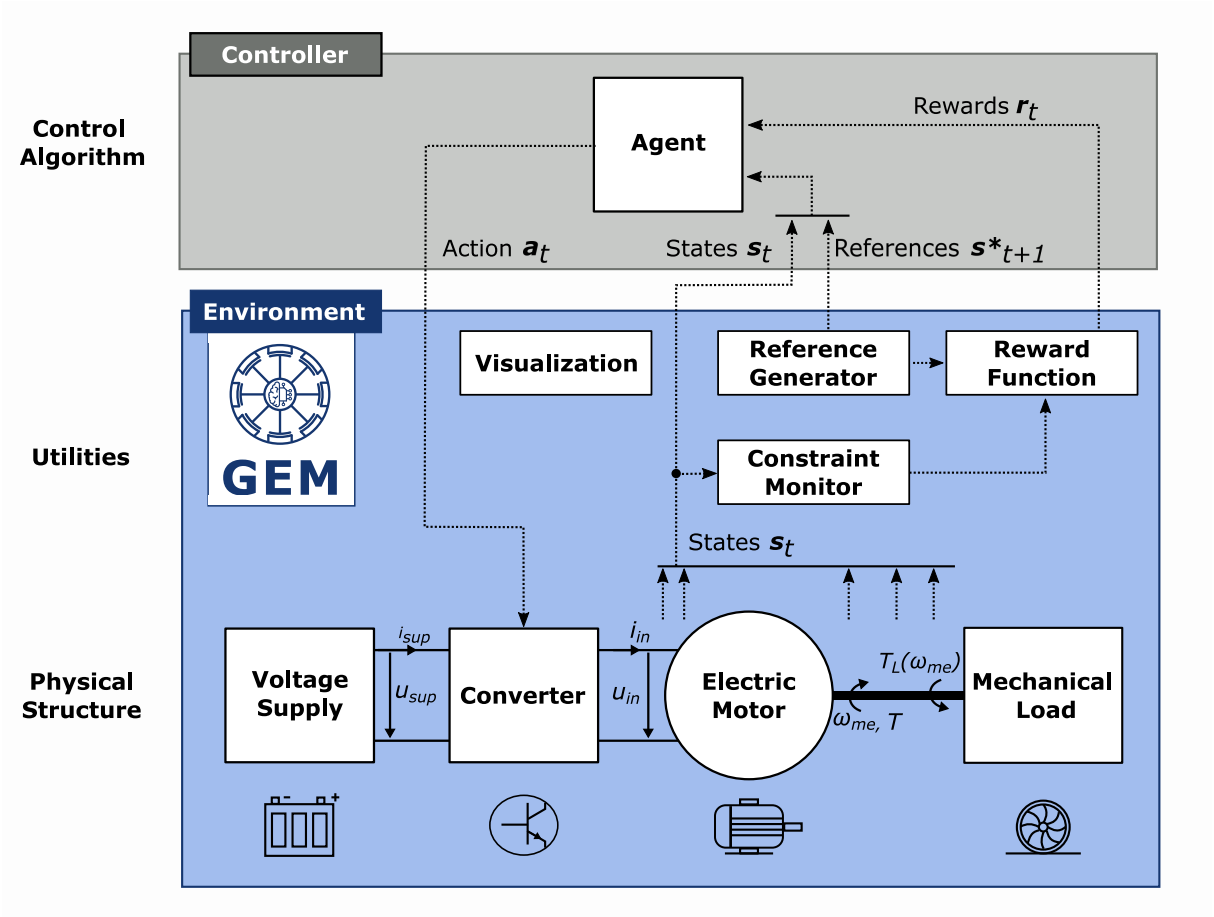
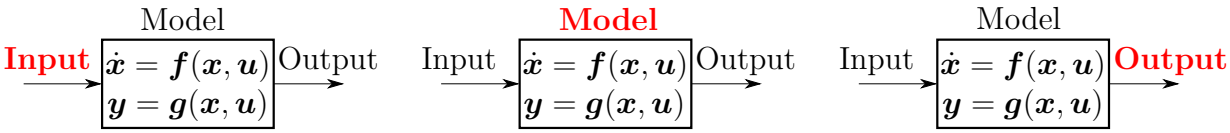


Fig. 2.12: Block diagram of the drive system created by the GEM toolbox showing the environment and the controller (cf. [10]).

Within this environment the supply voltage can be adjusted and it is also possible to choose from a variety of different converters. A range of both DC and AC motors, which includes the PMSM, are provided in the toolbox. Physical parameters of the selected motor can be set as well. In addition to the classical components, a reference generator and a reward function can be specified. The reference generator can be used to configure trajectories which output the required reference values during runtime. Commonly used functions are trigonometric, triangular, step or constant but also stochastic processes. The reward function defines a reward depending on the system states, e.g., the mean squared error (MSE) between reference and current trajectory of the motor. This is mainly used in the RL control methods. The GEM environment is based on the gym API established at OpenAI [11]. Therefore, the same functions can be used with GEM to create the classic agent environment loops [12], which will be discussed in Section 2.3.2.1.

2.2 System Identification

The system identification (SI) is a branch of systems theory that focuses on creating and validating empirical models [13][14]. Those models result from the identification of dependencies between input and output data of the system. In this context, a priori knowledge about the underlying system can be useful in order to incorporate physical relationships into the modeling. However, models can also be generated without prior knowledge through deterministic or statistical approximations.



(a) Control **(b)** Identification **(c)** Simulation

Fig. 2.13: The various sub-disciplines within system theory. Highlighted in red is the desired quantity.

In this context, it is common to refer to black-, grey- and white-box models, which will be discussed in the following.

White-box

In white-box models, an exact relation between input and output data exists or is assumed. In physical models, for example, Newtonian relationships are used. An advantage of these models is the short calculation time and the simpler interpretability. Unfortunately, this also leads to the fact that they are rarely applied because most real processes are complex by nature.

Grey-box

In the case of grey-box models, the underlying process is tried to be represented by differential equations, but with the knowledge that an exact representation is not possible. Therefore, unknown free parameters are to be determined via SI methods, which contribute to increasing the accuracy of the model.

Black-box

If a model is created using a black-box approach, no information is utilized from the underlying process. Typical examples are ANNs, which are learned by input and output data. Models of this type can be accurate, but this also comes with a high internal complexity. Therefore, they can be rarely interpreted and the calculation effort is also significantly higher.

SI is therefore always a trade-off between different aspects: Accuracy, interpretability, computational demand and design effort. Thus, depending on the application, it is necessary to choose the type of modeling that fits the problem best.

2.3 Machine Learning

This chapter deals with the basics of ML. The term ML dates back to 1959 [15] and was shaped by Arthur L. Samuel, who worked as a pioneer in the field of artificial intelligence (AI). A precise definition for this form of learning was provided by T. Mitchell:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance a tasks in T, as measured by P, improves with experience E."[16]

In the following no formal definition of the entities is to be provided but rather brief descriptions. Since a computer program is the formulation of an algorithm and its dataset, the term algorithm is used in the following, since the dataset is yet not denoted. Tasks T are possible fields of applications for ML based algorithms, where often traditional algorithms reach their limits, e.g., image classification or data clustering. In order to evaluate an algorithm based on its performance on a task, a performance measure P is required. It is important that this measure is suitable for the respective task. Experience E is gathered by an algorithm based on the learning process. For this the algorithm uses a dataset, which consists of many examples. From each of these examples, features can be extracted, with the help of which the algorithm should learn the properties of the examples.

2.3.1 Neural Networks

A subclass of ML are the ANNs or short NNs [17]. These networks are formed by artificial neurons, which are modeled after the biological brain. Many of these neurons are linked in layers and can be trained by large amounts of data. In the following sections, the structure

of the NNs will be discussed. Thereby the structures, which are also used in this work, are explained in more detail.

2.3.1.1 Structure of Neural Networks

Neural networks can be organized into layers of neurons. The first layer is often referred to as the input layer. This is where the data, or in this context also the so-called features, are fed into the network. The features then propagate further from layer to layer, being modified by the mathematical functions present in the layers. These layers, within the network, are the so-called hidden layers, where the depth of the NN is defined as the number of layers used. At the output layer of the network, the result of the entered features is then obtained. There are two main areas of application. In classification tasks, a class label is assigned to the input features, while in regression tasks, the NN assigns a floating-point value [17].

Fully-Connected Layer

A fully-connected layer is the simplest type of connection [17]. As the name implies, all output neurons of the previous layer are connected to each input neuron of the next layer. Each of these connections can be assigned a learnable weight and bias. This results in an affine transformation of the input signals, which is followed by a non-linear function. This can be represented mathematically by the following equation

$$\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \tag{2.17}$$

where $\boldsymbol{x}$ is the input of the layer and $\boldsymbol{y}$ is the output of the layer. $\boldsymbol{W}$ corresponds to the weights and $\boldsymbol{b}$ to the bias, which are both to be learned. The non-linearity $\boldsymbol{g}(\cdot)$ is also referred to as the activation function, because it determines whether neurons in the next layer become active. Typical functions are tanh [17] and rectified linear units (ReLUs) [17], as well as modified functions of the latter [18][19][20]. The typical structure of an NN which has fully connected layers is shown in Fig. 2.14.

Residual Connections

Another configuration that is mainly used in really deep networks are the so-called residual blocks [21]. Here, the residual $\boldsymbol{x}_i$ of a block of layers $\boldsymbol{f}(\boldsymbol{x}_i)$ is added to the output of this block $\boldsymbol{y}_i$. A block can be composed of one or more layers regardless of their function. The purpose of this is to ensure that gradient information is better transferred to the foremost layers when the error is traced back. NN where those residual blocks are mainly used are refereed us residual neural networks (ResNets) [21]. Fig. 2.15 shows a NN with residual blocks.

2.3.1.2 Training

Now that NN has been described structurally, it is important to understand how the parameters are determined. Therefore these models go through a learning process, refereed

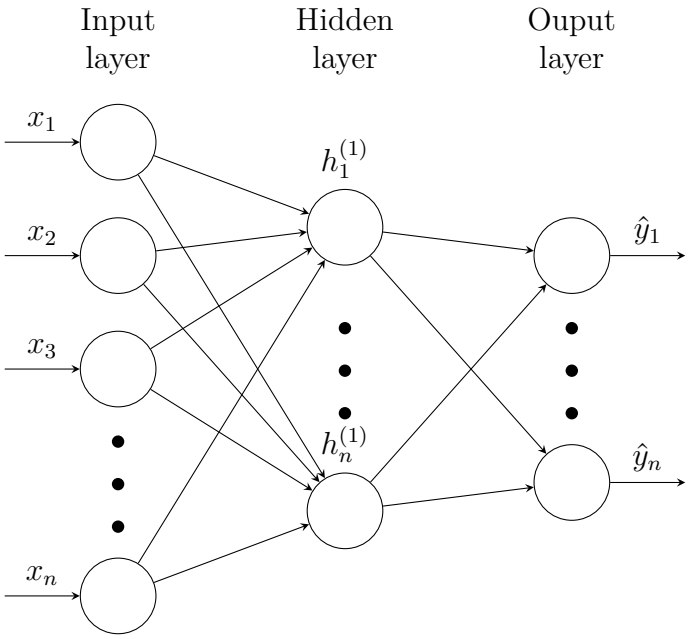


Fig. 2.14: Visualization showing the structure of a small NN with three layers, which are all fully connected.

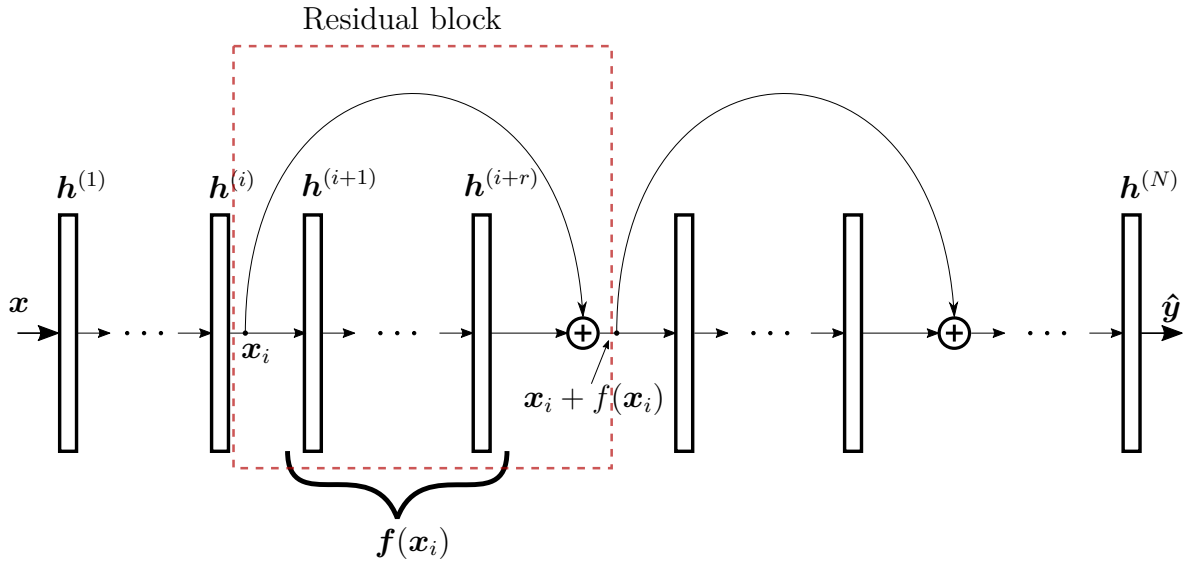


Fig. 2.15: Example implementation of a NN with residual blocks. Blocks can contain arbitrary number of layers, which in themselves can have different functions. If the size changes within the skipped block, it must also be adjusted in the skip connection. These blocks can be used alongside other layers.

to as training. Based on the examples seen in the training and with the help of gradient methods, parameters can be learned. Following is the derivation of the parameter updates and then it will be described how these gradients can be determined efficiently [17].

Parameter Update

Assume a dataset $\mathcal{D}$ consists of N data tuples $(\boldsymbol{x}_n, \boldsymbol{y}_n)$ and an NN $f(\boldsymbol{x}_n; \boldsymbol{\theta})$ which predicts an output $\hat{\boldsymbol{y}}_n$. The local loss for each of these tuples can then be written as $\mathcal{L}_{\text{local}}(\hat{\boldsymbol{y}}_n, \boldsymbol{y}_n)$, where summing over all data points gives the global loss

$$\mathcal{L}_{\text{global}}(\mathcal{D}; \boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_{\text{local}}(\hat{\boldsymbol{y}}_n, \boldsymbol{y}_n) \tag{2.18}$$

with

$$\hat{\boldsymbol{y}}_n = f(\boldsymbol{x}_n; \boldsymbol{\theta}). \tag{2.19}$$

The preferred loss function depends on the application, e.g., the MSE is often used for regression tasks [17]. A low loss follows from a good model, therefore the goal is to find an optimal set of parameters using gradient descent methods. To update the parameters of the network a gradient step is performed. The stochastic gradient descent (SGD) is one of the most common methods for calculating gradients, where the minimization of the global loss is achieved by repeated calculation of gradients on smaller random subsets $\mathcal{D}_{\text{b}}$ (mini-batch), which is sampled from $\mathcal{D}$

$$\frac{\partial \mathcal{L}_{\text{global}}(\mathcal{D}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \approx \frac{\partial \mathcal{L}_{\text{global}}(\mathcal{B}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{1}{|\mathcal{B}|} \sum_{(x_n, y_n) \in \mathcal{B}} \frac{\partial \mathcal{L}_{\text{local}}(\hat{\boldsymbol{y}}_n, \boldsymbol{y}_n)}{\partial \boldsymbol{\theta}}. \tag{2.20}$$

This leads to the update rule

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial \mathcal{L}_{\text{global}}(\mathcal{B}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}, \tag{2.21}$$

with the learning rate η. In summary, to train the NN, it is necessary to calculate the derivative of the loss function w.r.t. the NN parameters. Thus, by gradient descent a parameter set can be determined, which leads to a proper model fit. The SGD is often only a basis for a more complex algorithm, such as the adaptive moment estimation (Adam) [22]. Besides, there are also other methods of gradient descent [23], e.g., Levenberg–Marquardt algorithm [24] or Nelder-Mead method [25].

Backpropagation

To calculate the gradients in the NN efficiently, the chain rule is used. Consider a network consisting of N chained functions $f^1, f^2, ..., f^N$, each parameterized by a parameter set $\boldsymbol{\theta}^i$, yielding

$$\hat{\boldsymbol{y}} = f^{(1:N)}(\boldsymbol{x}; \boldsymbol{\theta}^{(1:N)}) = f^{(N)} \left(f^{(N-1)} \left(\cdots f^{(1)}(\boldsymbol{x}; \boldsymbol{\theta}^{(1)}) \cdots , \boldsymbol{\theta}^{(N-1)} \right) \boldsymbol{\theta}^{(N)} \right) \tag{2.22}$$

The activation of a neuron j within a the n-th hidden layer results from the output of neuron i of the previous hidden layer $h^{(n-1)}$. This is then transformed by weight w_{ji} and

bias b_j and assigned the activation function $g(\cdot)$. If summed up for all inputs $I^{(n-1)}$ this yields

$$h_j^{(n)} = g_n \left(\sum_{i \in I^{(n-1)}} w_{ji}^{(n)} h_i^{(n-1)} + b_j^{(n)} \right). \tag{2.23}$$

This can be summarized in general for a hidden layer n via matrix notation:

$$\boldsymbol{h}^{(n)} = g_n \left(\boldsymbol{h}^{(n-1)} \boldsymbol{W}^{(n)\top} + \boldsymbol{b}^{(n)} \right) \tag{2.24}$$

with

$$\begin{aligned} \boldsymbol{W}^{(n)\top} &= \begin{pmatrix} w_{1,1}^{(n)} & \cdots & w_{1,I^{(n)}}^{(n)} \\ \vdots & \ddots & \vdots \\ w_{I^{(n-1)},1}^{(n)} & \cdots & w_{I^{(n-1)},I^{(n)}}^{(n)} \end{pmatrix}, \\ \boldsymbol{h}^{(n-1)} &= (h_1^{(n)} \cdots h_{I^{(n-1)}}^{(n)}), \\ \boldsymbol{b}^{(n)} &= (b_1^{(n)} \cdots b_{I^{(n)}}^{(n)}). \end{aligned}$$

The parameters of a layer n are summarized for convenience in the vector $\boldsymbol{\theta}^{(n)}$. Together with the chain rule, an expression for calculating the gradient of the layer n can be formulated:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}^{(n)}} = \frac{\partial \mathcal{L}}{\partial h^{(N-1)}} \frac{\partial h^{(N-1)}}{\partial h^{(N-2)}} \cdots \frac{\partial h^{(n+1)}}{\partial \boldsymbol{\theta}^{(n)}} \tag{2.25}$$

If the gradient $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}^{(n-1)}}$ is also to be determined, the previously calculated result $\frac{\partial \mathcal{L}}{\partial h^{(N-1)}}$ in Eq. (2.25) can be used. From the chain rule follows

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}^{(n-1)}} = \frac{\partial \mathcal{L}}{\partial h^{(N-1)}} \frac{\partial h^{(N-1)}}{\partial h^{(N-2)}} \cdots \frac{\partial h^{(n+1)}}{\partial \boldsymbol{\theta}^{(n)}} \frac{\partial h^{(n)}}{\partial \boldsymbol{\theta}^{(n-1)}} \tag{2.26}$$

which can be repeated in this way for the remaining N parameters. The parameters of the network are then updated by the backpropagation of error, or short backpropagation [26]. This method is based on automatic differentiation (AD) [27]. In the forward pass through the NN a computational graph is built up in which operations and associated derivations are stored. In the backward pass this graph is utilized to compute the gradients. Common packages for Python where AD is already directly implemented are, e.g., `PyTorch` [28] and `TensorFlow` [29].

2.3.2 Reinforcement Learning

Besides supervised and unsupervised learning, which is used in the previously described NNs, there is another type of ML application, RL [30]. The focus lies on finding a policy function for cost-optimal, sequential decisions that affect an environment. The unit acting according to the policy is called agent. This agent learns from experience by receiving feedback on whether the action leads to a desired outcome in the environment. In contrast to supervised learning, no input/output tuples are specified here. An agent can therefore find a suitable policy even without information about the environment. This section covers some general information about RL and afterwards one type of agent is explained in more detail.

2.3.2.1 Agent-Enviroment Setup

In this setup, an agent is inserted into an environment and can take actions for which it receives a reward. Besides the reward, the states of the environment are changing, which the agent can observe and perform another action based on it. In this loop, the agent's goal is to maximize the accumulated reward. This is illustrated in Fig. 2.16.

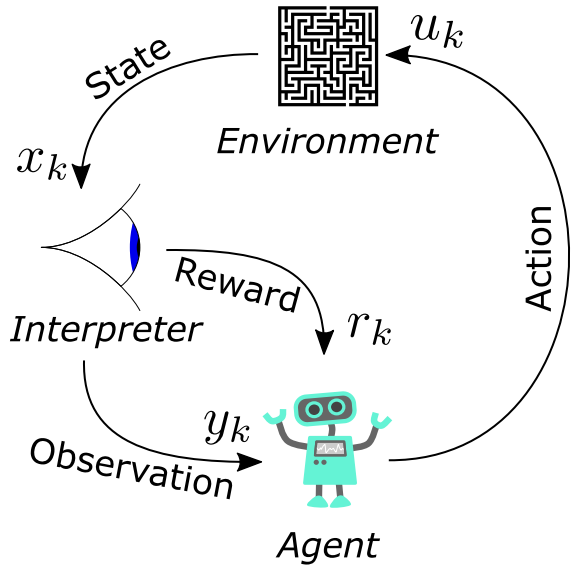


Fig. 2.16: Illustration of a typical agent-environment setup (cf. [31]).

As described, the agent interacts with the environment but does not need a priori information about it to learn from it. Secondly, the selected actions do not have to be optimal. In order to be able to develop an optimal policy, that is, a rule according to which actions are selected, it is always important to explore. On the other hand, the agent should be able to choose accurate actions at some point in time. Therefore, the goal is to find a good balance between exploration and exploitation [31].

2.3.2.2 Deep Deterministic Policy Gradient

The choice of the agent depends mostly on the environment, where for continuous state and action spaces the deep deterministic policy gradient (DDPG) agent [32] is a possible choice. It includes four NNs: an actor, a target actor, a critic and a target critic network. The action $\boldsymbol{u}_k$ results from the current state of the environment $\boldsymbol{x}_k$ given the parameters $\boldsymbol{\theta}$ present in actor network. Since the chosen action is always deterministic, this algorithm does not explore on its own. Therefore, an action noise $\mathcal{N}$, often implemented by a stochastic process, e.g., Ornstein-Uhlenbeck (see Eq. (A.5)), is added to the selected action in training:

$$\boldsymbol{u}_k = \mu(\boldsymbol{x}_k; \boldsymbol{\theta}) + \mathcal{N}. \tag{2.27}$$

The actor network $\mu(\boldsymbol{x}_k; \boldsymbol{\theta})$ is often referred to as policy of the DDPG agent. For each transition, the current state $\boldsymbol{x}_k$, current action $\boldsymbol{u}_k$ as also the received reward r_k and future state $\boldsymbol{x}_{k+1}$ is stored in a data buffer $\mathcal{D}$. This data tuple is often referred to as experience and will be used to update the policy. If the data buffer is sufficiently filled, a mini-batch $\mathcal{D}_\mathrm{b}$ is randomly sampled containing N data tuples

$$(\boldsymbol{x}_i, \boldsymbol{u}_i, r_i, \boldsymbol{x}_{i+1}). \tag{2.28}$$

Based on the states, the targets for the Q-function of the critic is given by the Bellman equation

$$\hat{q}_i = r_i + \gamma Q'(\boldsymbol{x}_{i+1}, \boldsymbol{u}'_i; \boldsymbol{w}^-) \tag{2.29}$$

with

$$\boldsymbol{u}'_i = \mu'(\boldsymbol{x}_i; \boldsymbol{\theta}^-). \tag{2.30}$$

The function $Q'(\boldsymbol{x}_{i+1}, \boldsymbol{u}'_k; \boldsymbol{w}^-)$ is the target critic network, which outputs the expected future reward for the given state action combination, where the action $\boldsymbol{u}'_k$ is given by the target actor network μ'. $\gamma \in [0, 1)$ is the discount factor, causing events in the present to be weighted more than events in the near future. Now the target Q-value $\hat{q}_i$ and the Q-value of the critic network

$$q_i = Q(\boldsymbol{x}_i, \boldsymbol{u}_i; \boldsymbol{w}) \tag{2.31}$$

are used to calculate the mean squared Bellman error (MSBE)

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} (q_i - \hat{q}_i)^2. \tag{2.32}$$

With this loss gradients can be computed to updated the parameters $\boldsymbol{\theta}$ and $\boldsymbol{w}$ within a gradient descent fashion (see Eq. (2.21)). The update of the target networks is delayed by

$$\begin{aligned} \boldsymbol{w}^- &\leftarrow (1-\delta)\boldsymbol{w}^- + \delta \boldsymbol{w} \\ \boldsymbol{\theta}^- &\leftarrow (1-\delta)\boldsymbol{\theta}^- + \delta \boldsymbol{\theta} \end{aligned} \tag{2.33}$$

where $\delta \ll 1$. This update of the target network corresponds with a low-pass characteristic, therefore abrupt changes are dampened. The minimization of MSBE becomes more stable and thus the training of the DDPG agent becomes more robust. The structure of this agent is summarized in Fig. 2.17, a detailed algorithm is attached to the appendix (see Algorithm 1).

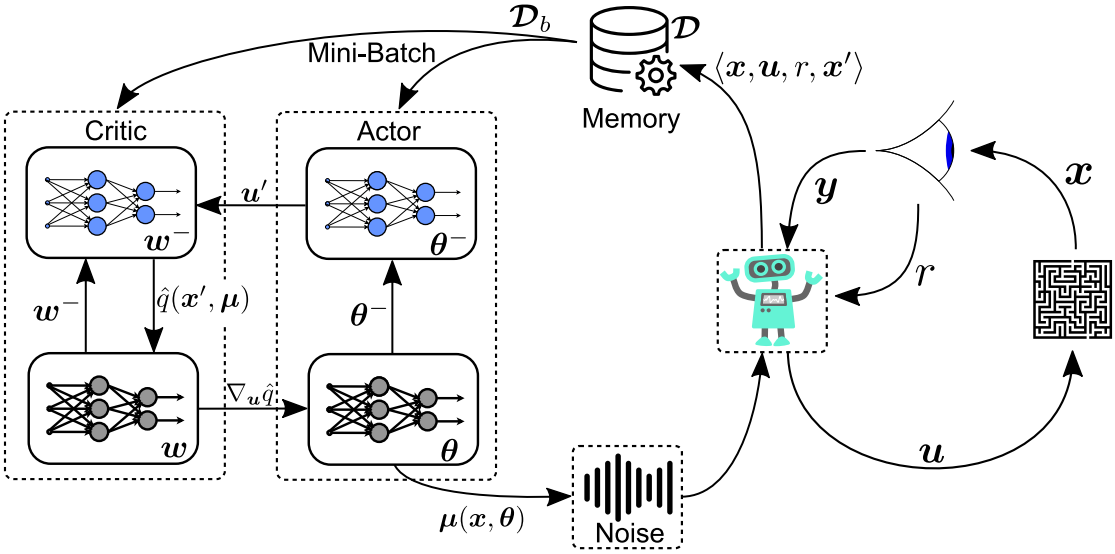


Fig. 2.17: Summarized DDPG agent (cf. [31]).

3 Modeling

This part of the work deals with the elaboration of the task formulated in the introduction. The goal is to use NODE to perform both current control and system identification. For this purpose, the concept of NODE and how they are applied within this work will be explained in more depth. Subsequently, the creation of the test data and the splitting of the data sets will be discussed. Finally, it is clarified how optimal settings of the system can be found to increase performance.

3.1 Neural Ordinary Differential Equations

As been shown in [33] and [34], the depth of an NN contributes significantly to its performance in certain tasks. At the same time the training of these deeper NNs is more difficult [35][36], due to the increased complexity in the connection between the parameters and the objective of the NN. One solution to the problem is the ResNet [21], which implements skip connections over multiple layers. This reduces the complexity and thus makes it feasible to add more layers.

Examining the resulting structure, one can find forms that typically occur in the numerical solution to ODE, using Euler's Method [37](see Appendix A.2). An idea that results from this realization is to express the ODE as an implicit layer in the NN, in order to take advantage of both methods. This approach is called NODE [38]. The NODE defines a parameterizable function $\boldsymbol{f}(t, \boldsymbol{x}(t); \boldsymbol{\phi})$ with a parameter set $\boldsymbol{\phi}$ that determines the change in the hidden state $\boldsymbol{x}(t)$:

$$\frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = \dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), t; \boldsymbol{\phi}), \tag{3.1}$$

In other words, the NODE is a learnable vector field and therefore defined continuously in time [38]. Since $\boldsymbol{f}_{\boldsymbol{\phi}}(\boldsymbol{x}(t), t) := \boldsymbol{f}(\boldsymbol{x}(t), t; \boldsymbol{\phi})$ is still an autonomous system, the definition is to be rewritten such that the equation receives an input vector $\boldsymbol{u}(t)$

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t; \boldsymbol{\phi}). \tag{3.2}$$

Under the assumption that the system is linear, Eq. (3.2) can be rewritten to

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}_{\boldsymbol{\phi}}(\boldsymbol{x}(t), \boldsymbol{u}(t), t) = \boldsymbol{A}_{\boldsymbol{\phi}}\boldsymbol{x}(t) + \boldsymbol{B}_{\boldsymbol{\phi}}\boldsymbol{u}(t) \tag{3.3}$$

where $\boldsymbol{A}_{\boldsymbol{\phi}}$ and $\boldsymbol{B}_{\boldsymbol{\phi}}$ are system matrices which contain the learnable parameters.

To train the NODE, i.e., the NN, a loss function is needed. In [39] the following cost function was established

$$\mathcal{L}(\boldsymbol{x}_0) = \underbrace{\int_0^T l(\xi_{\boldsymbol{\phi}}(\boldsymbol{x}_0, \boldsymbol{u}(\tau), \tau), \boldsymbol{u}(\tau), \tau)\mathrm{d}\tau}_{\text{integral cost}} + \underbrace{L(\xi_{\boldsymbol{\phi}}(\boldsymbol{x}_0, \boldsymbol{u}(T), T), \boldsymbol{u}(T), T)}_{\text{terminal cost}}. \tag{3.4}$$

The first term reflects the integral costs that occur over the trajectory within the integration interval $[0, T]$ and the second term specifies the cost in the terminal point T. The function $\xi_{\boldsymbol{\phi}}(\boldsymbol{x}_0, \boldsymbol{u}(t), t)$ with a parameter set $\boldsymbol{\phi}$ describes the solution of the initial value problem with exogeneous input [39]. To determine the loss, however, Eq. (3.1) must first be solved. For this purpose, the equation is to be set up as the following initial value problem (IVP):

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}_{\boldsymbol{\phi}}(\boldsymbol{x}, t) \qquad \text{s.t. } \boldsymbol{x}(t_0) = \boldsymbol{x}_0 \tag{3.5}$$

The IVP is to be solved within the interval $t \in [0, T]$, by conventional ODE solvers with either fixed step size (e.g., Euler's method, Runge-Kutta [40][41]) or adaptive step size (e.g., Dormand-Prince [42]). With the result, the loss can be determined, which in turn can be used to approximate the necessary gradients via the typical approach of AD. The problem is not to compute and apply the gradients via backpropagation through the ODE solvers but rather the memory usage related to this approach [38]. If the ODE solver requires N steps to solve the IVP on the interval $[0, T]$, the computational cost as well as the memory requirement for AD can be described by $\mathcal{O}(N)$. This means that if the ODE is to be solved with sufficient accuracy and the time interval is selected to be large, thereby increasing the number of necessary steps, the memory requirement increases significantly. Thus, the calculation of the gradients is performed using the adjoint method, which was first developed by Pontryagin [43] and then adopted for the use of NODE [38]. The great benefit of this method is the constant memory efficiency given by $\mathcal{O}(1)$. In the following, the adjoint method will be discussed in more detail in the context of the NODE.

Adjoint Method

The generalized form of the adjoint method is considered here. Given the IVP Eq. (3.5) and the cost function Eq. (3.4) the derivative with respect to the parameters then results in

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\phi}} = \frac{\partial L}{\partial \boldsymbol{\phi}} + \int_0^T \left(\boldsymbol{a}(t)^\top \frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}}(\boldsymbol{x}, t)}{\partial \boldsymbol{\phi}} + \frac{\partial l}{\partial \boldsymbol{\phi}} \right) \mathrm{d}\tau. \tag{3.6}$$

The $\boldsymbol{a}(t)$ is the Lagrangian multiplier which has to satisfy

$$\frac{\mathrm{d}\boldsymbol{a}(t)^\top}{\mathrm{d}t} = -\boldsymbol{a}(t)^\top \frac{\partial \boldsymbol{f}_{\boldsymbol{\phi}}(\boldsymbol{x}, t)}{\partial \boldsymbol{x}} - \frac{\partial l}{\partial \boldsymbol{x}}, \qquad \text{s.t. } \boldsymbol{a}(T)^\top = \frac{\partial L}{\partial \boldsymbol{x}(T)}.$$

This equation can be solved by another call of the ODE solver, which solves this terminal value problem backwards in time. The gradients are calculated accurate, if the ODE was solved accurately as well. The numerical errors can therefore be adjusted by a suitable choice of the error tolerance [38][39].

3.2 Internal Plant Model with Neural Controller

The goal is to exploit the concept of the NODE in the first step for a current control. Subsequently, the controller is to be extended to perform system identification. For the current control is first performed on an internal plant model (IPM), which has perfect model knowledge.
In the following, it will be discussed how this IPM is structured and how the current control is performed on it. In the next section the setup is extended by the system identification.

3.2.1 Structure

As shown in Eq. (3.3), the NODE can be rewritten to a linear dynamic system. This is exploited to integrate the ODEs of the PMSM into the IPM. For this purpose, ODEs from Eq. (2.10) are first to be converted into vector notation in order to determine the system matrices (and the vector). This results in

$$\boldsymbol{A} = \begin{pmatrix} -\frac{R_\mathrm{s}}{L_\mathrm{d}} & \omega_\mathrm{el}\frac{L_\mathrm{q}}{L_\mathrm{d}} \\ -\omega_\mathrm{el}\frac{L_\mathrm{d}}{L_\mathrm{q}} & -\frac{R_\mathrm{s}}{L_\mathrm{q}} \end{pmatrix}, \tag{3.7}$$

$$\boldsymbol{B} = \begin{pmatrix} \frac{1}{L_\mathrm{d}} & 0 \\ 0 & \frac{1}{L_\mathrm{q}} \end{pmatrix} \tag{3.8}$$

and

$$\boldsymbol{e} = \begin{pmatrix} 0 \\ -\frac{\Psi_\mathrm{p}}{L_\mathrm{q}}\omega_\mathrm{el}. \end{pmatrix} \tag{3.9}$$

The $\boldsymbol{e}$ is only a constant excitation, i.e., a shifting of the equilibrium. The vector $\boldsymbol{e}$ is kept within the IPM resulting in an affine linear system:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) + \boldsymbol{e}. \tag{3.10}$$

The state vector $\boldsymbol{x}(t)$ of the IPM contains the actual currents in dq-coordinates:

$$\boldsymbol{x}(t) = \begin{pmatrix} i_\mathrm{d}(t) \\ i_\mathrm{q}(t) \end{pmatrix}, \tag{3.11}$$

while the input vector $\boldsymbol{u}(t)$ contains the voltages in dq-coordinates:

$$\boldsymbol{u}(t) = \begin{pmatrix} u_\mathrm{d}(t) \\ u_\mathrm{q}(t) \end{pmatrix}. \tag{3.12}$$

What can be observed is that a structure has now been brought into the IPM. From the incorporation of this additional information, the black-box model can be transformed into a grey-box model.

3.2.2 Current Control via Neural Controller

Since in this case perfect model knowledge within the IPM is assumed, Eqs. (3.7) to (3.9) contain the true parameters of the system to be controlled. In order to control the current of the PMSM or here the IPM, the voltage at the motor must be set. For this purpose, the input vector $\boldsymbol{u}(t)$ is selected by an NN, which is subsequently referred to as neural controller (NC). This vector is defined on the basis of the policy of the NC

$$\boldsymbol{u}(t) = \boldsymbol{\pi}(\boldsymbol{x}_\mathrm{aug}(t), t; \boldsymbol{\theta}), \tag{3.13}$$

which is described by the parameter set $\boldsymbol{\theta}$. This policy receives as input features an augmented state vector

$$\widetilde{\boldsymbol{x}}_\mathrm{aug}(t) = \begin{pmatrix} \hat{\widetilde{\boldsymbol{x}}}(t) \\ \widetilde{\boldsymbol{x}}^*(t) \end{pmatrix} \tag{3.14}$$

$$= \begin{pmatrix} \hat{\widetilde{i}}_\mathrm{d}(t) & \hat{\widetilde{i}}_\mathrm{q}(t) & \widetilde{i}^*_\mathrm{d}(t) & \widetilde{i}^*_\mathrm{q}(t) \end{pmatrix}^\top. \tag{3.15}$$

It contains the current estimate $\hat{\tilde{\boldsymbol{x}}}(t)$ and the reference value $\tilde{\boldsymbol{x}}^*(t)$ of the state vector. The tilde above the quantities indicates a normalization, which is performed by dividing through the limit s of the respective quantity. Since the input features were normalized, the policy outputs the normalized input vector $\tilde{\boldsymbol{u}}(t)$, which again has to be multiplied with the limit to get the denormalized quantity:

$$\boldsymbol{u}(t) = s_{\mathrm{u}} \tilde{\boldsymbol{u}}(t). \tag{3.16}$$

In summary, the IPM accurately models the PMSM. The current control is done by the NC which provides the necessary input voltages given the policy $\boldsymbol{\pi_\theta}$, which is to be learned in the training. How the gradients can be determined has been explained in the previous sections, but not the practical implementation. For these, appropriate software frameworks are required. In Python the popular packages for these application are `torchdiffeq` [44] and `torchdyn` [45]. Both are based on `PyTorch` and have a similar usage, in this work the latter package is used.
The call of an ODE solver looks like this:

$$\hat{\boldsymbol{x}}(t_1), \hat{\boldsymbol{x}}(t_2), \cdots, \hat{\boldsymbol{x}}(t_N) = \mathbf{ODEsolver}(\boldsymbol{x}_0, \boldsymbol{f}(\hat{\boldsymbol{x}}(t), \boldsymbol{\pi_\theta}(\tilde{\boldsymbol{x}}_{\mathrm{aug}}(t), t), t), [t_1, t_2, \cdots, t_N]). \tag{3.17}$$

Besides the initial values $\boldsymbol{x}_0$, the solver takes the IPM as input argument and also the time steps $t_n = n \cdot t_s$, where t_s corresponds to the sampling interval. This allows the specification where the solver should evaluate the ODE. In solvers with a fixed step size, the procedure is evaluating a point in time and step to the next one, whereas adaptive methods do not have a fixed number between evaluating two points. Although this work does not deal with systems that have to be evaluated at runtime, a method with a fixed step size shall be chosen to ensure convergence in a reasonable time. In the Eq. (3.17), the NODE, or here in our application the IPM, is evaluated at the given points in time, so that the output is a trajectory of estimated states $\hat{\boldsymbol{x}}$. This trajectory is then used to determine the loss with the help of a suitable cost function. The goal of the NC is to follow the reference trajectories but under the constraint not to exceed the current limits present in the PMSM, therefore a composed loss is defined as

$$\mathbf{CL}(\hat{\tilde{\boldsymbol{x}}}, \tilde{\boldsymbol{x}}^*) = \lambda \cdot \mathbf{MSE}(\hat{\tilde{\boldsymbol{x}}}, \tilde{\boldsymbol{x}}^*) + (1 - \lambda) \cdot \mathcal{B}(\hat{\tilde{\boldsymbol{x}}}). \tag{3.18}$$

λ is the weighting of the terms, while $\mathcal{B}$ is a barrier function defined as

$$\mathcal{B}(\hat{\tilde{\boldsymbol{x}}}) = \frac{1}{N} \sum_{n=1}^{N} c_1 \left(\frac{1}{1 + \exp\left(-c_2 \left(\hat{\tilde{\boldsymbol{i}}}_{\mathrm{total}}[t_n] - c_3 \right) \right)} \right) \tag{3.19}$$

with

$$\hat{\tilde{\boldsymbol{i}}}_{\text{total}} = \sqrt{\left(\hat{\tilde{\boldsymbol{i}}}_{\text{d}}\right)^2 + \left(\hat{\tilde{\boldsymbol{i}}}_{\text{q}}\right)^2}. \tag{3.20}$$

The barrier function averages over the normalized total current trajectory $\hat{\tilde{\boldsymbol{i}}}_{\text{total}}$, which is additionally scaled with a sigmoid. The scaling is defined by c_1, c_2 defines the slope in the transition and c_3 corresponds to the shift of the sigmoid (see Fig. A.1). Exceeding the allowed total current at any time will add additional but bounded costs due to the sigmoid. An exponential barrier function would lead to instabilities in the calculation of the gradients if the limit is exceeded for a longer period of time.
With the help of this loss, gradients can now be calculated, which allow the parameters of the NC to be learned via the classical gradient methods. In the block diagram Fig. 3.1 the setup is summarized.

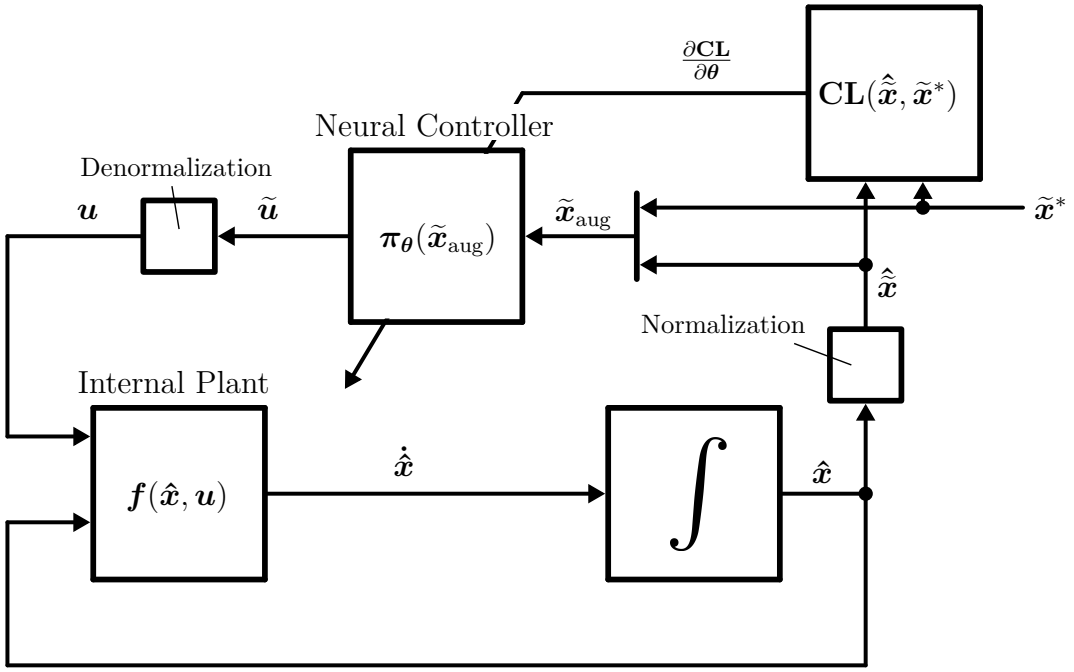


Fig. 3.1: Block diagram of the current control via the NC. Time dependencies have been omitted here in favor of a readable illustration.

The current control can now be performed on the IPM, but since the IPM describes perfect knowledge of the system, this is a special case. In reality the current control should run on a separate PMSM, where the parameters usually are not known. For this reason, the setup is now to be further expanded.

3.2.3 Current Control and System Identification

The control of an external plant (EP) is the actual goal. As in the previous section, a NC is trained for this purpose. The problem is that the EP, being a black-box, does not

provide a chain of mathematical operations along which gradients could be computed, which are needed to optimize the NC. Therefore, the IPM is used in order to identify the assumed constant values of the EP. On the basis of the IPM, gradients can then be obtained which are utilized to train the NC. Hence, the learnable parameters represented in Eq. (3.3) are now used. Since the current control is executed via the NC defined in the previous section, the system can be described by

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}_{\boldsymbol{\phi}}(\boldsymbol{x}(t), \boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{x}(t), t), t) = \boldsymbol{A}_{\boldsymbol{\phi}}\boldsymbol{x}(t) + \boldsymbol{B}_{\boldsymbol{\phi}}\boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{x}(t), t) \tag{3.21}$$

where

$$\boldsymbol{A}_{\boldsymbol{\phi}} = \begin{pmatrix} -\frac{\hat{R}_\mathrm{s}}{\hat{L}_\mathrm{d}} & \omega_\mathrm{el}\frac{\hat{L}_\mathrm{q}}{\hat{L}_\mathrm{d}} \\ -\omega_\mathrm{el}\frac{\hat{L}_\mathrm{d}}{\hat{L}_\mathrm{q}} & -\frac{\hat{R}_\mathrm{s}}{\hat{L}_\mathrm{q}} \end{pmatrix}, \quad \boldsymbol{B}_{\boldsymbol{\phi}} = \begin{pmatrix} \frac{1}{\hat{L}_\mathrm{d}} & 0 \\ 0 & \frac{1}{\hat{L}_\mathrm{q}} \end{pmatrix} \tag{3.22}$$

and

$$\boldsymbol{e}_{\boldsymbol{\phi}} = \begin{pmatrix} 0 \\ -\frac{\hat{\Psi}_\mathrm{p}}{\hat{L}_\mathrm{q}}\omega_\mathrm{el} \end{pmatrix}. \tag{3.23}$$

Here the available information about the structure of the PMSM ODEs is fully exploited. If the entire system is to be identified, exactly four values must therefore be identified. In this case, the parameters $\hat{L}_\mathrm{d}$, $\hat{L}_\mathrm{q}$, $\hat{R}_\mathrm{s}$ and $\hat{\Psi}_\mathrm{p}$. The hat indicating that these are the predictions i.e., the learnable parameters, that will be optimized/identified during the training.
The following steps are then carried out for the control and system identification. First, the control of the EP by the NC produces a state trajectory $\boldsymbol{x}(t_1), \boldsymbol{x}(t_2), \ldots, \boldsymbol{x}(t_N)$. The identical controller is then used to control the IPM by calling the ODE solver

$$\hat{\boldsymbol{x}}(t_1), \hat{\boldsymbol{x}}(t_2), \cdots, \hat{\boldsymbol{x}}(t_N) = \mathbf{ODEsolver}(\boldsymbol{x}_0, \boldsymbol{f}_{\boldsymbol{\phi}}(\hat{\boldsymbol{x}}(t), \boldsymbol{\pi}_{\boldsymbol{\theta}}(\widetilde{\boldsymbol{x}}_\mathrm{aug}(t), t), t), [t_1, t_2, \cdots, t_N]), \tag{3.24}$$

which leads to the estimated state trajectory. Here the case can occur that the trajectories do not have identical lengths. For example, the EP may have been aborted sooner due to limits being exceeded. In this work, therefore, an additional zero padding was inserted, which keeps the state trajectory of the EP at a constant length. The estimated trajectory of the state is multiplied by zeros depending on the trajectory length of the EP, in order

to neglect the information gained after termination. After the normalization of both trajectories, the MSE is then used to obtain the loss

$$\mathcal{L}(\hat{\tilde{\boldsymbol{x}}}, \tilde{\boldsymbol{x}}) = \mathbf{MSE}(\hat{\tilde{\boldsymbol{x}}}, \tilde{\boldsymbol{x}}), \tag{3.25}$$

which in turn is used to adapt the parameters of the IPM. The parameters of the NC are not updated in this step, since the objective is not to fit the trajectories between IPM and EP, but rather follow the actual reference trajectory. Therefore, after the update of the IPM, another trajectory is created according to Eq. (3.24) by the IPM with updated parameters. The trajectories are also modified according to the length of the run within the EP. The modified state estimation trajectory is then used along with the reference trajectory according to Eq. (3.18) to provide the necessary loss for the updated parameter of NC. Block diagram Fig. 3.2 is intended to summarize the structure, with a detailed implementation of the code added to the appendix (see Algorithm 2).

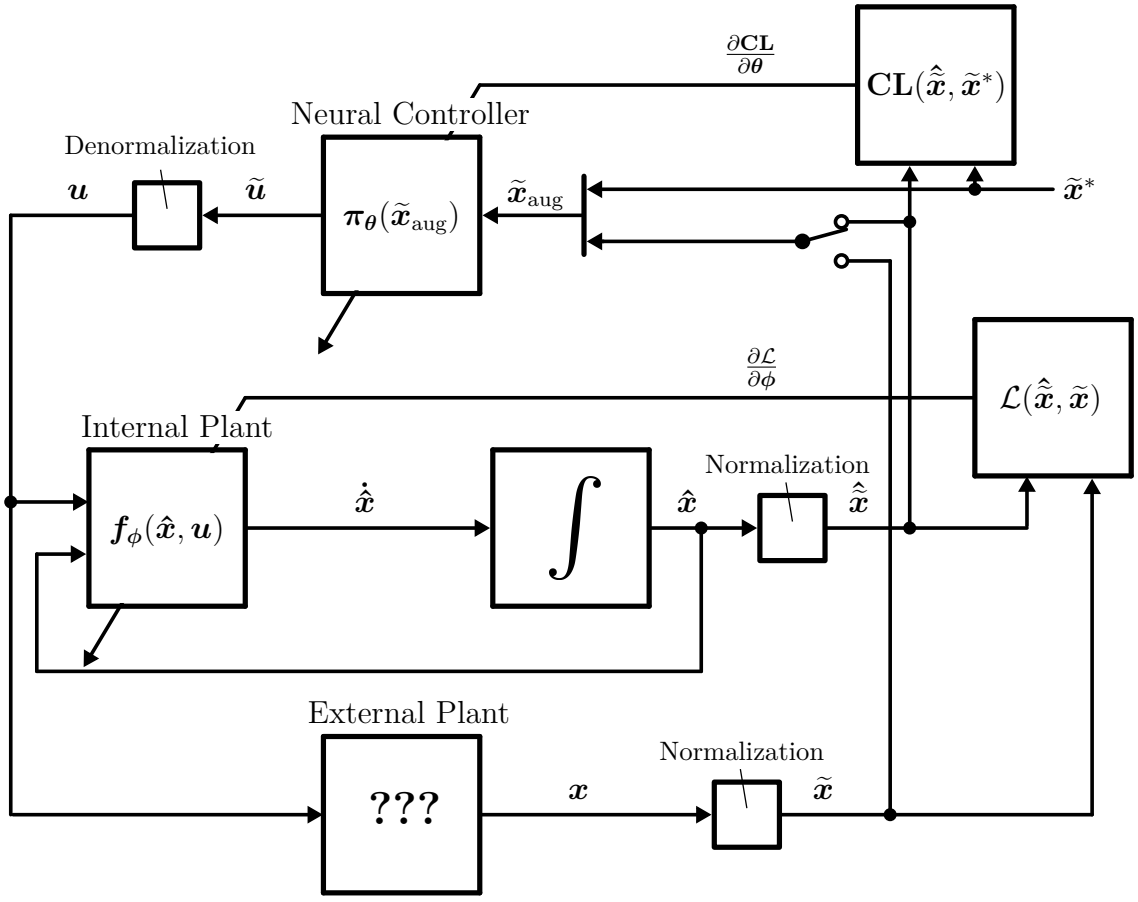


Fig. 3.2: Block diagram of the current control via the NC and simultaneously system identification of the EP by fitting the IPM. Time dependencies have been omitted here in favor of a readable illustration.

3.3 Data Generation

Now that it has been clarified how the NODEs can be trained, it should be explained with which example trajectories. It will be explained how these trajectories were created and how the data sets for the training are composed.
Besides gradients, the training of the NODE also requires examples from which the parameters of the NC but also IPM can be learned. For this purpose, a data set with reference trajectories for the current components d and q is generated in advance. The basis for this is a Wiener process $\mathcal{W}(t)$, which is a continuous-time stochastic process. For $0 \leq s < t \leq T$ the increments

$$\mathcal{W}(t) - \mathcal{W}(s) \sim \sqrt{t-s}\,\mathcal{N}(0,\sigma) \tag{3.26}$$

are independent and normal distributed with a variance σ. Since continuous processes cannot be represented within a digital environment, the process has been discretized with a step size $\mathrm{d}t = T/N$, yielding

$$d\mathcal{W}(t) \sim \mathcal{N}(0, \sigma\sqrt{\mathrm{d}t}) \tag{3.27}$$

as increments, where N denotes the number of samples. Since the trajectories created here are later used as reference currents, they have to be within the current constraints of the motor. These are quadratic constraints of the form

$$\left(\widetilde{i}_{\mathrm{d}}^{*}\right)^2 + \left(\widetilde{i}_{\mathrm{q}}^{*}\right)^2 \leq 1. \tag{3.28}$$

Furthermore, the $\widetilde{i}_{\mathrm{d}}^{*}$ component is limited to the negative range so that the motor is operating in flux-weakening mode if $\widetilde{i}_{\mathrm{d}}^{*} < 0$ and armature control range if $\widetilde{i}_{\mathrm{d}}^{*} = 0$ [6].
The compliance with these constraints is taken into account during the incrementation of the Wiener process, so that the current vector does not exceed its limit but remains within the negative $\widetilde{i}_{\mathrm{d}}$ half plane. Figure 3.3 shows the space of reference trajectories.

Each trajectory is given a random starting point within the said half-plane and a randomly chosen variance between $[1 \cdot 10^{-3}, 1 \cdot 10^{-1}]$, with which the stochastic process starts. In order to create a balanced coverage over the half-plane and thus informative examples, the probability density of trajectories are tracked via density estimation-based state-space coverage acceleration (DESSCA) [46]. At random times, based on a DESSCA sample, reference steps are applied to obtain more realistic drive trajectories and to cover operating points that have not occurred frequently. An example reference trajectory for the current components is shown in Fig. 3.4.

A total of 101000 trajectories for each current component were generated for this work. The training set includes 100000 examples with which the parameters are to be learned. 500 examples each are used for validation and evaluation respectively. Validation is performed

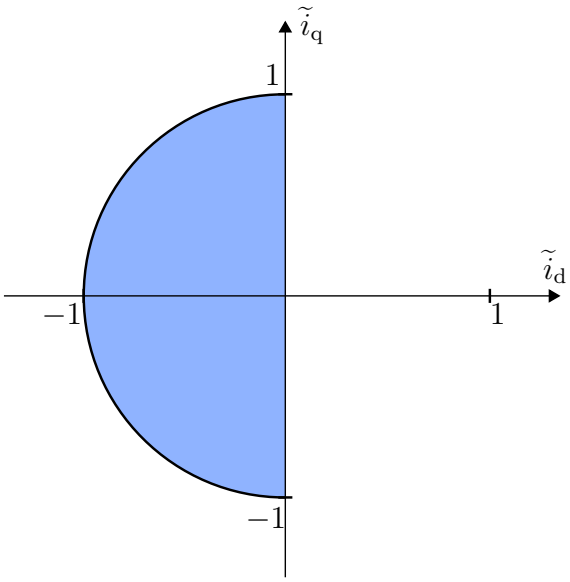


Fig. 3.3: Illustration of the reference space.

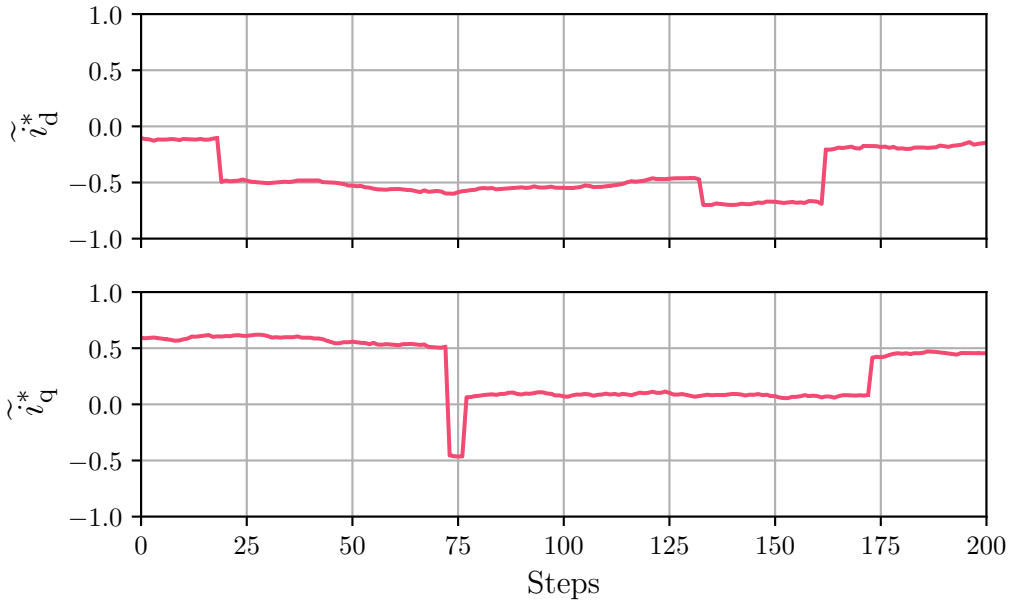


Fig. 3.4: Example reference trajectories.

after a certain number of episodes in order to measure performance. To prevent overfitting on the training data, early stopping can be performed based on the validation performance. The evaluation takes place after the training is completed. The performance achieved is the measure by which the systems are compared.

3.4 Hyperparameter Optimization

Hyperparameters are those parameters that affect the actual learning process. In terms of a NN, those can be, e.g., the number of layers and the neurons they contain, the activation function of the layers but also the optimizer and the learning rate with which the gradient update is performed. In contrast, the individual weights within the layers are not referred to as hyperparameters, as these are learned within the learning process.
Finding the parameter set that leads to the best performance at a particular setup is a complex problem, since there are countless possible combinations. This problem is solved by the process of hyperparameter optimization (HPO). There are different approaches such as grid search, random search, gradient-based optimization, evolutionary algorithms, etc. In this work the `optuna` framework [47] is used, which uses a Bayesian optimization. It is often used for black-box models such as NNs, since it shows better results than the previously listed methods [48][49]. The goal of the Bayesian optimization is to build a probabilistic model of a function, which maps the hyperparameters to actual objective, i.e., evaluating a NN. When selecting the parameters, the potential gain in information is always taken into account. The aim is to reduce uncertainties in the probabilistic model (i.e., exploration) but also to evaluate those combinations which are close to the optimum (i.e., exploitation). The parameter set that are most likely to improve the probabilistic model are then iteratively selected and evaluated. Based on the result of the evaluation, the probabilistic model is updated and a new parameter set is determined. The process of the HPO is shown in Fig. 3.5.

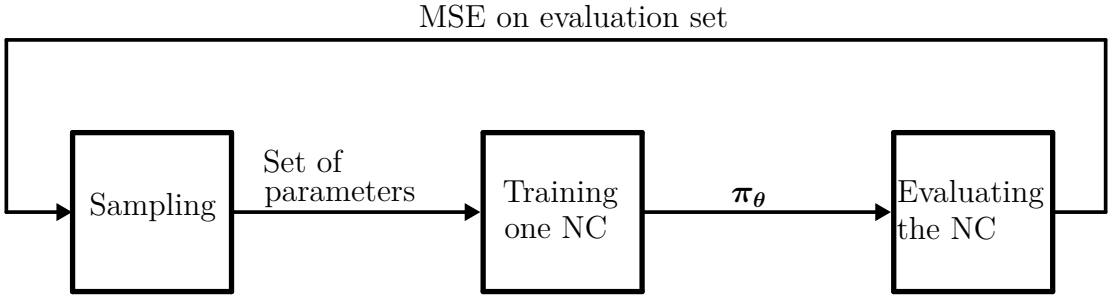


Fig. 3.5: Block diagram of the HPO.

4 Experimental Evaluation

In this chapter, the experiments are evaluated. First, a performance comparison between the approaches is made with respect to the current control. Secondly, the HPO for the NC approach is intended to find more about the different dependencies within the system. Finally, the current control and system identification is presented in a proof-of-concept and the results are presented and discussed for different applications.

4.1 GEM Drive Simulation

For the experiments the GEM toolbox is used to model a drive system. The environment includes the DC supply, the reference generator, and the PMSM that is to be simulated. The voltage supply is set to a constant value of $u_{\mathrm{DC}} = 400\,\mathrm{V}$. The data sets presented in section Section 3.3 are stored in the reference generator. For training, trajectories are thus randomly sampled from the training dataset. For the evaluation, the corresponding trajectories can be run one after the other. Each trajectory has a length of 201 steps. With a sampling interval of $\tau = 0.1\,\mathrm{ms}$, this corresponds to a duration of $T = 20\,\mathrm{ms}$. The controlled motor plant is a PMSM with the parameters as listed in Table 4.1.

Tab. 4.1: Drive parameters of the PMSM.

Name	Symbol	Value
Winding resistance	R_{s}	$15\,\mathrm{m\Omega}$
Series inductance	L_{d}	$0.37\,\mathrm{mH}$
Cross inductance	L_{q}	$1.2\,\mathrm{mH}$
Pole pair number	p	3
Permanent magnet flux	Ψ_{p}	$65.6\,\mathrm{mVs}$
Moment of inertia of the rotor	J_{rotor}	$0.039\,\mathrm{kg/m^2}$
Current limit of the motor	i_{max}	$400\,\mathrm{A}$
Voltage limit of the motor	u_{max}	$450\,\mathrm{V}$
Angular velocity limitation	$\omega_{\mathrm{el,max}}$	$400\pi\,\mathrm{s^{-1}}$

The angular velocity of the PMSM is kept constant at $\omega_{\mathrm{el}} = 100\pi\,\mathrm{s^{-1}}$.

4.2 Current Control of a PMSM

In this section, the current control capability of the different approaches are compared. First, the setups will be explained in more detail before the results are presented. Subsequently, the results will be put into context and will be discussed.

4.2.1 Baselines

The consider baselines are the classical FOC and the RL agent, which have already been presented in Section 2.1.5 and Section 2.3.2.2. The following sections describe the settings of these.

4.2.1.1 FOC with PI controller

The basics for controlling the PMSM by the FOC with a PI controller have been given, but now it should be specified how to set the parameters of the controller. The controller design usually takes place in the Laplace domain. For this purpose, the Laplace transformation is applied to the controller policy Eq. (2.12), which then yields

$$U(s) = K_\mathrm{p} \frac{T_\mathrm{N} s + 1}{T_\mathrm{N} s} E(s). \tag{4.1}$$

The transfer function of the PI controller is therefore given by

$$G_\mathrm{PI}(s) = \frac{U(s)}{E(s)} = K_\mathrm{p} \frac{T_\mathrm{N} s + 1}{T_\mathrm{N} s} E(s). \tag{4.2}$$

Besides the controller, the controlled system must also be transformed into the Laplace domain. This results in two separate transfer functions for the d and q component

$$G_\mathrm{d,q}(s) = \frac{1}{R_\mathrm{s}} \frac{1}{1 + \tau_\mathrm{d,q} s} \tag{4.3}$$

with

$$\tau_\mathrm{d,q} = \frac{L_\mathrm{d,q}}{R_\mathrm{s}}. \tag{4.4}$$

Both components of the control loop can be described as PT1 elements in the Laplace domain.

There are various requirements for the design in terms of steady-state and dynamic behavior, such as stationary accuracy, control behavior and disturbance behavior. The design procedure is done using the transfer function and its time constants. One method

of controller design for electric drives is the symmetric optimum (SO) [4]. According to that the general settings are

$$\begin{aligned} T_\mathrm{N} &= a^2 T_\sigma, \qquad T_\mathrm{N} > T_\sigma \\ K_\mathrm{p} &= \frac{1}{a}\frac{T_1}{V_\mathrm{s} T_\sigma}. \end{aligned} \tag{4.5}$$

The time constants T_1 and T_σ are the large and small delay constants of the drive system, respectively. V_s specifies the gain of the controlled system. The parameter a will be explained in more detail later on. Since the behavior is equivalent to a PT1-element there is only one time delay. The discretization results in a delay of τ and the control results in a further delay of 0.5τ. These dead times can be represented in the Laplace domain by another PT1-element with a time constant of 1.5τ. If the elements of the controlled system are combined, the result is a PT2-element with the following parameters:

$$\begin{aligned} T_1 &= \tau_\mathrm{d,q}, \\ T_\sigma &= 1.5\tau, \\ V_\mathrm{s} &= \frac{1}{R_\mathrm{s}}. \end{aligned} \tag{4.6}$$

This results in the following control law after the transformation into the time domain:

$$u(t) = \frac{1}{a}\frac{L_\mathrm{d,q}}{1.5\tau} e(t) + \frac{L_\mathrm{d,q}}{a^3 (1.5\tau)^2} \int_0^t e\left(\tilde{t}\right) \mathrm{d}\tilde{t}. \tag{4.7}$$

The parameter a decisively determines the dynamics of the system. The smaller the value, the faster the system is controlled, but at the expense of greater overshoot and lower noise damping. Due to the constraint in Eq. (4.5), a must always be chosen greater than one. In this work $a = 4$ was chosen, because it is the default setting within the implementation of the GEM toolbox. A summary of the control loop in the Laplace domain is shown in Fig. 4.1.

Since the implementation of the controller takes place within a digital environment, the control law given in Eq. (2.12) must be discretized. The continuous variables are defined at time points k, which are multiples of the sampling interval τ. This yields

$$u[k] = K_\mathrm{p} e[k] + \frac{K_\mathrm{p}}{T_\mathrm{N}} \sum_{i=0}^{k} e[k]\tau \tag{4.8}$$

where $e[k]$ is the discrete control error. In addition to the now discrete P component, the integral of the I component is approximated by a sum, where the control errors are multiplied by the sampling interval and summed up.

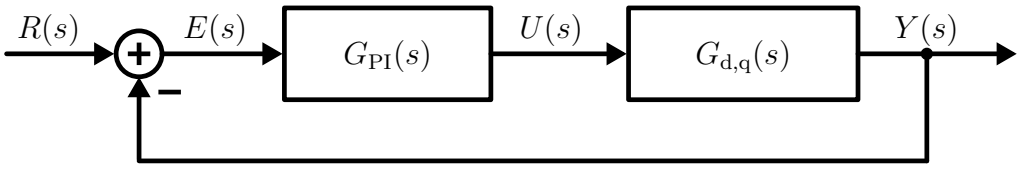


Fig. 4.1: Contol loop within the Laplace domain.

The FOC by the PI controller is summarized in Fig. 4.2.

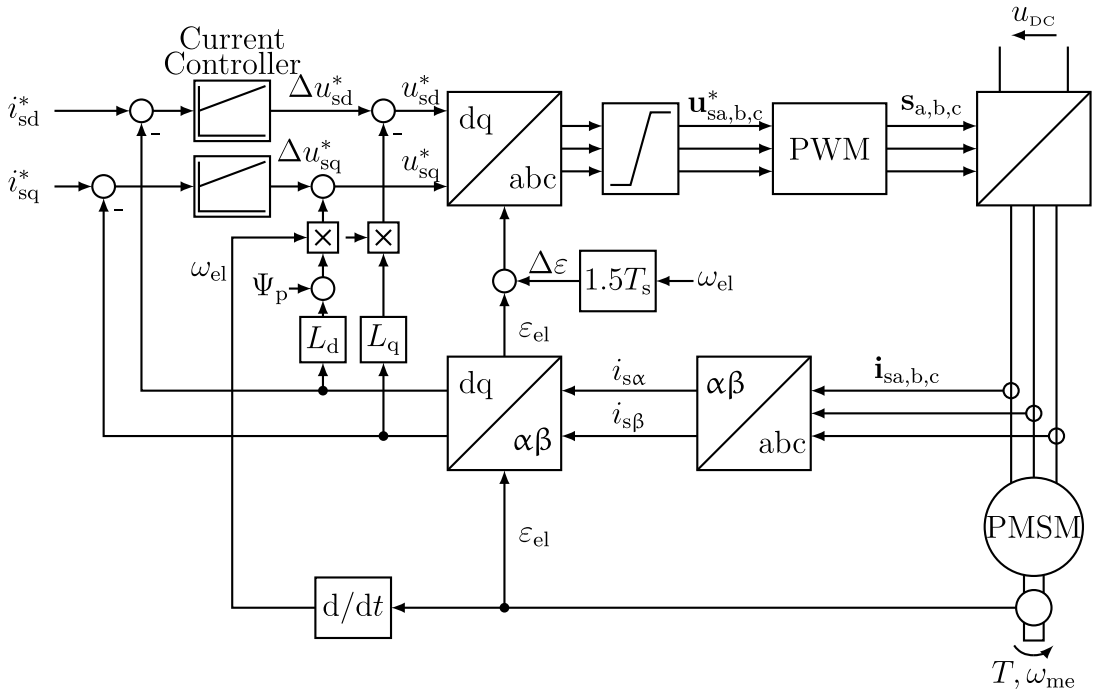


Fig. 4.2: Block diagram of the current control via PI controller of the PMSM.

4.2.1.2 RL Agent

The second baseline for current control is the DDPG agent. It provides the comparison to a data-driven approach. For this work a slightly adapted version of the example available in the GEM toolbox is used. In the following the setup is specified in more detail.

The actor network receives the state vector as input. This includes the normalized reference currents and actual currents:

$$\boldsymbol{x} = \left(\widetilde{i}_\mathrm{d} \;\; \widetilde{i}_\mathrm{q} \;\; \widetilde{i}^*_\mathrm{d} \;\; \widetilde{i}^*_\mathrm{q}\right)^\top . \tag{4.9}$$

The output is the respective action $\boldsymbol{u}$. The parameters of the actor network are listed in Table 4.2.

Tab. 4.2: Parameters of the actor network.

Name	Symbol	Value
Number of hidden layer	$l_{\mathrm{h,actor}}$	3
Number of neurons per hidden layer	$n_{\mathrm{h,actor}}$	64
Hidden layer activation function	$\boldsymbol{g}_{\mathrm{h,actor}}$	LeakyReLU ($\alpha = 0.1$)
Output layer activation function	$\boldsymbol{g}_{\mathrm{out,actor}}$	tanh

The critic network receives besides the state vector also the chosen action of the actor. The output is a scalar value. Parameters of the critic network can be seen in Table 4.3.

Tab. 4.3: Parameters of the critic network.

Name	Symbol	Value
Number of hidden layer	$l_{\mathrm{h,critic}}$	4
Number of neurons per hidden layer	$n_{\mathrm{h,critic}}$	128
Hidden layer activation function	$\boldsymbol{g}_{\mathrm{h,critic}}$	LeakyReLU ($\alpha = 0.1$)
Output layer activation function	$\boldsymbol{g}_{\mathrm{out,critic}}$	linear

The target networks are copies of the networks presented above.
For the agent to learn, it needs a reward function that depends on the total current

$$\widetilde{i}_{\mathrm{total}} = \sqrt{\left(\widetilde{i}_{\mathrm{d}}\right)^2 + \left(\widetilde{i}_{\mathrm{q}}\right)^2}. \tag{4.10}$$

The reward per time step is given by

$$r = \begin{cases} 1 - \left(\widetilde{i}_{\mathrm{total}} - \frac{i_{\mathrm{n}}}{i_{\mathrm{lim}}}\right)\left(1 - \frac{i_{\mathrm{n}}}{i_{\mathrm{lim}}}\right)^{-1} \cdot 1/10 - 1/10, & \text{if } \widetilde{i}_{\mathrm{total}} > \frac{i_{\mathrm{n}}}{i_{\mathrm{lim}}} \\ (2 - r_{\mathrm{d}} - r_{\mathrm{q}}) \cdot 1/20, & \text{else} \\ -1, & \text{if episode terminates early} \end{cases} \tag{4.11}$$

with

$$r_\mathrm{d} = \left(\sqrt{\frac{|\widetilde{i}_\mathrm{d}^* - \widetilde{i}_\mathrm{d}|}{2}} + \left(\frac{\widetilde{i}_\mathrm{d}^* - \widetilde{i}_\mathrm{d}}{2} \right)^2 \right) \cdot 1/2$$
$$r_\mathrm{q} = \left(\sqrt{\frac{|\widetilde{i}_\mathrm{q}^* - \widetilde{i}_\mathrm{q}|}{2}} + \left(\frac{\widetilde{i}_\mathrm{q}^* - \widetilde{i}_\mathrm{q}}{2} \right)^2 \right) \cdot 1/2.$$

In this case, $i_\mathrm{n} = 240\,\mathrm{A}$ is defined as nominal current and $i_\mathrm{lim} = 400\,\mathrm{A}$ as current limit through the respective motor. To speed up the learning process of the agent, the reward per step is chosen on an interval of $[0, 1]$.

Further settings for the training of the DDPG agent can be taken from Table 4.4.

Tab. 4.4: Parameters of the DDPG agent, regarding the training.

Name	Symbol	Value
Optimizer actor	-	Adam
Learning rate actor	η_actor	$5 \cdot 10^{-6}$
Optimizer critic	-	Adam
Learning rate critic	η_critic	$5 \cdot 10^{-4}$
Action noise	$\mathcal{N}$	Ornstein-Uhlenbeck (see Eq. (A.5))
Discount factor	γ	0.99
Update rate	δ	0.25
Data buffer	$\lvert\mathcal{D}\rvert$	$5 \cdot 10^5$
Batchsize	$\lvert\mathcal{D}_\mathrm{b}\rvert$	256

The algorithm trains for 800000 steps, which, assuming full episode length of 201 steps, can be extrapolated to almost 4000 episodes within one training. In each episode a weight update takes place.

4.2.2 Neural Controller

For this experiment, the setup described in Section 3.2.2, is used. Therefore, perfect model knowledge within the IPM is assumed. Additionally, the IPM is extended by further components. Those components are multiple transformations into the different coordinate systems, which are present in the drive system implemented by GEM, but cannot be represented by the ODE of the PMSM alone. Consequently, the IPM is extended with those transformations.

The NC receives the augmented feature vector and outputs the control action. For the further setup, the parameters of the NC are specified in Table 4.5.

Tab. 4.5: Parameters of the NC.

Name	Symbol	Value
Number of hidden layer	l_h	1
Number of neurons per hidden layer	n_h	128
Hidden layer activation function	$\boldsymbol{g}_\mathrm{h}$	ReLU
Output layer activation function	$\boldsymbol{g}_\mathrm{out}$	$\mathrm{clip}(-1,1)$
Composed loss	c_1	1/10
	c_2	50
	c_3	1
Optimizer	-	Adam
Learning rate	η	$3 \cdot 10^{-3}$
Batchsize	$\lvert\mathcal{D}_\mathrm{b}\rvert$	1024

The training runs for 200 episodes. After each episode the parameters are optimized on the basis of a batch containing 1024 trajectories. In total, 200 parameter updates take place over the duration of the training, whereby the NC is trained over a total of 204800 trajectories.

4.2.3 Results

The 500 evaluation trajectories, which were not shown to any data-driven control approach during training, were used to compare the approaches. The benchmark metric for all of them is the MSE between the normalized current and reference values. It should be noted that since the FOC with PI controller is a purely deterministic approach, therefore only one run is performed. Since the data-driven approaches are trained with a random initialization of parameters, it follows that by using gradient descent and given the non-convex cost landscape, the approaches end up in different local minimas. To better identify this inadequacy of the SGD as a cost function optimizer, nine evaluations are conducted. Based on the variance of these evaluations, a better prediction of the performance can be made. The results obtained can be seen in Table 4.6.

Among the nine experiments conducted, the NC was unable to learn a successful policy in only one experiment. This run is considered a failure and is not taken into account for the statistical evaluations. In the median, the NC has the best performance with a value of $16.194 \cdot 10^{-3}$, the FOC with PI controller is ranked behind it with a value of $23.672 \cdot 10^{-3}$. The MSE of the DDPG agent is $32.044 \cdot 10^{-3}$ and is thus almost twice as high as of the NC. On average over the runs, the NC has also the lead with a value of $16.526 \cdot 10^{-3}$. Here, the second place is occupied by the PI controller with a value of $23.672 \cdot 10^{-3}$. The worst performance was achieved by the DDPG agent with a value of $32.225 \cdot 10^{-3}$. In Fig. 4.3, the results are illustrated for the DDPG agent and the NC.

Tab. 4.6: Results of the experiments. Shown here is the MSE on the evaluation set, where 9 runs were performed for the data-driven algorithms. Bold font highlights the best score across the control approaches.

	MSE (10^{-3} p.u.)		
Run	PI controller	DDPG agent	NC
1	23.672	29.993	16.118
2	-	38.908	16.262
3	-	26.533	16.004
4	-	32.044	15.845
5	-	25.136	17.528
6	-	25.742	135.018
7	-	34.717	16.074
8	-	44.327	17.463
9	-	32.627	16.877
Median	23.672	32.044	**16.194**
Mean	23.672	32.225	**16.526**

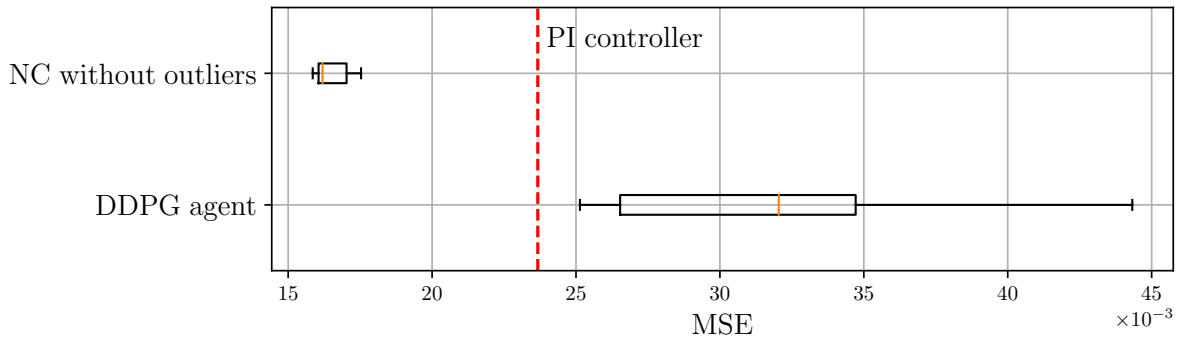


Fig. 4.3: Graphical presentation of the results, with the data-driven approaches shown as boxplots. The result of the PI controller is indicated as a vertical line.

4.2.4 Discussion

Based on Table 4.6, it can be concluded that the results of the NC are significantly better than those of the baselines. It should be emphasized that the variance within the runs considered is also significantly lower than with the DDPG agent (see Fig. 4.3). This can be explained mainly by the fact that the NC has perfect model knowledge and can therefore be learned in a more systematic way, leading to the lower variance in the results.

Selected evaluation trajectories will be used here to illustrate the behavior of the NC in comparison to the other control algorithms (see Fig. 4.4).

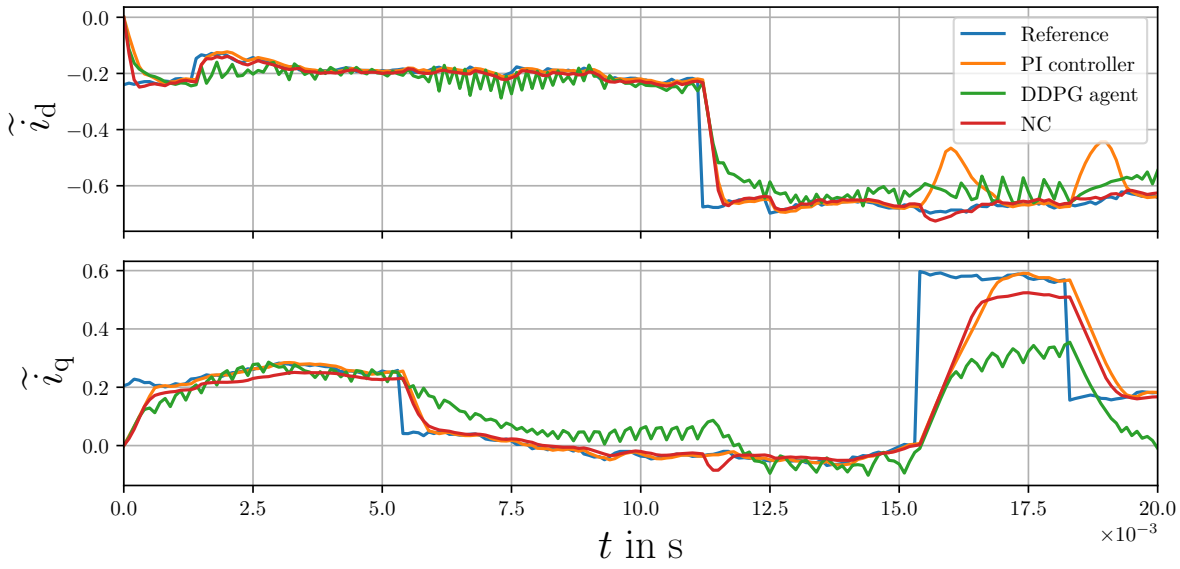


Fig. 4.4: Comparison between the different control algorithms.

The reference trajectory for both current components is shown in blue. It is clearly visible that all control algorithms start from the origin. When comparing the trajectories, the one of the DDPG agent stands out. This agent seems to have clear difficulties in following the reference, so that an alternating pattern of overestimating and underestimating the reference emerges. Steps in the d-component can still be followed relatively well, as can be seen in the plot at a time of $t \approx 11\,\mathrm{ms}$. By steps in the q-component, as at $t \approx 15\,\mathrm{ms}$, the following seems to be much more difficult for the agent. The orange trajectories of the PI controller and the red trajectories of the NC show a similar good behavior in the stationary case. However, the NC seems to systematically underestimate the q-component of the current, visible here in the range of $t \in [1, 5]\,\mathrm{ms}$. Both controllers also react similarly well to steps regarding the tracking control. In the case of the PI controller, the d-component is affected by steps in the q-component, as can be seen at $t \approx 50\,\mathrm{ms}$ and $t \approx 18\,\mathrm{ms}$. A dependence can also be observed for NC. In this case, steps in the d-component lead to adjustments of the q-component as in $t \approx 11\,\mathrm{ms}$, but they are significantly smaller.

This strong dependence of the PI controller between the d-component and the q-component leads to problems when already operating close to the constraint. This is shown in Fig. 4.5. At about $t \approx 8\,\mathrm{ms}$, the d-component has a value of -0.7. Due to a step in the q-component, the current of the d-component is further reduced so that it exceeds the current limit at $t \approx 15\,\mathrm{ms}$. At this point the GEM environment stops the evaluation, which leads to the fact that the evaluation can be described as incomplete. Of the total 500 evaluation trajectories, the current limit was exceeded 42 times by the FOC with PI controller. The DDPG agent, which also works in the GEM environment, does not exceed the current limits within the evaluations. The NC also remains within the current limits for all evaluation trajectories.

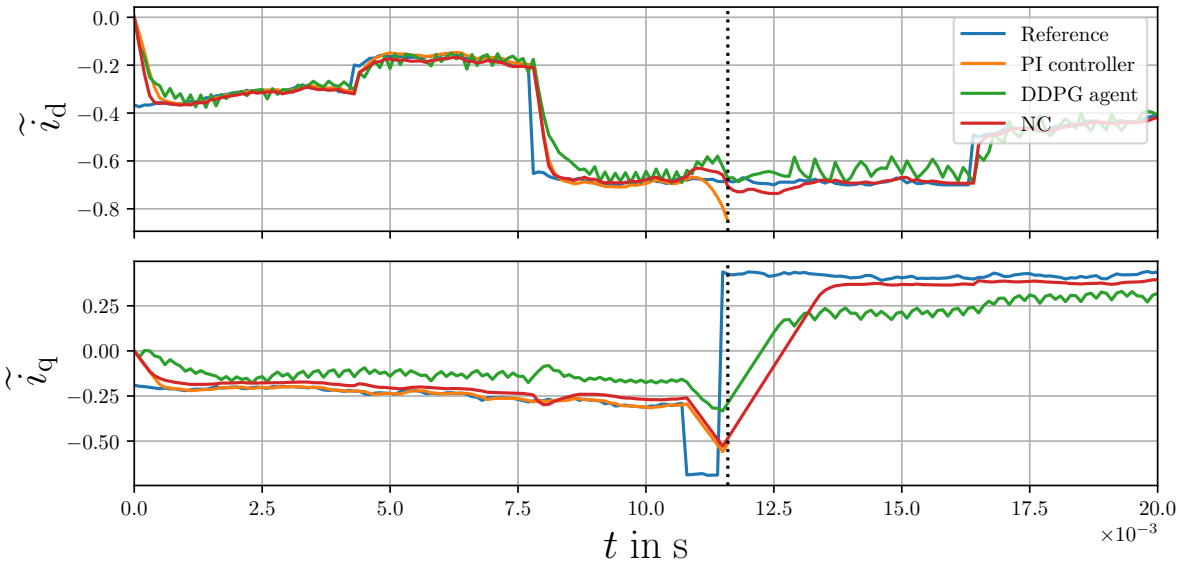


Fig. 4.5: Comparison of the control algorithms, whereby the FOC is terminated prematurely due to the violation of the constraints. The time of termination is marked by the dashed line.

Since the NC is also a data-driven approach, poor initialization can also lead to the failure of a training, which clearly happened to run 6. This cannot be prevented, since the problem originates from the gradient descent. Due to the non-convex cost landscape, the system remains in a local minimum, which the optimizer cannot get out of.
It should be noted that the DDPG agent remained significantly below expectations. Possible errors could be badly configured hyperparameters. Poorly set learning rates can cause NNs to be stuck in poor local minimas. Furthermore, the update rate can also lead to the fact that the target networks were adapted too quickly. Another point is the choice of the update rate, which is relatively high here with 0.25. This could have led to a too fast change of the target networks, which in turn could have brought instabilities into the training. Whether it was a bad setting of hyperparameters or a bug in the implementation, which caused the DDPG agent to produce such poor results, cannot be said without a more detailed analysis of the setup. However, this was outside the scope of this thesis.

Nevertheless, the results of the NC show outstanding results that outperform the baselines. Of course, this is always to be seen under the consideration that a perfect model knowledge about the drive system was assumed here.

4.3 HPO of the Neural Controller

This section deals with the HPO of the NC. First the setup will be explained and the selectable parameters will be listed. Afterwards the results of the HPO will be presented and discussed.

4.3.1 Setup

To evaluate the current control capabilities of the NC, again the setup described in Section 3.2.2, is used. Therefore perfect model knowledge is assumed within the IPM and the additional components of the drive systems are considered. The training data of 100000 trajectories is used and the final evaluation is performed on the evaluation set of 500 trajectories, equivalent to the previous experiment.

The choice of hyperparameters for the NC and the optimizer is up to the sampling method of the HPO. The following parameters are taken into account:

Tab. 4.7: Parameters for the HPO

Description	Range	Datatype
Number of hidden layers	$\{2, \dots, 5\}$	Integer
Number of neurons within the hidden layers	$\{16, \dots, 128\}$	Integer
Activation function of the hidden layers	Tanh ReLU LeakyReLU ELU	Categorical
Negative slope of the LeakyReLU	$[0.01, 0.3]$	Float
Optimizer	Adam SGD	Categorical
Learning rate of the optimizer	$[1 \cdot 10^{-5}, 1 \cdot 10^{-1}]$	Float

The framework used here for the HPO is `optuna`. For one HPO, a study is created, which contains a specified number of trials. In this work 650 trials are performed. In each of these trials, nine NC are trained. A training lasts 200 episodes, after each episode a parameter update with a batchsize of 1024 is performed. Each of these NC is then evaluate and the median of these nine experiments is reported to the sampling algorithm. Based on the result, new parameters are sampled, with which the next trial is then started.

4.3.2 Results

Based on the given hyperparameters, the HPO was able to achieve the best result for the NC with 5 hidden layers, using the ReLU activation function. Training was done with the Adam optimizer with a learning rate of $9.521 \cdot 10^{-3}$. The trial achieved in the median a MSE of $15.605 \cdot 10^{-3}$ on the evaluation set.

The course of the HPO can be seen in Fig. 4.6, where the median MSE obtained is plotted for each trial. The line shows the course of the best result.

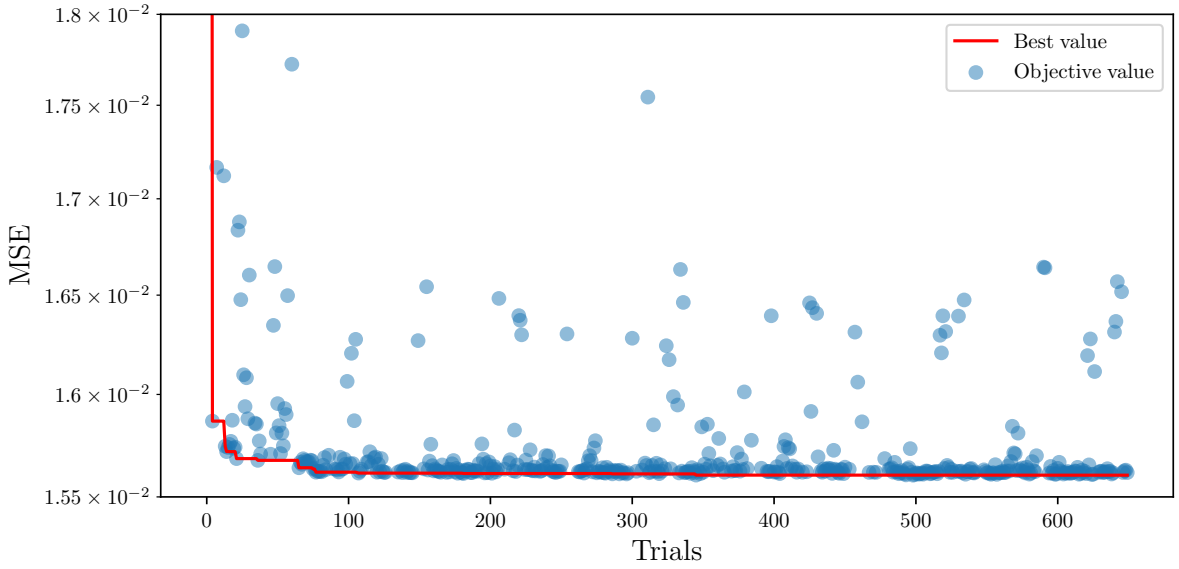


Fig. 4.6: Results of trials plotted over the course of the HPO. The circles indicate the achieved MSE of each trial on the evaluation set. The curve marks the best result over the duration of the HPO.

The trend shows that rapid convergence has been achieved. After only about 100 trials, the best value drops only slightly. The y-axis indicates that the plot has been zoomed in strongly to make deviations visible (see Fig. A.2). The majority of the objective values lie in the range close to the best value for the respective trial. This distribution indicates that a local minimum has been reached and no major improvements are to be expected.

The following Fig. 4.7 shows the total sampled parameters over the course of the HPO. In each plot, the median MSE is plotted over the selected parameters. The color of a point indicates the time of the sample. In the following, the plots will be discussed in sequential order.

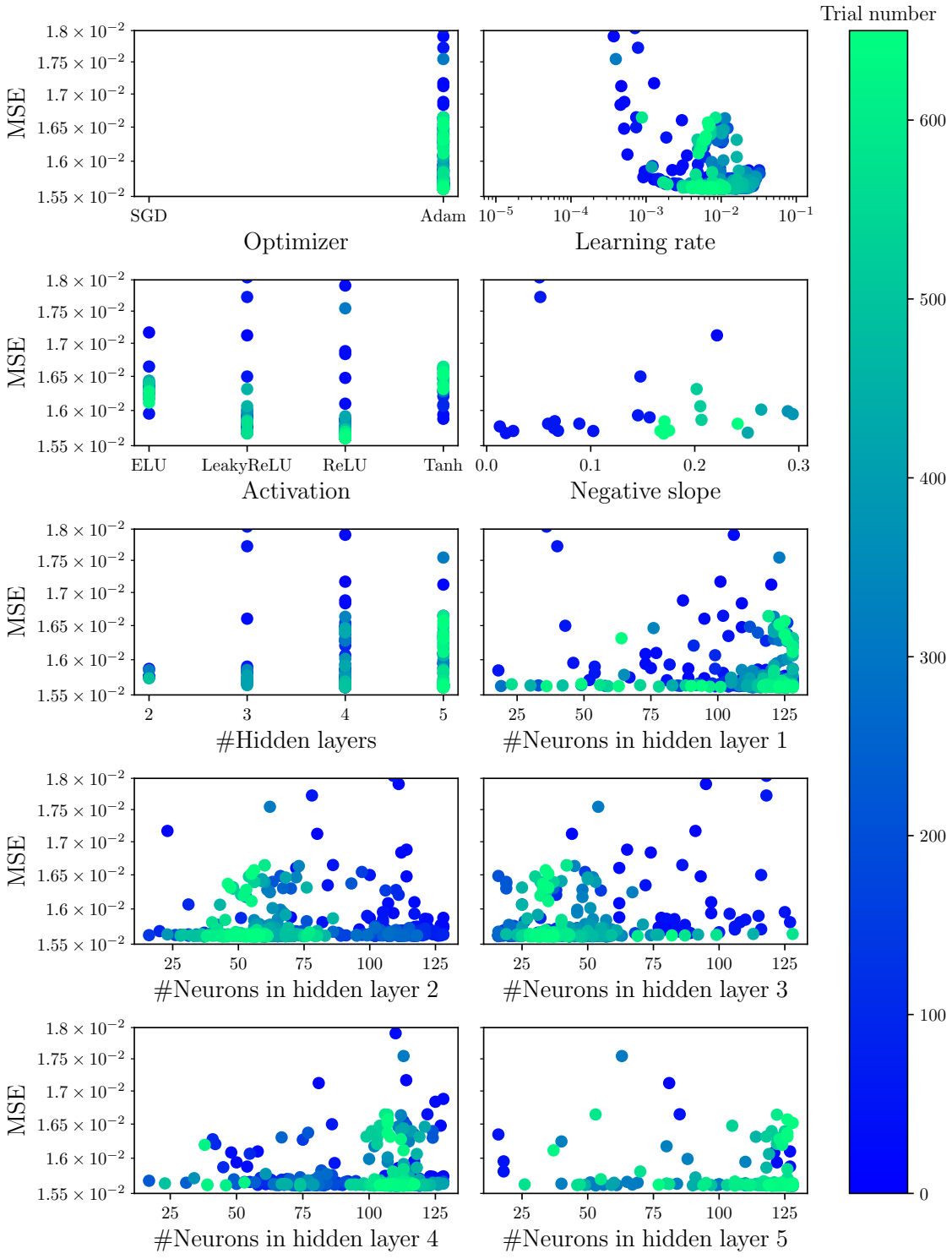


Fig. 4.7: This plot shows the different parameters that were sampled during the HPO in dependence of the achieved MSE of the respective constellation. Each trial is marked with a dot and the time of the trial is indicated by the color.

When sampling the optimizer, the Adam algorithm seems to achieve significantly better results than the SGD. In the case of the learning rate, the range around $1 \cdot 10^{-2}$ is strongly sampled, whereby the learning rate was reduced in the course of the HPO. In the activation function, the ReLU and LeakyReLU are ahead of the ELU and the tanh. But all functions seem to be sampled quite frequently, as the colors indicate. The negative slope of the LeakyReLU was sampled fairly uniformly, showing a trend toward 0.3 over the course of the HPO. The choice of the number of hidden layers, as well as the number of neurons within the layers, yields a similarly good result as can be seen in the plots. The best performance of each model size, in terms of the number of hidden layers, is listed in Table 4.8.

Tab. 4.8: Performance comparison regarding model size.

# Hidden layers	MSE (10^{-3} p.u.)
2	15.734
3	15.637
4	15.605
5	**15.605**

The results indicate that the best results of the respective sizes are relatively close to each other. Between the smallest and the largest model, there is a difference in performance of about $1.3 \cdot 10^{-4}$. It is also noticeable that the model size of 4 and 5 hidden layers are close to the same result.

4.3.3 Discussion

When considering the results of the NC from the last experiment, it can first be stated that the results can be improved even further with the selection of suitable hyperparameters. The MSE could thus be reduced from $16.194 \cdot 10^{-3}$ to $15.605 \cdot 10^{-3}$.

The progression of the objective values shown in Fig. 4.6 suggests that the cost landscape moved into a local minima fairly early. Samples are mainly made with slightly changed parameters of the best trial, which leads to the results being close to the best value as seen in Fig. 4.6. Larger jumps are not observed over the course of the best value of the HPO, which suggests a relatively flat cost landscape.

Looking at the overall parameters, the choice of optimizer is the most significant, with Adam being clearly preferred over SGD. The Adam algorithm can be seen as an extension, which additionally uses the squared gradients to adjust the learning rate of individual parameters. Furthermore, in the parameter update, the momentum is used by a moving average over the gradients, while in the case of SGD only the gradient is used. The learning rate also seems to be an important parameter, as it was sampled in a narrow range. Especially since it can be observed here that in the course of the HPO the values

move further into the said range. The activation function also seems to have only a marginal effect on the performance of the NC. All functions were sampled equally, with ReLU slightly ahead of LeakyReLU in terms of the median MSE. There is also no clear trend in the number of hidden layers. Although the HPO sampled 5 layers more frequently at the end, the results of 3 and 4 layers seem to be only minimally worse. In applications, smaller meshes should be preferred due to faster execution time. In general, no precise statement can be made about the suggested number of neurons within the hidden layer. The samples are evenly distributed and are all close to the best value, so a lower number should also be preferred here when it comes to implementation.

In summary, no unusual parameter dependence could be determined. With an appropriate optimizer and a learning rate in the said range, reproducible results can be achieved that are outperforming than the baselines considered in the previous experiment. Large networks with many neurons are not significantly superior if the last percentages in the performance are not important.

4.4 Current Control and System Identification

This experiment is intended to provide proof of concept for simultaneous current control and system identification of a PMSM. For this purpose, the setup presented in Section 3.2.3 will first be specified in more detail. This is followed by the presentation of the results and a final discussion.

4.4.1 Setup

The setup used within this experiment is divided into three components: the EP, the IPM and the NC. Each component will be considered individually below.

External Plant

The EP is the system that is to be controlled. In this work it is a PMSM, which is simulated by the GEM toolbox and specified with the parameters from Section 4.1. A problem with the later training of the NC and parameter estimation is the serial execution of the simulations, therefore several simulations cannot be executed in parallel. Hence, it is necessary to store the trajectories of the simulation temporarily in order to create a batch. Another problem is the different duration of the simulations due to the current limitation. That is, if the NC let the currents exceed the limits, this can lead to the termination of the simulation. Aborted simulations and therefore too short trajectories are therefore zero-padded to have identical lengths.

Internal Plant Model

The parameters to be identified are now specified in the IPM. Based on the knowledge that the plant PMSM can be modeled with a linear system model, the matrices $\boldsymbol{A}_\phi$ and $\boldsymbol{B}_\phi$ as well as the vector $\boldsymbol{e}_\phi$ are set up here according to Section 3.2.3. Two experiments are to be conducted in the process. The first one considers partial model knowledge, therefore

only two parameters, $\hat{L}_\mathrm{d}$ and $\hat{L}_\mathrm{q}$, are identified, keeping the exact values of R_s and Ψ_p. In another experiment, all four parameters, $\hat{L}_\mathrm{d}$, $\hat{L}_\mathrm{q}$, $\hat{R}_\mathrm{s}$ and $\hat{\Psi}_\mathrm{p}$ are to be identified. Hence, the parameter identification experiments use the expert knowledge about the structure and number of parameters of the EP.
For the initialization of the learnable parameters a value is sampled from a uniform distribution between $[0, 1]$. Due to the fact that the true values are positive constants (see Table 4.1) and cannot be negative in reality, the learnable parameters are additionally placed in the exponential function, yielding

$$\boldsymbol{A}_{\boldsymbol{\phi}} = \begin{pmatrix} -\frac{\exp(\hat{R}_\mathrm{s})}{\exp(\hat{L}_\mathrm{d})} & \omega_\mathrm{el}\frac{\exp(\hat{L})_\mathrm{q}}{\exp(\hat{L}_\mathrm{d})} \\ -\omega_\mathrm{el}\frac{\exp(\hat{L}_\mathrm{d})}{\exp(\hat{L}_\mathrm{q})} & -\frac{\exp(\hat{R}_\mathrm{s})}{\exp(\hat{L}_\mathrm{q})} \end{pmatrix}, \qquad \boldsymbol{B}_{\boldsymbol{\phi}} = \begin{pmatrix} \frac{1}{\exp(\hat{L}_\mathrm{d})} & 0 \\ 0 & \frac{1}{\exp(\hat{L}_\mathrm{q})} \end{pmatrix}$$

and

$$\boldsymbol{e}_{\boldsymbol{\phi}} = \begin{pmatrix} 0 \\ -\frac{\exp(\hat{\Psi}_\mathrm{p})}{\exp(\hat{L}_\mathrm{q})}\omega_\mathrm{el} \end{pmatrix}.$$

This is another aspect where expert knowledge is brought to the IPM. On the one hand, this ensures that the values are always greater than 0. On the other hand, the exponential function accelerates a faster ascent or descent of the parameters in gradient-based optimization. Identical to the current control, the IPM is also extended here by the components of the GEM drive system.

Neural Controller

The NC is realized with the same parameters as in the current control experiment in Section 4.2. Therefore the parametrization can be taken from Table 4.5. The setups should be kept the same to allow a fair comparison between the experiments.

4.4.2 Training

The training is divided into two phases. First, a warm-up is performed, where initially only the learnable parameters of the IPM are identified. The NC is left randomly initialized and performs random control actions for both the EP and the IPM. The identification is then based on the loss between EP and IPM, obtained by the 2 trajectories. The optimizer of the learnable parameters of the IPM is Adam with a learning rate of $3 \cdot 10^{-1}$. The warm-up consists of 50 episodes, where after each episode a batch of 1024 trajectories is used for the update.
In the second phase, not only the IPM is adapted but also the NC is trained on the basis

of the IPM estimate and the reference, as explained in Section 3.2.3. Based on the loss of these two trajectories, the parameters of the NC are updated with a second optimizer. This is again Adam with a learning rate of $3 \cdot 10^{-2}$. This phase of the training includes 200 episodes, again using a batch of 1024 trajectories to optimize in the first step the IPM and afterwards the NC. Therefore, the parameter updates with respect to the NC are identical to the current control from Section 4.2. This allows the comparison of the results.

4.4.3 Results

Since this experiment is composed of both current control and system identification, the results are presented separately. First, the results of the current control of the NC with parameter identification will be compared with the results of the NC with perfect model knowledge from Section 4.2.3. Subsequently, the parameter identification of the individual experiments are examined in more detail, and the best runs in each case are considered in detail.

Current Control Comparison

For the results, the evaluation dataset is used which includes 500 episodes. The metric is again the MSE between normalized current and reference trajectory on the EP. The experiments are repeated nine times to compensate for the shortcoming of the SGD, as discussed above. The results of the experiments are shown in Table 4.9.

Tab. 4.9: Results of the experiments. Shown here is the MSE on the evaluation set, where 9 runs were performed for each setup. Bold font highlights the best score across the experiments.

	MSE (10^{-3} p.u.)		
Run	Perfect model knowledge	Identifying 2 parameters	Identifying 4 parameters
1	16.118	15.729	15.714
2	16.262	138.983	149.979
3	16.004	15.726	150.159
4	15.845	15.722	150.159
5	17.528	15.813	150.159
6	135.018	137.859	15.760
7	16.074	15.845	15.827
8	17.463	15.781	149.979
9	16.877	15.817	150.016
Median	16.194	15.776	**15.760**
Mean	16.526	15.781	**15.767**

In the left column, the results of the NC with perfect model knowledge are taken from Section 4.2.3. The NC was not always finding a policy successfully. In the first experiment, i.e., the identification of 2 parameters, run 2 and 6 failed. When identifying 4 parameters, i.e., the whole system, only in run 1, 6 and 7 the NC was able to learn a policy.
If the results are only considered for those runs where the NC was able to learn a policy, then the identification of 4 parameters leads to the best results with $15.760 \cdot 10^{-3}$ in terms of the median and $15.767 \cdot 10^{-3}$ in terms of the mean. Then follows the experiment with partial model knowledge. Here, the median is $15.776 \cdot 10^{-3}$ and the mean is $15.781 \cdot 10^{-3}$. The results of the NC based on the perfect model knowledge only reach a median of $16.194 \cdot 10^{-3}$ and mean of $16.526 \cdot 10^{-3}$.

Partial Model Knowledge

First, for all 9 runs of the experiment, the ratio between the learned parameters and the ground truth is reported. The results after completion of the warm-up phase, as well as after training, are summarized in the table Table 4.10.

Tab. 4.10: Ratio between parameter estimates and ground truth after initial warm-up phase and after completed training phase.

	Parameter ratio			
	After warm-up		After training	
Run	$\hat{L}_\mathrm{d}/L_\mathrm{d}$	$\hat{L}_\mathrm{q}/L_\mathrm{q}$	$\hat{L}_\mathrm{d}/L_\mathrm{d}$	$\hat{L}_\mathrm{q}/L_\mathrm{q}$
1	1.126	1.122	1.000	1.000
2	1.115	1.098	1.000	1.000
3	1.073	0.919	1.000	1.000
4	1.098	0.885	1.002	1.000
5	1.000	1.043	0.977	0.998
6	0.993	1.019	1.000	1.000
7	0.999	1.096	1.000	1.000
8	1.124	0.859	0.999	1.000
9	0.895	0.989	1.000	1.000

After the warm-up phase, all runs are already close to the parameters to be identified. After training, except for a stronger deviation at run 6 for $\hat{L}_\mathrm{d}$, both inductances were identified exactly to the 4th decimal place for almost all runs.
In order to get a better insight into the parameter identification, run 1 is examined in more detail, as it provided the best results. In Fig. 4.8, two plots are shown. The upper one displays the progression of the losses in logarithmic scale over the training. The blue curve corresponds to the loss between the IPM to the EP. Due to the warm-up, the orange curve starts at episode 50. This is the loss between estimated trajectory of the IPM and reference trajectory. The lower plot shows the average trajectory length per episode. Both losses converge quickly, where the MSE has already reached a value of $1 \cdot 10^{-3}$ after

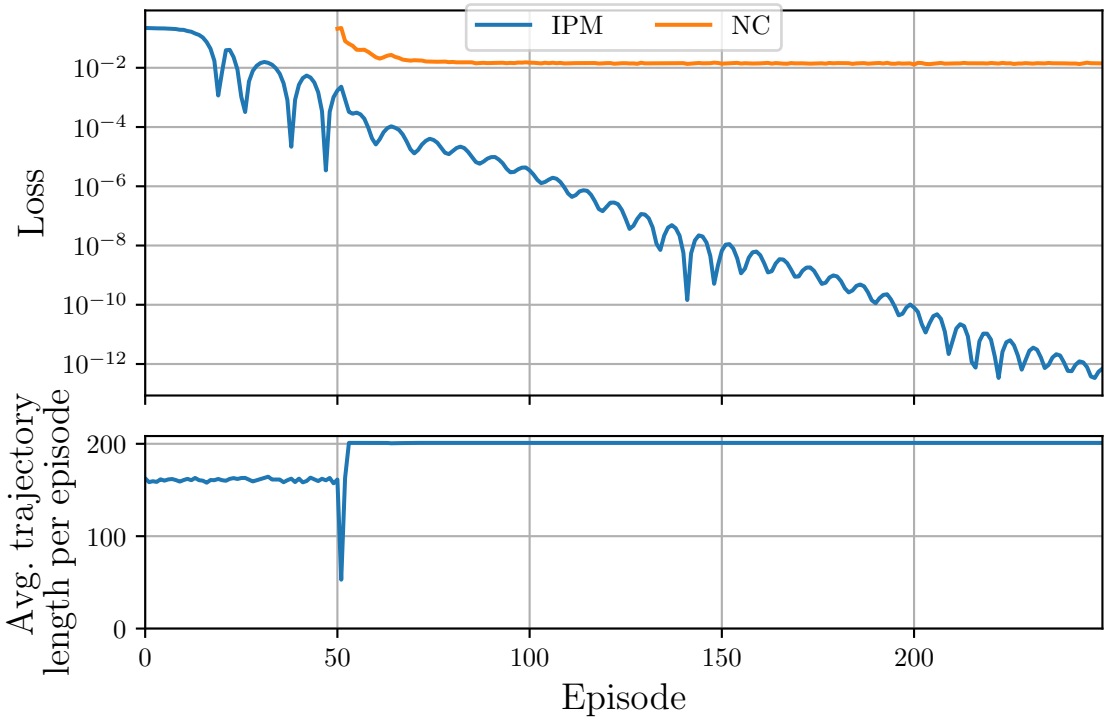


Fig. 4.8: Training progression of the 2 parameter identification for run 1. The upper plot shows the training progression of the IPM and NC in terms of the losses in logarithmic scale. In the lower plot shows the progression of the averaged trajectory length of the EP.

completion of the warm-up. This then decreases steadily to a final value of approximately $5 \cdot 10^{-13}$. The course of the CL reaches convergence at around the 90th episode with a value of approximately $14 \cdot 10^{-3}$. Around this value the trajectory fluctuates for the rest of the training. The average trajectory curve initially oscillates around a value of about 160. After the warm-up, there is a sharp drop to an average length of 50, which increases again sharply in the next episodes and already reaches a value of 201 in the 55th episode. The rest of the training uses full trajectory lengths for the optimization. In Fig. 4.9, the learnable parameters are plotted over the course of the training.

In both plots a convergence of the learnable parameters to the ground truth can be seen. The parameter $\hat{L}_\mathrm{d}$ has a more damped trajectory after the warm-up, while $\hat{L}_\mathrm{q}$ oscillates damped around the actual value for the whole training. The deviation between prediction and the ground truth after 250 episodes the relative error (see Eq. (A.7)) is $\nu_{L_\mathrm{d}} = 4.77 \cdot 10^{-6}$ and $\nu_{L_\mathrm{q}} = 9.83 \cdot 10^{-6}$.

Four Parameter Identification

Firstly, the quotient between the learned parameters and the ground truth after each run of this experiment are summarized. A perfect identification corresponds to a value of 1. The results are shown in Table 4.11.

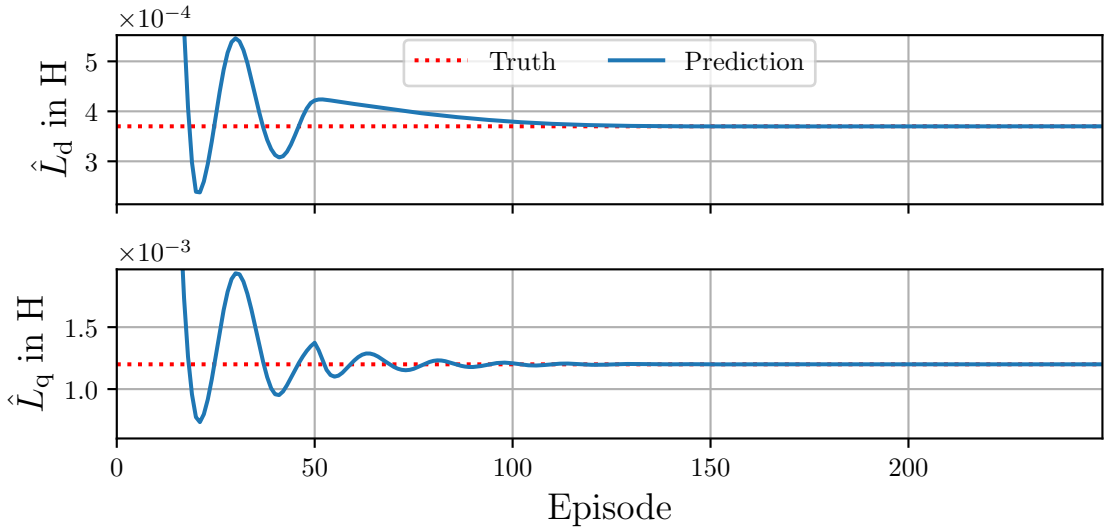


Fig. 4.9: Progression of the IPM's two learned parameters over the course of the training.

Tab. 4.11: Ratio between parameter estimates and ground truth after initial warm-up phase and after completed training phase.

	Parameter ratio							
	After warm-up				After training			
Run	$\hat{L}_\mathrm{d}/L_\mathrm{d}$	$\hat{L}_\mathrm{q}/L_\mathrm{q}$	$\hat{R}_\mathrm{s}/R_\mathrm{s}$	$\hat{\Psi}_\mathrm{p}/\Psi_\mathrm{p}$	$\hat{L}_\mathrm{d}/L_\mathrm{d}$	$\hat{L}_\mathrm{q}/L_\mathrm{q}$	$\hat{R}_\mathrm{s}/R_\mathrm{s}$	$\hat{\Psi}_\mathrm{p}/\Psi_\mathrm{p}$
1	1.084	1.147	1.365	0.995	1.013	1.002	1.616	1.062
2	0.000	0.024	0.946	0.006	1.006	0.004	0.112	0.116
3	0.032	0.001	0.042	0.037	1.004	0.001	0.166	0.172
4	0.036	0.003	0.001	0.004	0.023	0.002	0.000	0.042
5	0.044	0.046	0.012	0.027	1.011	0.002	0.068	0.070
6	0.620	0.206	0.035	0.113	1.116	0.989	0.240	4.508
7	0.805	0.939	6.325	1.091	1.000	1.002	10.546	0.985
8	0.016	0.009	0.001	0.007	1.111	0.000	0.007	0.007
9	0.050	0.236	0.018	0.056	1.017	1.130	0.321	0.558

After completion of the warm-up phase, it can be seen that the parameter ratios are significantly worse than in the previous experiment. Exceptions to this are run 1 and 7, which are closest to 1 in terms of the parameter ratio. After the training the parameter $\hat{L}_\mathrm{d}$ is closely approximated in most runs. For the other parameters, it depends strongly on the run whether a ratio is close to 1.
Since run one has the best result, this one will be looked at more closely here. In Fig. 4.10, two plots summarizing the training progression. The upper plot shows the course of the

two losses in logarithmic scale. In the lower plot the average trajectory length per episode can be seen.

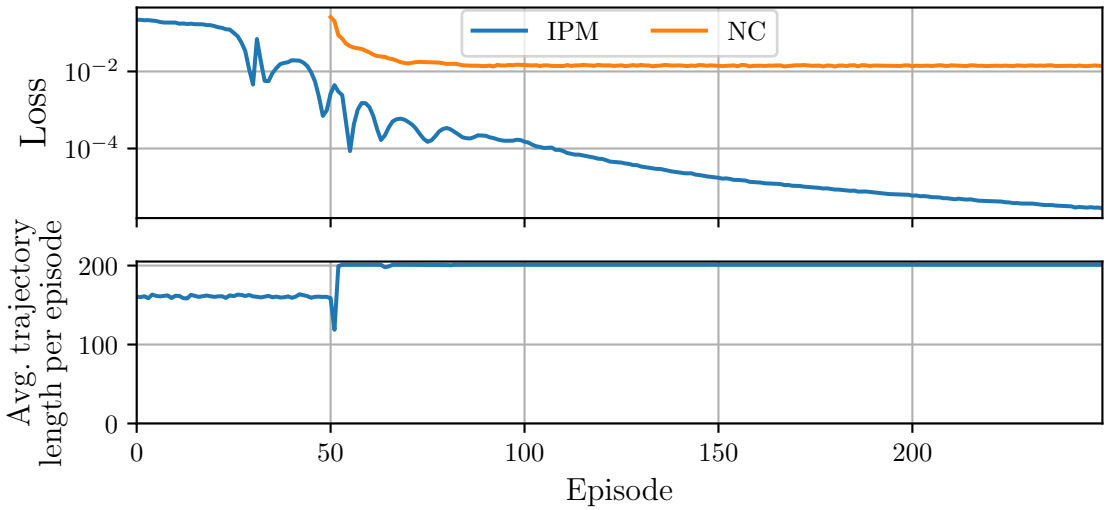


Fig. 4.10: Training progression of the 4 parameter identification for run 1. The upper plot shows the training progression of the IPM and NC in terms of the losses in logarithmic scale. In the lower plot shows the progression of the averaged trajectory length of the EP.

The loss of the IPM already reaches a value of $2.6 \cdot 10^{-3}$ after the warm-up. Thereafter, it steadily decreases to a final value of approximately $2.5 \cdot 10^{-6}$. The loss of the NC starts at episode 50 due to the warm-up. It quickly converges towards a value of $14 \cdot 10^{-3}$ after 35 episodes. After that, it no longer changes significantly during the rest of the training. The averaged length of the trajectories starts at about 160 and then oscillates slightly around this value for the time of the warm-up. After the warm-up, a drop to a value of 120 can be seen, but in the next step it is already close to the maximum averaged length again. It stays at a maximum of 201 for the rest of the training after episode 85. In Fig. 4.11, the progressions of the learned parameters over the training are shown. The blue curve indicates the respective course of the parameter, with the red dotted line marking the value of the true parameter.

The parameter $\hat{L}_\mathrm{d}$ moves quickly towards the true value, so that it is already close to after the warm-up phase. Subsequently, it slowly converges against the true value. The estimation of $\hat{L}_\mathrm{q}$ shows strong overshoots at the beginning of the training, which then decrease over time and finally lead to a good identification of the exact parameter at the end of the training. The parameter $\hat{R}_\mathrm{s}$ shows at the beginning also a fast change into the direction of the true parameter. After completion of the warm-up, there is a deviation of approx. $5 \cdot 10^{-3}\Omega$, which can then hardly be reduced during the rest of the training. For the last parameter $\hat{\Psi}_\mathrm{p}$ a strong convergence in the warm-up phase can be seen again. After the warm-up, this parameter reaches an almost exact approximation of the true value with $65.9 \cdot 10^{-3}$Vs. In the further course of the training the value is slightly underestimated

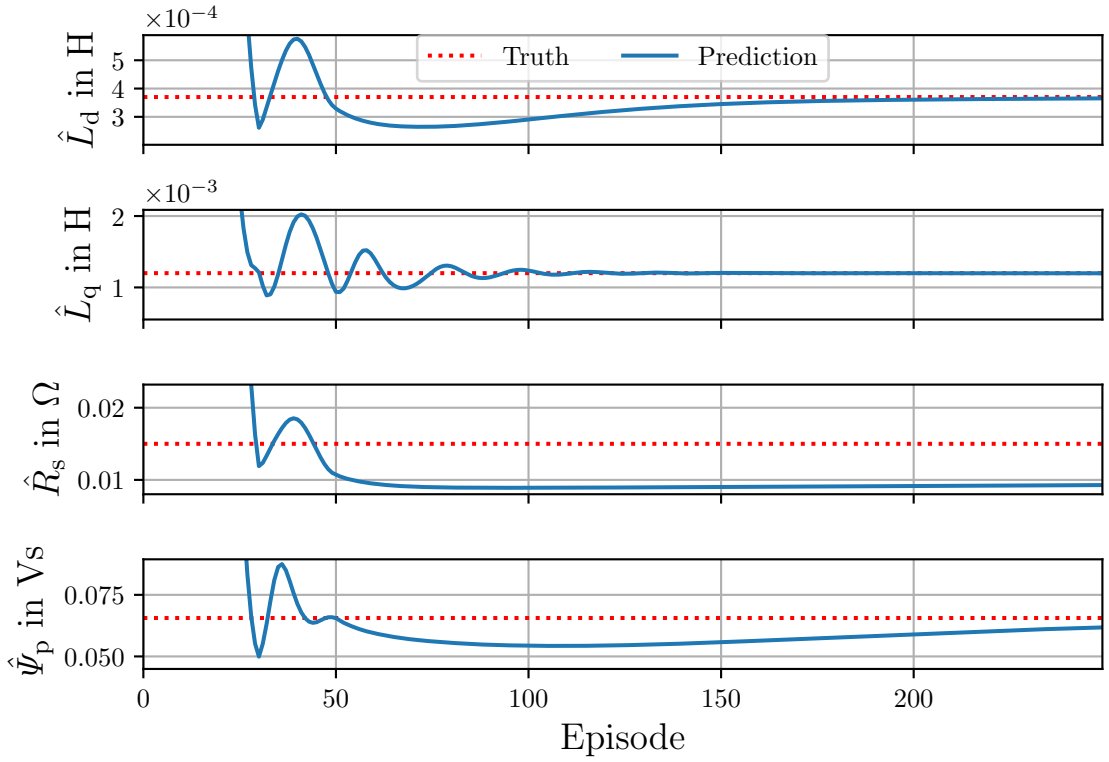


Fig. 4.11: The learnable parameters of the IPM for run 1 are shown over the course of the training.

and only towards the end of the training it approaches the true value again. The relative error of the learned parameters at the end of the training are as follows: $\nu_{L_\mathrm{d}} = 16.9 \cdot 10^{-3}$, $\nu_{L_\mathrm{q}} = 115.2 \cdot 10^{-3}$, $\nu_{R_\mathrm{s}} = 2.1133$ and $\nu_{\Psi_\mathrm{p}} = 792.7 \cdot 10^{-3}$.

4.4.4 Discussion

First of all, the results in Table 4.9 show that simultaneous current control and system identification works. Thus, the experiments provide the desired proof of concept.
Considering the runs where the NC was able to learn a policy, the results of the 2 and 4 parameter identification are better than the experiment with perfect model knowledge. This implies that parameter identification has a positive impact on the learning of the policy. One explanation is that because of the slight changes in IPM due to the learning parameters, the NC must still be able to find an appropriate policy. The control is more challenging due to the model inaccuracies, which in turn leads to a more robust control.

When comparing the parameter ratios in Table 4.10 and Table 4.10 of the two experiments with parameter identification, it is also noticeable that in the case of the 2 parameters the IPM was more accurate compared to the identification of the 4 parameters. However,

a more accurate IPM did not lead to better current control as can be seen in Table 4.9. This also supports the hypothesis that learning a good policy does not require perfect model knowledge.

Another interesting conclusion is that even perfect model knowledge can lead to unsuccessful training. According to Table 4.9, this was 1 run, which failed to learn a policy. The fact that no policy could be learned even with perfect model knowledge indicates that in this approach the initialization of the parameters of the NC already plays a role in learning a successful policy. However, this is a known problem with data-driven approaches and does not occur exclusively with the use of NODEs.

Another case that shows the dependence of the parameters is run 2 in the case of the partially known model. In Table 4.10 it is clear that at the time after the warm-up there was sufficient proximity to the parameters of the EP. Also, compared to the other runs, these show similar variations in parameter ratio and were able to learn a successful policy. Nevertheless, in this case the NC was not able to learn current control. This shows once again that the initialization of the NC plays an important role in policy learning.
Furthermore, it also follows that even if parameters are identified almost perfectly over the course of the training, the NC is not able to change its policy significantly. Again, the problem lies in the nature of the cost landscape and the use of gradient decent als cost function optimizer. Unfortunately, there is no simple solution to prevent this problem. The run can only be aborted and repeated or the NC has to be reinitialized.

However, it turns out that the number of failed policies is significantly higher when the entire system is identified than in the experiment with partial or full model knowledge. This shows that learning an appropriate policy depends on the IPM as well. If the parameters of the IPM are not well identified, the NC cannot learn a policy. Particularly important is the phase after the warm-up. This is evident in Table 4.11, where runs 1 and 7 in particular show a proximity of the learned parameters to the true parameters after the warm-up. This means that if no sufficient proximity to the true parameters has been found up to this point, the NC will start learning an undesirable policy. Since the NC is then optimized on an IPM which is an inadequate representation of the EP, an undesirable policy is learned. This policy then in turn leads to the trajectories of the EP aborting prematurely because current limits of the GEM environment are exceeded. Due to shorter and therefore also non-informative trajectories, the IPM can only slowly identify the parameters. The loss of the NC gets stuck in a local minimum, out of which the policy cannot be further improved.
This problem can be avoided by not simply using the NC with random initialization to control the EP in the warm-up phase, but by first applying a classical control approach. In this way, longer and more informative trajectories can be generated on the EP, with the help of which a faster identification of the parameters can presumably be achieved. Assuming that the parameters converge close to a good approximation of the true parameters, it can be excluded that the training of the NC fails due to wrong parameters in the IPM.

5 Summary

5.1 Summary

In this work, a novel approach to simultaneous current control and system identification of a PMSM drive system was presented. Utilizing the NODE framework, a model could be obtained from data by combining expert knowledge of plant structure with data-driven learning. The proposed algorithm achieved better results than classical FOC and a state-of-the-art RL algorithm.

At first, the approach was derived based on the NODEs. It was shown how to rewrite the NODE equation into a linear system with learnable matrices. The adjoint method was discussed, which allows to calculate memory efficient gradients within this framework. The structure of the PMSM was then exploited to integrate expert knowledge into the system matrices. The current control was then realized by implementing an NC. Based on an augmented state vector and the learned strategy, the NC can select an action to perform the current control. For the framework to run on external plants it needs a corresponding system model. In the next step, the current control was extended by system identification. Besides the current control, the entries of the matrices within are learned/identified at the same time. In order to obtain the most informative test data, attention was paid to a balanced coverage of the operating zone, while at the same time considering the limitations of the system.
For the experiments a drive system was simulated with the help of the GEM toolbox which was kept identical for all the experiments. The first experiment was a comparison of the different methods of the current control of a PMSM. In this, the NC achieved better results than the classical FOC with PI controller and the DDPG agent. In a further experiment, the structure of the NC was examined more closely. Through a HPO it was shown in which ranges the parameters should be chosen. In addition, it was shown that the results in the closer range of the parameters led to identically results. In the NC, large NNs do not achieve significantly better results than smaller ones. In the last experiment, simultaneous current control and system identification was evaluated. The results have been able to prove the concept with outstanding performance. The results of the experiments with partial model knowledge and no model knowledge achieved better results than the experiment with complete model knowledge. However, the system still lacked robustness, especially when identifying more than 2 parameters, so that many of

the runs failed. Problems that arise due to the system identification were discussed and possible solutions were provided.

Although the results of this thesis are very promising, they need to be verified on real world test benches. Nevertheless, this work provides a good foundation on which future experiments can be built upon. In the following an outlook for possible extensions will be given.

5.2 Outlook

First, the weaknesses pointed out in the presented implementation should be addressed. The biggest problem with the simultaneous current control and system identification is the transition from the warm-up phase to the actual learning of the NC. As already elaborated in the evaluation, a method should be implemented which ensures convergence of the learnable parameters. Trajectories that are as informative as possible help here. These can, i.e., be obtained by a classic controller during the warm-up phase. It should be checked in a further experiment whether the NC can be made to learn the policy again by re-initializing it even during the running training.

A HPO should also be performed for simultaneous current control and system identification. In particular, the learning rate of the optimizer for the IPM should be analyzed, since it is very important for the convergence of the IPM. In the presented implementation, however, this would take a very long time due to the implementation of the GEM toolbox, which leads directly to the next enhancement.

Much of the time that has gone into this work has been spent on bringing the systems implemented in the GEM toolbox into the framework used in this thesis. The insights gained here can be used to create a version of GEM that allows both parallel processing and support for AD. To facilitate future ML experiments, but also to exploit new research opportunities, a corresponding extension of the GEM toolbox would be beneficial.

Since these are simulative experiments based only on models of the PMSM, the next step should be to use a more complex model of the PMSM that takes into account, e.g., iron losses, magnetic saturation or temperature dependencies. Also, an implementation on the test bench would be a milestone for the NODE approach, as there are no practical implementations of the approach in the literature so far.

In the following, further enhancements for the components and the system itself will be given. The IPM can be extended to cover cross-saturation effects. Furthermore, the learnable parameters can be represented by additional NNs to incorporate non-linearities, e.g., time or temperature dependencies. In addition, an ODE solver with a fixed step size is currently used to determine the estimated trajectories. The implementation could be extended by adaptive solvers. The use of the adaptive solver would allow the possibility of an additional adjustment between error tolerance and computational effort. To achieve a faster overall system, the second call of the ODE solver could also be omitted. The

gradients that are determined by a single execution are sufficient, but then two gradients must be carried with each variable. This is a more elegant but also a more complex solution. Nevertheless, should the system run in real time in the future, this could significantly improve the runtime of the algorithm.

Appendix

A.1 Dynamical Systems

To model time-dependent processes in various fields, such as mathematics, physics, engineering, finance, etc., dynamical systems are used. The goal is to create a connection between the often highly complex system and the empirical data resulting from it. No matter where those systems are applied, they always contain one or more states, which represents a position in the related state-space [50]. The change of a state over time, then is a function of the current state $\boldsymbol{x}(t) \in \mathbb{R}^n$ with consideration of the initial state $\boldsymbol{x}(t_0)$

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), t), \qquad \boldsymbol{x}(t_0) = \boldsymbol{x}_0. \tag{A.1}$$

The Eq. (A.1) is an ODE. In simple cases, such as the function $f(\cdot)$ being linear in $\boldsymbol{x}(t)$, an analytical solution can easily be derived. However, most applications in reality don't behave linearly and must either be approximated via a linearization or by using numerical solvers.

A.2 Derivation of NODE

An intuitive derivation to the concept of NODEs will be shown. Looking at a ResNet a residual block can be described given the following relationship:

$$\boldsymbol{x}_{i+1} = f(\boldsymbol{x}_i, \boldsymbol{\theta}) + \boldsymbol{x}_i, \tag{A.2}$$

where $\boldsymbol{x}_i$ denotes the input, as well as the hidden state of the i-th layer. $\boldsymbol{x}_{i+1}$ is the summation of the output of the residual block and the residual itself. On closer examination of this equation, one recognizes a certain similarity to Euler's method

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + hf(\boldsymbol{x}_i, t_i), \tag{A.3}$$

but with a step size of $h = 1$. This leads to the conclusion, that the one residual block is a discretization of a continuos function $f(\cdot)$ via Euler's method. Assuming a ResNet with an infinite number of these blocks could be generated and the step size would approach zero, the resulting function could be framed as an ODE:

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), t, \boldsymbol{\theta}). \tag{A.4}$$

These infinitesimal small steps cause the transition from the discrete time domain to the continuous time domain. The result is the parameterizable function $f(\boldsymbol{x}(t), t, \boldsymbol{\theta})$ which defines the derivative of the hidden state $\dot{\boldsymbol{x}}(t)$. The analogy to dynamical systems can be clearly drawn by comparing it with the formula A.1. The evaluation of the function $f(\boldsymbol{x}(t), t, \boldsymbol{\theta})$ at a time t, given the hidden state $\dot{\boldsymbol{x}}(t)$, provides the dynamics in this particular point, i.e. the answer to the question, how the hidden state will change.

A.3 Equations

A.3.1 Ornstein-Uhlenbeck Prozess

$$\mathrm{d}x(t) = \Theta(\mu - x(t))\mathrm{d}t + \sigma \mathrm{d}\mathcal{W}(t), \tag{A.5}$$

where $\Theta > 0$ and $\sigma > 0$. $\mathrm{d}\mathcal{W}(t)$ is the discretized Wiener process. μ is a constant.

A.3.2 Approximation Error

The absolute error between a value x and its approximation $\hat{x}$ is defined as

$$\triangle x = |x - \hat{x}| \,. \tag{A.6}$$

The relative error is defined as

$$\nu_x = \frac{\triangle x}{|x|} = \left| \frac{x - \hat{x}}{x} \right| = \left| 1 - \frac{\hat{x}}{x} \right| . \tag{A.7}$$

A.4 Plots

A.4.1 Sigmoid

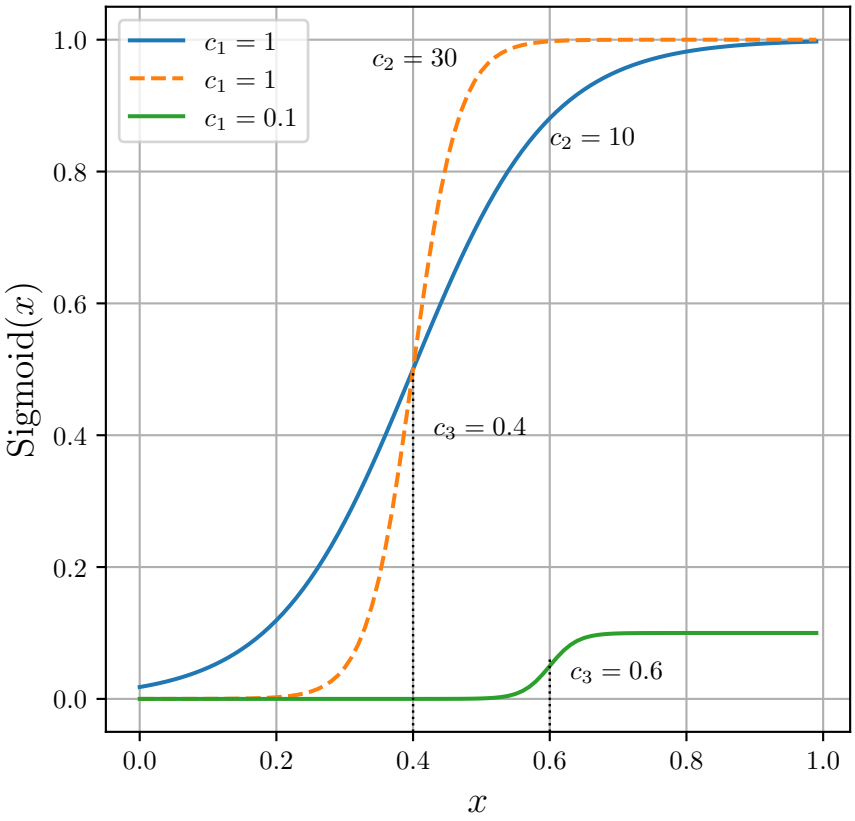


Fig. A.1: Sigmoid with different parameters.

A.4.2 HPO

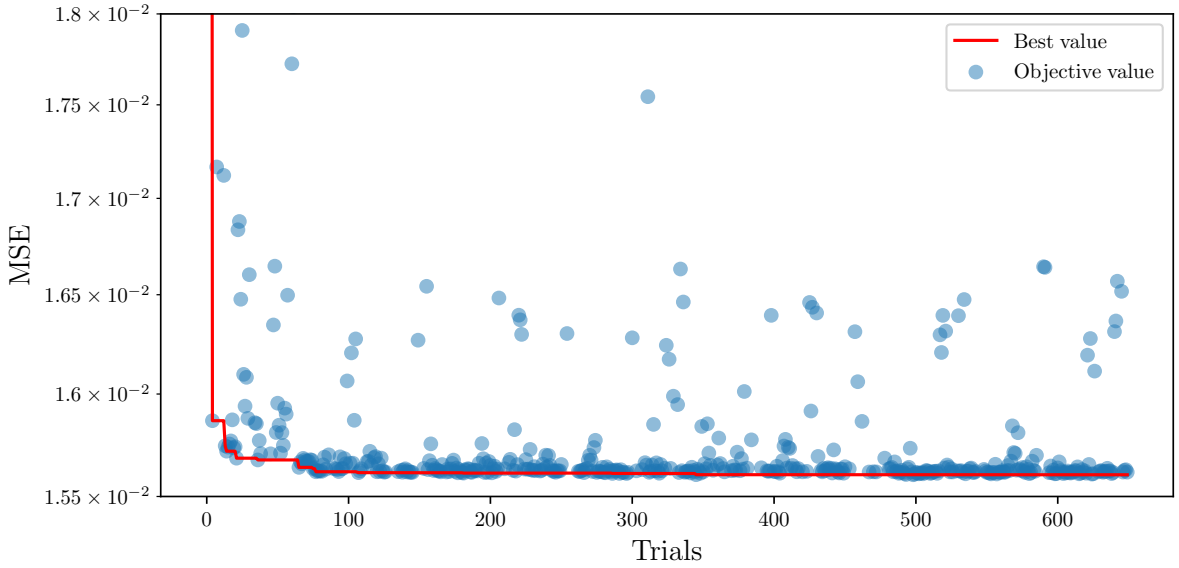


Fig. A.2: Results of trails plotted over the course of the HPO. The blue circles mark the achieved MSE of each trail on the evaluation set. The red curve marks the best result at the trail.

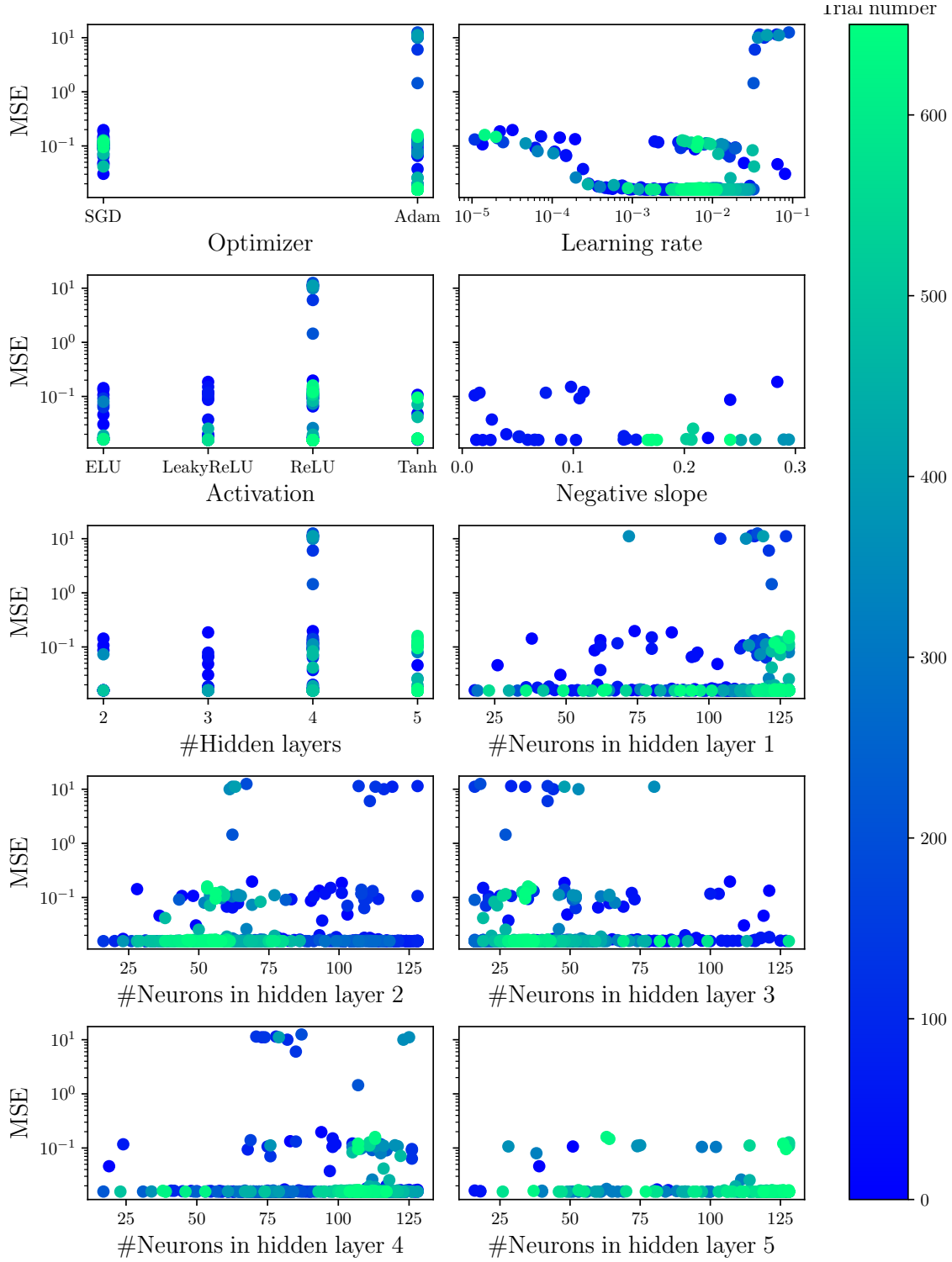


Fig. A.3: This plot shows the different parameters that were sampled during the HPO in dependence of the achieved MSE of the respective constellation. Each trial is marked with a dot and the time of the trial is indicated by the color.

A.5 Algorithms

A.5.1 DDPG Algorithm

Algorithm 1 DDPG algorithm from [32]

Randomly initialize critic network $Q(\boldsymbol{x}, \boldsymbol{u}; \boldsymbol{w})$ and actor $\mu(\boldsymbol{x}; \boldsymbol{\theta})$ with weights $\boldsymbol{w}$ and $\boldsymbol{\theta}$.
Initialize target networks Q' and μ' with weights $\boldsymbol{w}^- \leftarrow \boldsymbol{w}$, $\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$
Initialize data buffer $\mathcal{D}$
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $\boldsymbol{x}_1$
  for $t = 1, T$ **do**
    Select action $\boldsymbol{u}_k = \mu(\boldsymbol{x}_k; \boldsymbol{\theta}) + \mathcal{N}$ according to policy and exploration noise
    Apply action $\boldsymbol{u}_k$ and observe reward r_k and observe new state $\boldsymbol{x}_{k+1}$
    Store transition $(\boldsymbol{x}_k, \boldsymbol{u}_k, r_k, \boldsymbol{x}_{k+1})$ in $\mathcal{R}$
    Sample random mini-batch $\mathcal{D}_\mathrm{b}$ of N transitions $(\boldsymbol{x}_i, \boldsymbol{u}_i, r_i, \boldsymbol{x}_{i+1})$ from $\mathcal{D}$
    Set $\hat{q}_i = r_i + \gamma Q'(\boldsymbol{x}_{i+1}, \mu'(\boldsymbol{x}_i; \boldsymbol{\theta}^-); \boldsymbol{w}^-)$
    Update critic by minimizing the loss: $\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} (Q(\boldsymbol{x}_i, \boldsymbol{u}_i; \boldsymbol{w}) - \hat{q}_i)^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{u}} Q(\boldsymbol{x}_i, \boldsymbol{u}_i; \boldsymbol{w}) \nabla_{\boldsymbol{\theta}} \mu(\boldsymbol{x}_i; \boldsymbol{\theta})$$

    Update the target networks:

$$\boldsymbol{w}^- \leftarrow (1 - \delta) \boldsymbol{w}^- + \delta \boldsymbol{w}$$
$$\boldsymbol{\theta}^- \leftarrow (1 - \delta) \boldsymbol{\theta}^- + \delta \boldsymbol{\theta}$$

  end for
end for

A.5.2 Current Control and System Identification Algorithm

Algorithm 2 IPM and NC optimization.

Randomly initialize IPM and NC with weights $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$.
for episode = 1, M **do**
  for batch = 1, $|\mathcal{D}_\mathrm{b}|$ **do**
    Receive initial state $\boldsymbol{x}_0$
    for $n = 1, N$ **do**
      Apply NC to EP and obtain $\boldsymbol{x}(t_n)$
      if $\boldsymbol{x}$ terminal **then**
        break
      end if
    end for
    if $n < N$ **then**
      Add $N - n$ zero entries to $\boldsymbol{x}(t_1), \boldsymbol{x}(t_2), \ldots, \boldsymbol{x}(t_n)$
    end if
  end for
  Combine all N state trajectories to tensor
  Use **ODEsolver** to obtain $\hat{\boldsymbol{x}}(t_1), \hat{\boldsymbol{x}}(t_2), \cdots, \hat{\boldsymbol{x}}(t_N)$
  Update the IPM by minimizing the loss $\mathcal{L}(\hat{\tilde{\boldsymbol{x}}}, \tilde{\boldsymbol{x}}) = \mathbf{MSE}(\hat{\tilde{\boldsymbol{x}}}, \tilde{\boldsymbol{x}})$
  if episode > warm-up **then**
    Use **ODEsolver** with the improved IPM to obtain $\hat{\boldsymbol{x}}(t_1), \hat{\boldsymbol{x}}(t_2), \cdots, \hat{\boldsymbol{x}}(t_N)$
    Update the NC by minimizing the loss $\mathbf{CL}(\hat{\tilde{\boldsymbol{x}}}, \tilde{\boldsymbol{x}}^*) = \lambda \cdot \mathbf{MSE}(\hat{\tilde{\boldsymbol{x}}}, \tilde{\boldsymbol{x}}^*) + (1-\lambda) \cdot \mathcal{B}(\hat{\tilde{\boldsymbol{x}}})$
  end if
end for

Lists

List of Tables

List of Figures

Acronyms

AD automatic differentiation

Adam adaptive moment estimation

AI artificial intelligence

ANN artificial neural network

DDPG deep deterministic policy gradient

DE differential equation

DESSCA density estimation-based state-space coverage acceleration

EP external plant

FOC field-oriented control

GEM gym-electric-motor

HPO hyperparameter optimization

IGBT insulated-gate bipolar transistor

IPM internal plant model

IVP initial value problem

ML machine learning

MOSFET metaloxide semiconductor field-effect transistor

MSBE mean squared Bellman error

MSE mean squared error

NC neural controller

NN neural network

NODE neural ordinary differential equation

ODE ordinary differential equation

PI controller proportional-integral controller

PMSM permanent magnet synchronous motor

ReLU rectified linear unit

ResNet residual neural network

RL reinforcement learning

SGD stochastic gradient descent

SI system identification

SO symmetric optimum

VSI voltage-source inverter

Nomenclature

t	time
k	timestep
S_a, S_b, S_c	switch
s_a, s_b, s_c	switching signal
u_DC	DC supply voltage
u_0	zero component of the voltage
$u_\mathrm{ab}, u_\mathrm{bc}, u_\mathrm{ca}$	line-to-line voltage
$u_\mathrm{a}, u_\mathrm{b}, u_\mathrm{c}$	voltages for phase a,b,c
u_α, u_β	voltages in α- and β-coordinates
$u_\mathrm{d}, u_\mathrm{q}$	voltages in d- and q-coordinates
$i_\mathrm{a}, i_\mathrm{b}, i_\mathrm{c}$	currents for phase a,b,c

i_α, i_β	currents in α- and β-coordinates
$i_\mathrm{d}, i_\mathrm{q}$	currents in d- and q-coordinates
Ψ_a, Ψ_b, Ψ_c	magnetic flux for phase a,b,c
Ψ_α, Ψ_β	magnetic flux in α- and β-coordinates
Ψ_d, Ψ_q	magnetic flux in d- and q-coordinates
$\boldsymbol{T}_{23}$	transformation matrix ($\alpha\beta$- to dq-coordinates)
R_s	stator resistance
L_s	stator inductance
$L_\mathrm{d}, L_\mathrm{q}$	inductance in d- and q-coordinates
Ψ_p	permanent magnet flux
ε	rotor angle
$\boldsymbol{Q}(\varepsilon)$	rotation matrix
ω_el	electrical angular velocity
p	pole pair number
ω_me	mechanical angular velocity
J_rotor	moment of inertia of the rotor
e	control error
u	control variable
r	reference
y	system output
K_p	proportional gain
K_p	integral gain
T_N	integration time
τ, t_s	sampling interval
$\boldsymbol{x}$	feature/input/state vector
$\boldsymbol{W}$	weight matrix
$\boldsymbol{b}$	bias vector
$\boldsymbol{g}$	activation function
$\boldsymbol{y}$	output vector

$\mathcal{D}$	data buffer/dataset
f	function/model
$\boldsymbol{\theta}$, $\boldsymbol{w}$, $\boldsymbol{\phi}$	parameter set of a neural network
$\hat{\boldsymbol{y}}$	predicted output
$\mathcal{L}$	loss
$\mathcal{B}$	barrier function
η	learning rate
$\boldsymbol{u}$	action/input vector
$\mathcal{N}$	random process
μ	actor network
Q	critic network
μ'	target actor network
Q'	target critic network
r	reward
$\boldsymbol{\theta}^-$, $\boldsymbol{w}^-$	delayed parameter set
γ	discount factor
q	Q-value
$\hat{q}$	target Q-value
δ	update rate
$\boldsymbol{a}$	Lagrangian multiplier
T	terminal point
$\boldsymbol{A}$, $\boldsymbol{B}$	system matrix
$\boldsymbol{e}$	constant excitation vector
$\boldsymbol{A}_\phi, \boldsymbol{B}_\phi$	system matrix containing learnable parameters
$\boldsymbol{e}_\phi$	constant excitation vector containing learnable parameters
$\tilde{\square}$	normalization
s	limit
π	policy
$\hat{\square}$	estimated

$\square^*$	ground truth/reference
$\square^\top$	transpose
c_1, c_2, c_3	form factors
λ	weighting factor
$\mathcal{W}$	Wiener process
σ	variance
$d\mathcal{W}$	discretized Wiener process
G_{PI}	transfer function of the PI controller
$G_{\mathrm{d,q}}$	transfer function of controlled system
$\tau_{\mathrm{d,q}}$	stator time constant
T_1, T_σ	time constant
V_s	gain of the controlled system
a	adjustable parameter for the PI controller

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[3] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," *Advances in neural information processing systems*, vol. 31, 2018.

[4] D. Schröder and J. Böcker, *Elektrische antriebe-regelung von antriebssystemen*. Springer, 2009, vol. 2.

[5] R. M. Llorente, *Practical Control of Electric Machines: Model-Based Design and Simulation*. Springer Nature, 2020.

[6] J. Böcker, "Geregelte drehstromantriebe," *Universität Paderborn, New York*, vol. 170, 2009.

[7] K. Hasse, "Zum dynamischen verhalten der asynchronmaschine bei betrieb mit variabler standerfrequenz und standerspannung," *ETZ-A Bd.*, vol. 89, p. 77, 1968.

[8] F. Blaschke, "The principle of field orientation as applied to the new transvector closed-loop system for rotating-field machines," *Siemens review*, vol. 34, no. 3, pp. 217–220, 1972.

[9] V. M. Bida, D. V. Samokhvalov, and F. S. Al-Mahturi, "Pmsm vector control techniques—a survey," in *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, IEEE, 2018, pp. 577–581.

[10] P. Balakrishna, G. Book, W. Kirchgässner, M. Schenke, A. Traue, and O. Wallscheid, "Gym-electric-motor (gem): A python toolbox for the simulation of electric drive systems," *Journal of Open Source Software*, vol. 6, no. 58, p. 2498, 2021. [Online]. Available: https://doi.org/10.21105/joss.02498.

[11] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[12] A. Traue, G. Book, W. Kirchgässner, and O. Wallscheid, "Toward a reinforcement learning environment toolbox for intelligent electric motor control," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 3, pp. 919–928, 2022.

[13] K. J. Åström and P. Eykhoff, "System identification—a survey," *Automatica*, vol. 7, no. 2, pp. 123–162, 1971.

[14] R. Isermann and M. Münchhof, *Identification of dynamic systems: an introduction with applications*. Springer, 2011, vol. 85.

[15] A. L. Samuel, "Some studies in machine learning using the game of checkers. ii—recent progress," *Computer Games I*, pp. 366–400, 1988.

[16] T. M. Mitchell and T. M. Mitchell, *Machine learning*. McGraw-hill New York, 1997, vol. 1.

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[18] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[19] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[20] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia, "Incorporating second-order functional knowledge for better option pricing," *Advances in neural information processing systems*, vol. 13, 2000.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

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

[24] J. J. Moré, "The levenberg-marquardt algorithm: Implementation and theory," in *Numerical analysis*, Springer, 1978, pp. 105–116.

[25] D. M. Olsson and L. S. Nelson, "The nelder-mead simplex procedure for function minimization," *Technometrics*, vol. 17, no. 1, pp. 45–51, 1975.

[26] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[27] L. B. Rall, *Automatic differentiation: Techniques and applications*. Springer, 1981.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[29] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

[30] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[31] W. Kirchgässner, M. Schenke, O. Wallscheid, and D. Weber, *Reinforcement learning course material*, Paderborn University, 2020. [Online]. Available: https://github.com/upb-lea/reinforcement_learning_course_materials.

[32] 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," *arXiv preprint arXiv:1509.02971*, 2015.

[33] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[34] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[35] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[36] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[37] Y. Lu, A. Zhong, Q. Li, and B. Dong, "Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations," in *International Conference on Machine Learning*, PMLR, 2018, pp. 3276–3285.

[38] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, *Neural ordinary differential equations*, 2019. arXiv: 1806.07366 [cs.LG].

[39] S. Massaroli, M. Poli, F. Califano, J. Park, A. Yamashita, and H. Asama, "Optimal energy shaping via neural approximators," *SIAM Journal on Applied Dynamical Systems*, vol. 21, no. 3, pp. 2126–2147, 2022.

[40] C. Runge, "Über die numerische auflösung von differentialgleichungen," *Mathematische Annalen*, vol. 46, no. 2, pp. 167–178, 1895.

[41] W. Kutta, "Beitrag zur naherungsweisen integration totaler differentialgleichungen," *Z. Math. Phys.*, vol. 46, pp. 435–453, 1901.

[42] P. J. Prince and J. R. Dormand, "High order embedded runge-kutta formulae," *Journal of computational and applied mathematics*, vol. 7, no. 1, pp. 67–75, 1981.
[43] L. S. Pontryagin, *Mathematical theory of optimal processes*. CRC press, 1987.
[44] M. Poli, S. Massaroli, A. Yamashita, H. Asama, J. Park, and S. Ermon, "Torchdyn: Implicit models and neural numerical methods in pytorch,"
[45] R. T. Q. Chen, *Torchdiffeq*, 2018. [Online]. Available: https://github.com/rtqichen/torchdiffeq.
[46] M. Schenke and O. Wallscheid, *Improved exploring starts by kernel density estimation-based state-space coverage acceleration in reinforcement learning*, 2021. arXiv: 2105.08990 [cs.LG].
[47] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.
[48] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in neural information processing systems*, vol. 24, 2011.
[49] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.
[50] P. J. Antsaklis and A. N. Michel, *A linear systems primer*. Springer Science & Business Media, 2007.