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

Master Thesis

Data-Driven Induction Motor Control by Reinforcement Learning Including a Model Predictive Safety Layer

by

Felix Book

Supervisor: Dr.-Ing. Oliver Wallscheid, Barnabas Haucke-Korber, Maximilian Schenke

Thesis Nr.: MA 123

Filing Date: September 28, 2022

Declaration of Authorship

Abstract

For the control of an induction motor (IM), there exist different approaches to control the torque. These include the classical field-oriented controller (FOC), but also other methods such as a model predictive controller (MPC). Both control methods have in common that an exact knowledge of the system parameters is necessary. As these are often not exactly known and the system behavior is nonlinear, an optimal control of the motor is difficult to reach. Therefore, a data-driven control method is presented in this thesis. This is based on reinforcement learning (RL), which has been used for the last few years for the control of technical systems. Previous research has shown that this control method can also be used for the control of motors, such as the permanent magnet synchronous motor (PMSM).

Furthermore, a data-driven approach for estimating the magnetic flux is presented. Because an RL agent does not consider limitations when selecting an action, a system model is identified in parallel. This model is used to predict the following state, in order to decide whether an action is safe or not. As there are many suitable RL algorithms, four algorithms are compared in an extended hyperparameter optimization. It has been observed that the Proximal Policy Optimization (PPO) agent shows the best control performance. Finally, the performance of the RL controller is compared simulatively with an FOC. It is shown that the RL controller can follow the reference quite well, but cannot achieve the efficiency of the FOC.

Contents

1 Introduction

1.1 Motivation and goal of the thesis

IMs are widely used in various industrial applications and are also increasingly considered for electric vehicles. Although they require more construction space than the PMSMs that are otherwise used and have a lower peak efficiency, the construction space for the motor in electric vehicles is not a limiting factor and the efficiency is superior to that of the PMSM across the whole operating range.[1]

Classically, field-oriented approaches are applied for the control of an IM. Alternatively, it is possible to use an MPC, but both approaches require a lot of expertise and precise knowledge about the motor parameters. Furthermore, both control methods require exact knowledge about the rotor flux, which is usually an unknown motor state. In contrast, reinforcement learning controllers have been used more and more frequently in recent years. They are designed to find an approximate optimal policy in a model-free and data-driven fashion. RL controllers have not yet been used to control IMs, but the results of controlling, for example, PMSMs are promising. Therefore, the use of RL controllers in the future can reduce the required expertise and speed up the controller development process.

Other challenges arise, which need to be solved in order to implement an RL controller for this task. First, RL algorithms are designed to make decisions for a Markov decision process (MDP) with fully observable states. Due to the lack of knowledge about the rotor flux, in this case it is a partially observable Markov decision process (POMDP). Therefore, different techniques are investigated to provide the RL agent the missing information about the rotor flux. Second, the determination of the action of an RL agent do not consider safety-relevant system limitations. This is why a model-based safety layer is needed, which intercepts unsafe actions and selects a safe action instead.

The implementation of an RL agent includes a hyperparameter optimization, where all tuning parameters of the agent should be adapted to the specific task. Many different RL algorithms exist, however, the selection of an algorithm is not trivial. For this reason the hyperparameter optimization is extended to find the best RL algorithm for this task. Finally the performance of the controller will be compared with that of a classical FOC.

1.2 Structure of the thesis

This thesis is divided into eight chapters. First, the principles of the drive system and the FOC are explained. Subsequently the different RL agents, that are compared in the hyperparameter optimization, are described. In addition, some implementation aspects of RL controllers are discussed. The different techniques to obtain an estimate for the rotor flux are presented in chapter 4. The following chapter deals with the safety layer and the identification of a system model that is required for the safety layer. In chapter 6 the results of the hyperparameter optimization are presented and analyzed. The performance of the RL controller is examined and compared with the performance of an FOC in chapter 7. Finally, the results of the thesis are summarized and an outlook is given.

2 Drive System

2.1 Basics of the drive system

The drive system consists of four components. These include the voltage supply, the converter, the electric motor and the mechanical load. The voltage supply provides the converter with an ideally constant voltage. The converter also receives the control signals from the controller so that a voltage vector is switched on the inputs of the motor. The electric motor converts the electrical power into mechanical power, which drives the load. Sensors are used to measure state variables of the system, for example the speed or the currents. Figure 2.1 shows the structure of the entire drive system.

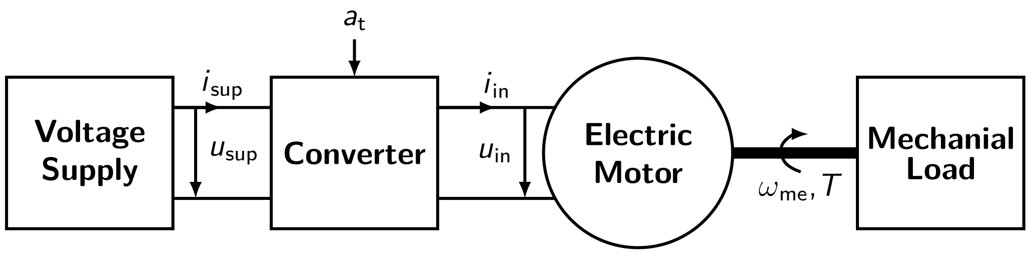


Fig. 2.1: Structure of the drive system [2]

One component of the drive system is the electric motor, in this case a three-phase induction motor. An electric motor consists of a stator and a rotor, which rotates inside or around the stator during operation. On the stator, three coil winding pairs are mounted, which are fed with a three phase AC voltage. A current flow causes each of these coils to generate a magnetic field according to Ampere's circuital law. Since the windings are symmetrical and fed with three phase AC voltages, the result is a magnetic field rotating with the frequency of the AC voltage, respectively divided by the pole pair number, which is defined as the synchronous speed.

The rotor of a squirrel cage induction motor (SCIM) is the name-giving cage of short-circuited solid windings. The rotating magnetic field generated by the stator induces an electromagnetic field in the short-circuited rotor. According to Faraday's law, the rotating magnetic field generated by the stator induces an electromagnetic field in the short-circuited rotor, resulting in a current flow in the squirrel cage. The induced voltage is calculated using Faraday's law of induction

$$u_\mathrm{i} = N \cdot \frac{\mathrm{d}\phi}{\mathrm{d}t} = \dot{\varPsi} \tag{2.1}$$

where ϕ is the magnetic flux, N is the number of turns and $\varPsi$ is the flux linkage. Due to the acting Lorentz force, the rotor starts to rotate. Laminated iron plates are also placed in the rotor to enhance the electromagnetic fields while not increasing the eddy current losses.

In motor operation, the rotor always rotates at a lower speed than the synchronous speed, which is why an IM is classified as an asynchronous motor. The rotor tries to reach the synchronous speed, but if both speeds were the same, the magnetic field would be static from the rotor's point of view. Thus no voltage would be induced in the rotor. The ratio difference between the two speeds is called slip

$$s = \frac{n_\mathrm{s} - n}{n_\mathrm{s}} \tag{2.2}$$

where n_s is the synchronous speed and n is the rotor speed. Usually the slip is in a range of a few percent.[3]

Based on the induction law in equation (2.1), the differential equations of the motor can be determined. The induced voltages on the windings are given by the clamping voltage minus the ohmic voltage drop, so that the following equations hold for the magnetic fluxes in the stator and the rotor. The rotor flux equation can be simplified since there is no external voltage applied to the rotor. The description is provided in the stator fixed $\alpha\beta$ coordinates, where the real part is the α orientation and the imaginary part is the β orientation. The superscript indicates the reference of the coordinate system of the voltages u and the currents i. R_s is the ohmic resistance of the stator. This gives the differential equations

$$\underline{\dot{\varPsi}}_\mathrm{s}^\mathrm{s} = \underline{u}_\mathrm{s}^\mathrm{s} - R_\mathrm{s}\underline{i}_\mathrm{s}^\mathrm{s} \tag{2.3}$$

$$\underline{\dot{\varPsi}}_\mathrm{r}^\mathrm{r} = \underline{u}_\mathrm{r}^\mathrm{r} - R_\mathrm{s}\underline{i}_\mathrm{r}^\mathrm{r} = -R_\mathrm{s}\underline{i}_\mathrm{r}^\mathrm{r} \tag{2.4}$$

to describe the behavior of the motor. The Park transformation is used to convert the quantities from one coordinate system to the other. In this case the angle ε is the angle of rotation between the rotor and the stator coordinate system. The transformation of the quantities is performed by the multiplication with the matrix

$$\mathbf{Q}(\varepsilon) = \begin{bmatrix} \cos\,\varepsilon & -\sin\,\varepsilon \\ \sin\,\varepsilon & \cos\,\varepsilon \end{bmatrix}. \tag{2.5}$$

Also the differential equations (2.3) and (2.4) can be transformed by this transformation into a common coordinate system, which is generally called k. As the derivative of the angle is the angular velocity ω, the term with the angular velocity between the corresponding coordinate systems results in the differential equations

$$\dot{\underline{\Psi}}_{\mathrm{s}}^{\mathrm{k}} = \mathrm{j}\omega_{\mathrm{sk}}\underline{\Psi}_{\mathrm{s}}^{\mathrm{k}} + \underline{u}_{\mathrm{s}}^{\mathrm{k}} - R_{\mathrm{s}}\underline{i}_{\mathrm{s}}^{\mathrm{k}} \tag{2.6}$$

$$\dot{\underline{\Psi}}_{\mathrm{r}}^{\mathrm{k}} = \mathrm{j}\omega_{\mathrm{rk}}\underline{\Psi}_{\mathrm{r}}^{\mathrm{k}} - R_{\mathrm{s}}\underline{i}_{\mathrm{r}}^{\mathrm{r}}. \tag{2.7}$$

A SCIM can be described by two magnetically coupled windings. Therefore, the modeling is comparable to that of a transformer, resulting in the same relationships of magnetic fluxes and currents. Figure 2.2 shows the equivalent circuit diagram of a transformer, which can also be applied to the IM. The inductances L_{s} and L_{r} are respectively the self inductance of the winding and L_{m} is the mutual inductance, which is the same for the stator flux

$$\underline{\Psi}_{\mathrm{s}} = L_{\mathrm{s}}\underline{i}_{\mathrm{s}} + L_{\mathrm{m}}\underline{i}_{\mathrm{r}} \tag{2.8}$$

and the rotor flux

$$\underline{\Psi}_{\mathrm{r}} = L_{\mathrm{m}}\underline{i}_{\mathrm{s}} + L_{\mathrm{r}}\underline{i}_{\mathrm{r}}. \tag{2.9}$$

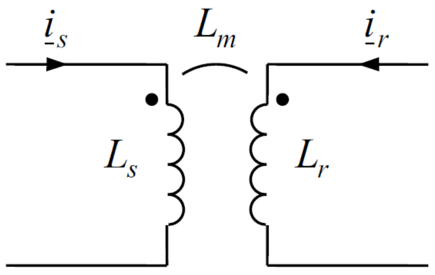


Fig. 2.2: Equivalent circuit diagram of a transformer [2]

Combining the differential equations of the fluxes and the relationship of the fluxes to the currents, we obtain the following system of differential equations and the equivalent circuit diagram for the SCIM in stator fixed coordinates. The parameter σ is the leakage factor and τ_σ is the leakage time constant. τ_{r} is the rotor time constant.

$$\frac{\mathrm{d}i_{\mathrm{s}\alpha}}{\mathrm{d}t} = -\frac{1}{\tau_\sigma}i_{\mathrm{s}\alpha} + \frac{R_{\mathrm{r}}L_{\mathrm{m}}}{\sigma L_{\mathrm{r}}^2 L_{\mathrm{s}}}\Psi_{\mathrm{r}\alpha} + p\omega_{\mathrm{me}}\frac{L_{\mathrm{m}}}{\sigma L_{\mathrm{r}}L_{\mathrm{s}}}\Psi_{\mathrm{r}\beta} + \frac{1}{\sigma L_{\mathrm{s}}}u_{\mathrm{s}\alpha} \tag{2.10}$$

$$\frac{\mathrm{d}i_{\mathrm{s}\beta}}{\mathrm{d}t} = -\frac{1}{\tau_\sigma}i_{\mathrm{s}\beta} - p\omega_{\mathrm{me}}\frac{L_{\mathrm{m}}}{\sigma L_{\mathrm{r}}L_{\mathrm{s}}}\Psi_{\mathrm{r}\alpha} + \frac{R_{\mathrm{r}}L_{\mathrm{m}}}{\sigma L_{\mathrm{r}}^2 L_{\mathrm{s}}}\Psi_{\mathrm{r}\beta} + \frac{1}{\sigma L_{\mathrm{s}}}u_{\mathrm{s}\beta} \tag{2.11}$$

$$\frac{\mathrm{d}\Psi_{\mathrm{r}\alpha}}{\mathrm{d}t} = \frac{L_{\mathrm{m}}}{\tau_{\mathrm{r}}}i_{\mathrm{s}\alpha} - \frac{1}{\tau_{\mathrm{r}}}\Psi_{\mathrm{r}\alpha} - p\omega_{\mathrm{me}}\Psi_{\mathrm{r}\beta} \tag{2.12}$$

$$\frac{\mathrm{d}\Psi_{\mathrm{r}\beta}}{\mathrm{d}t} = \frac{L_{\mathrm{m}}}{\tau_{\mathrm{r}}}i_{\mathrm{s}\beta} + p\omega_{\mathrm{me}}\Psi_{\mathrm{r}\alpha} - \frac{1}{\tau_{\mathrm{r}}}\Psi_{\mathrm{r}\beta} \tag{2.13}$$

$$\frac{\mathrm{d}\varepsilon_{\mathrm{el}}}{\mathrm{d}t} = p\omega_{\mathrm{me}} \tag{2.14}$$

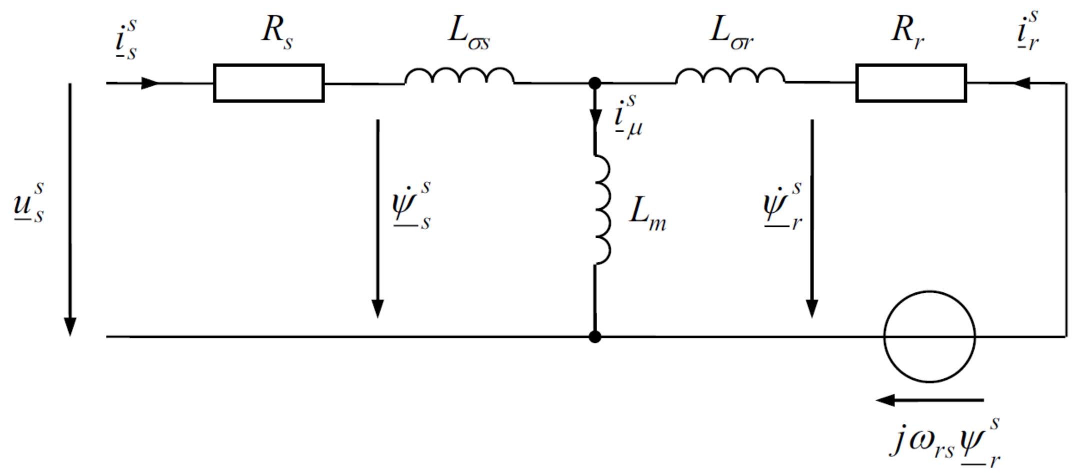


Fig. 2.3: Equivalent circuit diagram of the SCIM in stator-fixed coordinates [2]

After the electrical behavior of the SCIM has been described, the dependence of a current operating point and the resulting torque will be identified. The resulting torque can be determined, for example, from the power balance of the motor. The power is generally calculated as the product of the voltages and the conjugate complex currents, independent of the selected coordinate system. This leads to the two equations for the electrical powers in the stator

$$\underline{u}_{\mathrm{s}}\underline{i}_{\mathrm{s}}^* = \dot{\underline{\Psi}}_{\mathrm{s}}\underline{i}_{\mathrm{s}}^* + \mathrm{j}\omega_{\mathrm{rs}}\underline{\Psi}_{\mathrm{s}}\underline{i}_{\mathrm{s}}^* + R_{\mathrm{s}}\underline{i}_{\mathrm{s}}\underline{i}_{\mathrm{s}}^* \tag{2.15}$$

and the rotor

$$0 = \dot{\underline{\Psi}}_{\mathrm{r}}\underline{i}_{\mathrm{r}}^* + R_{\mathrm{r}}\underline{i}_{\mathrm{r}}\underline{i}_{\mathrm{r}}^*. \tag{2.16}$$

The supplied electrical power is converted in the motor in different forms. One part of the power is converted into heat due to the electrical resistance of the windings and wires. Another part of the input power is stored in the windings in form of magnetic energy. Finally, some of the power is converted into mechanical power, resulting in

$$P_{\mathrm{el}} = P_{\mathrm{d}} + \dot{W}_{\mathrm{magn}} + P_{\mathrm{me}} \tag{2.17}$$

as the power balance. Identifying now the power components from equation (2.17) in equations (2.15) and (2.16), yields in the electric power

$$P_{\mathrm{el}} = \frac{3}{2}\mathrm{Re}(\underline{u}_{\mathrm{s}}\underline{i}_{\mathrm{s}}^*), \tag{2.18}$$

the power dissipation

$$P_{\mathrm{d}} = \frac{3}{2}\mathrm{Re}(R_{\mathrm{s}}\underline{i}_{\mathrm{s}}\underline{i}_{\mathrm{s}}^* + R_{\mathrm{r}}\underline{i}_{\mathrm{r}}\underline{i}_{\mathrm{r}}^*) \tag{2.19}$$

and the change of the magnetic energy

$$\dot{W}_{\mathrm{magn}} = \frac{3}{2}\mathrm{Re}(\underline{\dot{\mathit{\Psi}}}_{\mathrm{s}}\underline{i}_{\mathrm{s}}^{*} + \underline{\dot{\mathit{\Psi}}}_{\mathrm{r}}\underline{i}_{\mathrm{r}}^{*}). \tag{2.20}$$

The remaining real part of the power

$$P_{\mathrm{me}} = \frac{3}{2}\mathrm{Re}(\mathrm{j}\omega_{\mathrm{rs}}\underline{\Psi}_{\mathrm{s}}\underline{i}_{\mathrm{s}}^{*}) = \frac{3}{2}\omega_{\mathrm{rs}}\mathrm{Im}(\underline{\mathit{\Psi}}_{\mathrm{s}}^{*}\underline{i}_{\mathrm{s}}) \tag{2.21}$$

is thus converted into mechanical power. It is the product of the torque and the mechanical angular velocity, so that the torque

$$T = \frac{3}{2}p\frac{L_{\mathrm{m}}}{L_{\mathrm{r}}}\mathrm{Im}(\underline{\mathit{\Psi}}_{\mathrm{r}}^{*}\underline{i}_{\mathrm{s}}) \tag{2.22}$$

is finally obtained. This torque equation motivates to choose the coordinate system in such a way that one coordinate axis is pointing in the direction of the magnetic flux. In this way, the torque only depends directly on the current in one of the two coordinate axes. It is irrelevant whether rotor flux or stator flux is used for the orientation of the coordinate system. The resulting coordinate axes are called direct and quadrature axis. Selecting the d-axis in the direction of the rotor flux, leads to

$$T = \frac{3}{2}p\frac{L_{\mathrm{m}}}{L_{\mathrm{r}}}\mathit{\Psi}_{\mathrm{rd}}i_{\mathrm{sq}} \tag{2.23}$$

for the torque, which is independent of the i_{sd} current, because the flux in q-direction is zero.

The generated torque drives the mechanical load attached to the rotor output of the motor, such as the gears of an electric or hybrid vehicle, conveyor belts or even pumps or fans. Depending on the type of the load, different requirements can be made on the control of the motor.

The input clamps of the motor are fed by a B6 bridge inverter. The inverter is supplied by a DC voltage. In principle, the inverter works like three ideal switches, which apply either the positive or negative potential to the corresponding output of the inverter. In total, eight different switching vectors can be connected to the motor with this method. By these eight switching vectors the voltages

$$\mathbf{u}_{\mathrm{a,b,c}}(t) = \frac{1}{2}\mathbf{s}_{\mathrm{a,b,c}}(t)u_{\mathrm{dc}}(t), \tag{2.24}$$

where the entries of the vector $\mathbf{s}_{\mathrm{a,b,c}}(t)$ can be set to $+1$ and -1 for the upper and lower switching position, can be applied to the inputs of the motor. The reference voltage is set here to half of the output voltage, but has no effect, because the windings of the motor are connected in star or delta the voltage between two lines is the difference between the two applied voltages. Furthermore, the voltages can be transformed into the $\alpha\beta$ coordinate system, where the zero component u_0 has only an influence on parasitic shift currents in

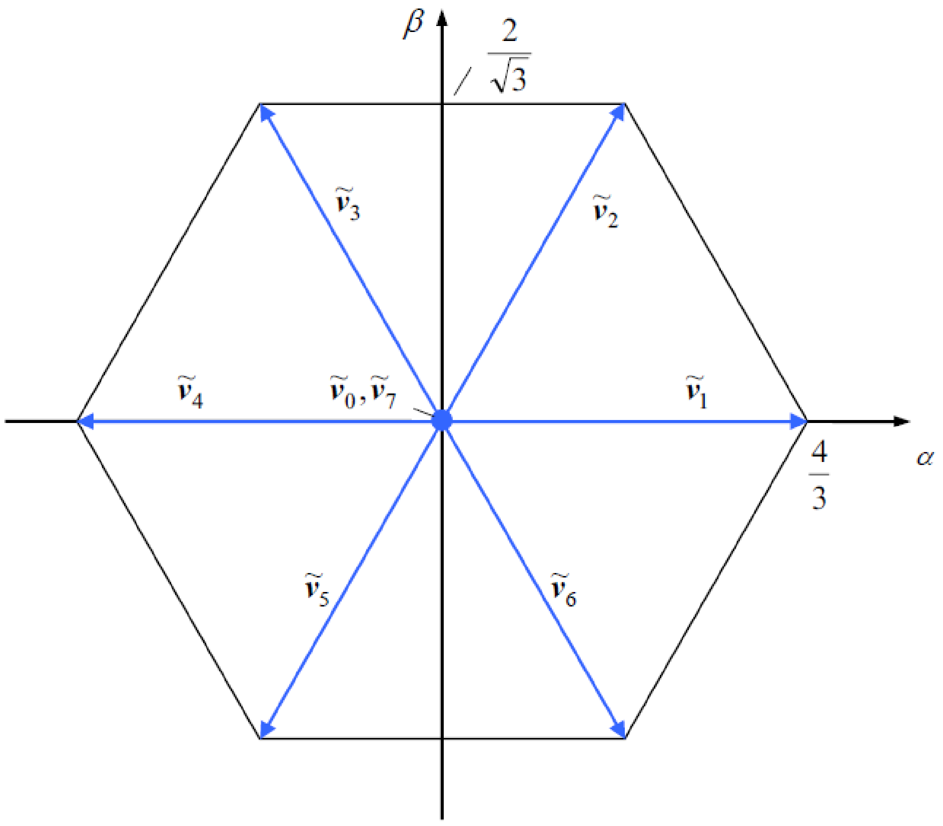


Fig. 2.4: Normalized switching vectors of the inverter [2]

the motor and is therefore neglected in the following. The both vectors v_0 and v_7 form the zero voltage vector, as only the voltage difference between two outputs is relevant and this is zero if all switches are in the same position. This results in the hexagon of switching vectors shown in figure 2.4.

From a technical point of view, the switches are implemented by transistors with a diode connected in parallel. Figure 2.5 shows the equivalent circuit diagram of an inverter with Metal Oxide Field Effect Transistor (MOSFET), but other transistors such as Isolated Gate Bipolar Transistors (IGBTs) can also be used. The capacitor at the input of the inverter avoids that the input current i_{dc}, which changes abruptly when switching, is fed back to the supply voltage source. In addition, the capacitor smoothes the input voltage.

The voltage source should ideally supply a constant DC link voltage to the inverter regardless of the current. If the inverter is supplied from the power grid, a simple diode rectifier can be used to convert to DC voltage. However, if a power flow from the motor to the voltage source is to be enabled, a bidirectional configuration, such as a mirrored configuration of the inverter, has to be used. A bidirectional power flow should be possible especially in mobile applications such as electric trains in order to recuperate braking energy. Another difficulty might be that the voltage is supplied by a battery, which does not provide a constant voltage level.

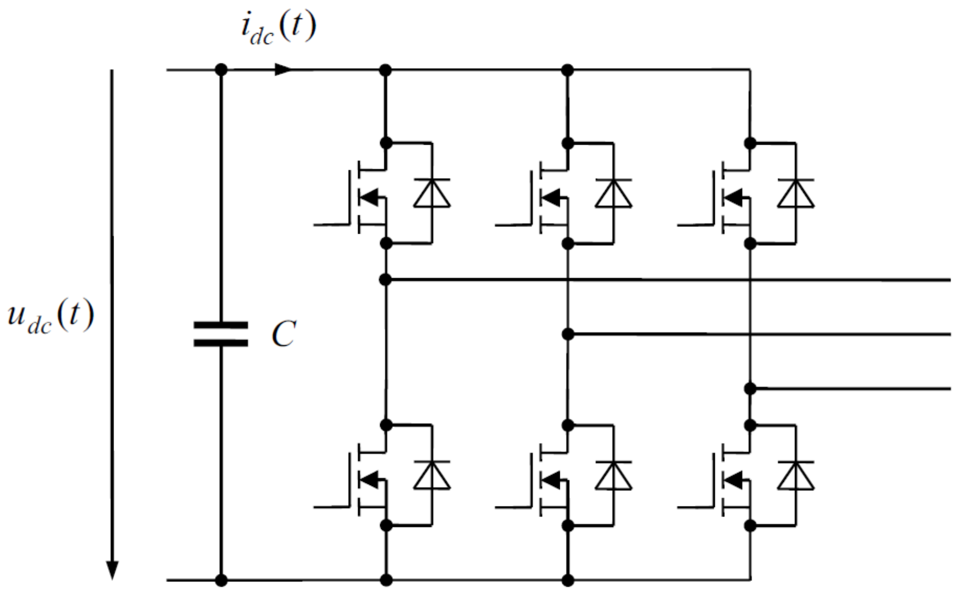


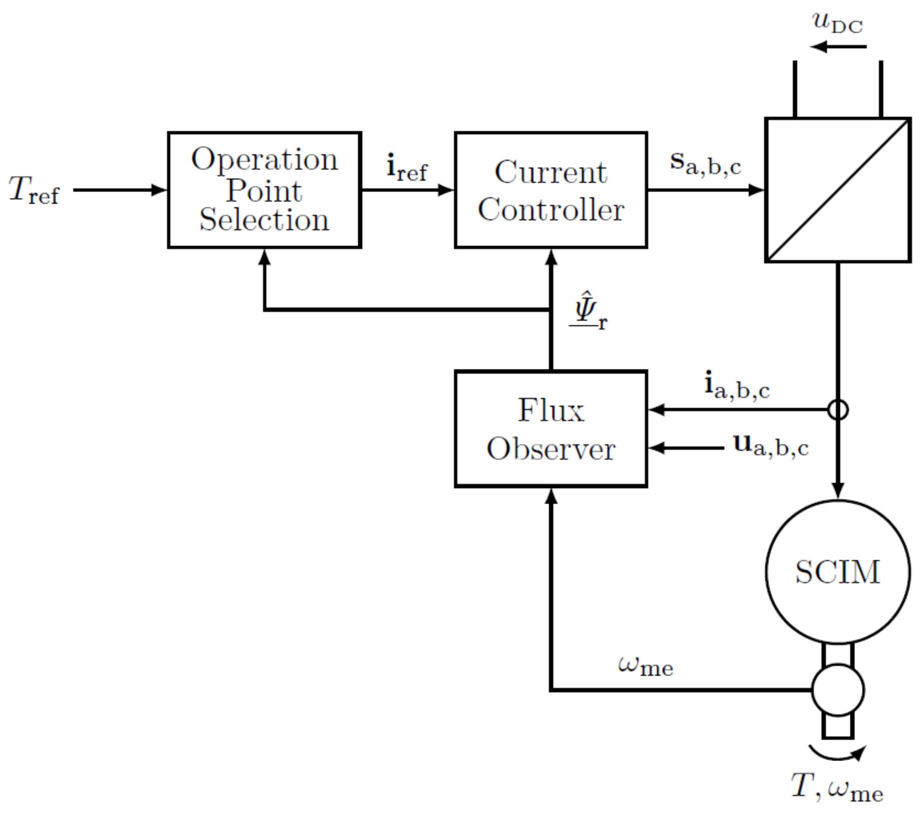
Fig. 2.5: Equivalent circuit diagram of the four quadrant inverter with MOSFETs [2]

2.2 Field-oriented control and flux observer

Various methods exist for the torque control of an IM. One of the most commonly used methods is the FOC. The name field-oriented refers to the coordinate system, which is oriented to the rotor flux of the motor, used for the control quantities. This simplifies the operation point selection for the given reference torque. As can be seen in formula (2.23), the torque in this coordinate system depends only on the magnitude of the magnetic flux and the i_{sq} current.

The controller is divided into three function blocks which can be seen in figure 2.6. The operation point selection handles the task of the torque control stage, which determines the reference currents in dq coordinates starting from a reference torque. These reference currents are controlled by the current control stage, which calculates the input voltages for the inverter. The torque of the IM depends on the magnitude of the magnetic flux. However, this flux is not measured, so it needs to be estimated for controlling the SCIM. Furthermore, the control is performed in flux-oriented coordinates. The angle of the rotor flux is required for the transformation of the state variables. For these reasons, a flux observer is used, which estimates the magnetic rotor flux from the measured currents and the known voltages.

For current control, we consider the current differential equations (2.10) and (2.11). These equations are summarized by the complex notation and transformed into a general



u_{DC}
T_{ref}
Operation Point Selection
$\mathbf{i}_{\mathrm{ref}}$
Current Controller
$\mathbf{s}_{\mathrm{a,b,c}}$
$\underline{\hat{\Psi}}_{\mathrm{r}}$
$\mathbf{i}_{\mathrm{a,b,c}}$
Flux Observer
$\mathbf{u}_{\mathrm{a,b,c}}$
SCIM
ω_{me}
T, ω_{me}

Fig. 2.6: Scheme of the FOC

coordinate system k to get

$$\sigma L_{\mathrm{s}} \dot{\underline{i}}_{\mathrm{s}}^{\mathrm{k}} = \underline{u}_{\mathrm{s}}^{\mathrm{k}} - \mathrm{j}\omega_{\mathrm{ks}}\sigma L_{\mathrm{s}}\underline{i}_{\mathrm{s}}^{\mathrm{k}} - \left(R_{\mathrm{s}} + R_{\mathrm{r}}\frac{L_{\mathrm{m}}^2}{L_{\mathrm{r}}^2}\right)\underline{i}_{\mathrm{s}}^{\mathrm{k}} + \left(-\mathrm{j}\omega_{\mathrm{rs}}\frac{L_{\mathrm{m}}}{L_{\mathrm{r}}} + \frac{L_{\mathrm{m}}R_{\mathrm{r}}}{L_{\mathrm{r}}^2}\right)\underline{\Psi}_{\mathrm{r}}^{\mathrm{k}} \tag{2.25}$$

as the complex differential equation for the currents. The first term on the right side $\underline{u}_{\mathrm{s}}^{\mathrm{k}}$ is the input voltage, which can be selected to regulate the corresponding current reference values. The second term describes a cross-coupling of the two current components by multiplication with the imaginary unit j. The ohmic voltage drop is given by the third term, while the last term describes feedback effects due to the rotor flux. A sensible choice of the input voltage can compensate both, the cross-coupled currents and the feedback of the rotor flux. The two components are combined in the feedforward part $\underline{u}_{\mathrm{s}}^{\mathrm{k0}}$, resulting in

$$\underline{u}_{\mathrm{s}}^{\mathrm{k}} = \Delta\underline{u}_{\mathrm{s}}^{\mathrm{k}} + \underline{u}_{\mathrm{s}}^{\mathrm{k0}} = \Delta\underline{u}_{\mathrm{s}}^{\mathrm{k}} + \mathrm{j}\omega_{\mathrm{ks}}\sigma L_{\mathrm{s}}\underline{i}_{\mathrm{s}}^{\mathrm{k}} + \left(\mathrm{j}\omega_{\mathrm{rs}}\frac{L_{\mathrm{m}}}{L_{\mathrm{r}}} - \frac{L_{\mathrm{m}}R_{\mathrm{r}}}{L_{\mathrm{r}}^2}\right)\underline{\Psi}_{\mathrm{r}}^{\mathrm{k}} \tag{2.26}$$

for the input voltages. This simplifies the differential equation for the current to

$$\dot{\underline{i}}_{\mathrm{s}}^{\mathrm{k}} = -\frac{1}{\tau_\sigma}\underline{i}_{\mathrm{s}}^{\mathrm{k}} + \frac{1}{\sigma L_{\mathrm{s}}}\Delta\underline{u}_{\mathrm{s}}^{\mathrm{k}} \tag{2.27}$$

with

$$\tau_\sigma = \frac{\sigma L_\mathrm{s}}{R_\mathrm{s} + R_\mathrm{r}\frac{L_\mathrm{m}^2}{L_\mathrm{r}^2}}. \tag{2.28}$$

This differential equation corresponds mathematically to a PT_1 plant model, so that proportional-integral controllers (PI controllers) can be used for controlling the currents. Different methods exist for the adjustment of the gains of the proportional and integral part of the PI controllers, such as the symmetric optimum or the magnitude optimum.

The input voltages calculated in dq coordinates are converted to a stator fixed coordinate system using the angle of the rotor flux determined by the flux observer and limited by the hexagon from figure 2.4. If the input voltage is within the limits, the I-component is integrated, respectively summed up in a discrete controller. If the summation is performed without checking the limits, a wind-up could arise and the input voltage would unnecessarily be stuck in the limit for a longer time.

The FOC assumes a continuous input space of voltages, but the inverter can only switch on eight different switching vectors to the inputs of the motor. Therefore, a modulation method must be used to determine a sequence of switching vectors from the input voltages, such as the pulse width modulation. Figure 2.7 shows the entire structure of the current control stage.

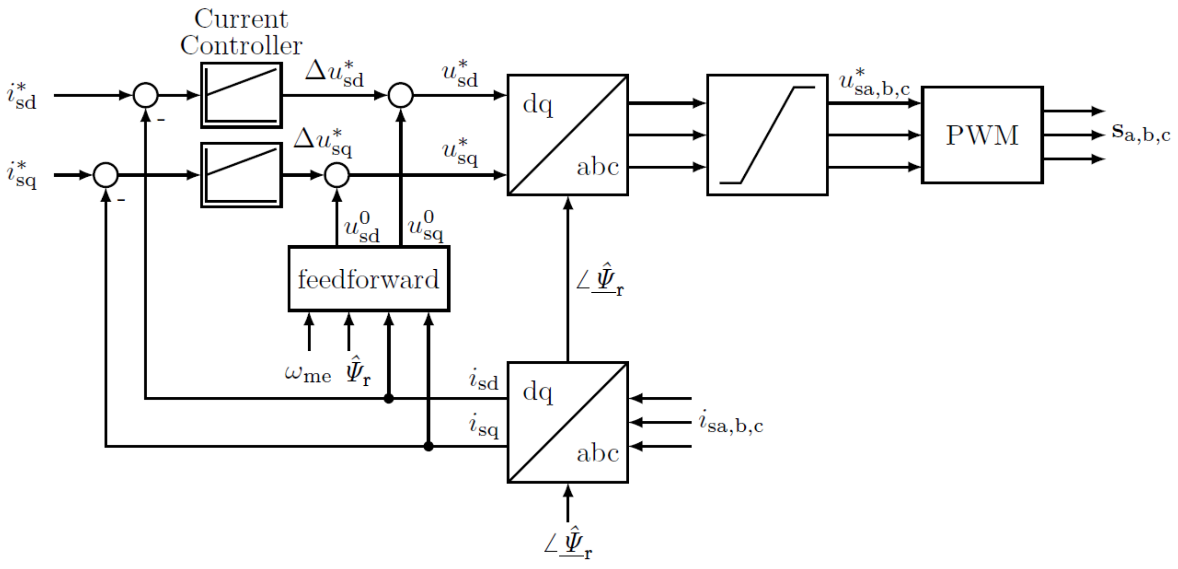


Fig. 2.7: Current control stage of the FOC

As can be seen from equation (2.23), the torque depends only on the two states i_sq and Ψ_rd. By selecting the coordinate system based on the direction of the magnetic rotor flux, the flux Ψ_rq is always zero by definition. The differential equation of the rotor flux from equation (2.7) can be simplified to

$$\dot{\varPsi}_{\mathrm{rd}} = -\frac{R_{\mathrm{r}}}{L_{\mathrm{r}}}\varPsi_{\mathrm{rd}} + \frac{R_{\mathrm{r}}L_{\mathrm{m}}}{L_{\mathrm{r}}}i_{\mathrm{sd}}. \tag{2.29}$$

This motivates a separate consideration of the two current components in order to control a reference torque. Furthermore, this equation shows that a given flux can be converted into a reference value for the i_{sd} current by the help of a PI controller. Also the reference value for the i_{sq} current results directly from the torque equation, if the magnetic flux is known, which is estimated with the help of the flux observer.

Thus, the only remaining question is which magnetic flux has to be upheld. The basic objective is to maximize the efficiency of the motor at each operation point, which is calculated by dividing the output power by the input power. The output power is given by the reference torque and the speed of the motor. As shown in equation (2.17), the absorbed input power separates into the ohmic losses, the re-magnetization work and the mechanical power, i.e., the output power. If only the steady state is considered, i.e., if all derivatives are equal to zero, the part of the re-magnetization work is eliminated and the only possible starting point to increase the efficiency is the minimization of the ohmic losses.

To minimize the ohmic losses from equation (2.19), first the unknown rotor currents must be identified. With the knowledge of the rotor flux, the rotor currents can be determined as a function of the stator currents to

$$i_{\mathrm{rd}} = \frac{1}{L_{\mathrm{r}}}\varPsi_{\mathrm{rd}} - \frac{L_{\mathrm{m}}}{L_{\mathrm{r}}}i_{\mathrm{sd}} \tag{2.30}$$

$$i_{\mathrm{rq}} = -\frac{L_{\mathrm{m}}}{L_{\mathrm{r}}}i_{\mathrm{sq}} \tag{2.31}$$

with the help of the equation (2.9). Under the steady state conditions, the equation (2.29) can be solved to

$$\varPsi_{\mathrm{r}} = L_{\mathrm{m}}i_{\mathrm{sd}} \tag{2.32}$$

for the rotor flux, resulting in a zero i_{rd} current. In this way, the ohmic losses

$$P_{\mathrm{d}}(i_{\mathrm{sd}}, i_{\mathrm{sq}}) = \frac{3}{2}\left(R_{\mathrm{s}}i_{\mathrm{sd}}^2 + \left(R_{\mathrm{s}} + R_{\mathrm{r}}\frac{L_{\mathrm{m}}^2}{L_{\mathrm{r}}^2}\right)i_{\mathrm{sq}}\right) \tag{2.33}$$

can be calculated as a function of the stator currents. The torque equation (2.23) is also simplified by the analysis in steady-state operation to

$$T_{\mathrm{ss}} = \frac{3}{2}p\frac{L_{\mathrm{m}}^2}{L_{\mathrm{r}}}i_{\mathrm{sd}}i_{\mathrm{sq}}, \tag{2.34}$$

so that the torque also depends only on the two stator currents. The goal is to minimize the losses for a given torque, resulting in an optimization problem with one constraint. To solve this, the corresponding Lagrange function

$$L(i_{\mathrm{sd}}, i_{\mathrm{sq}}, \lambda) = T(i_{\mathrm{sd}}, i_{\mathrm{sq}}) - \lambda P_{\mathrm{d}}(i_{\mathrm{sd}}, i_{\mathrm{sq}}) \tag{2.35}$$

is drawn up and solved by setting the partial derivatives to zero. So, an optimal current operating point for a given torque is obtained. Theoretically, these values for the current could be used as a reference for the current control stage, but they are valid only for the simplified assumption of steady-state operation. Therefore, equation (2.32) is used to calculate the optimal magnetic flux

$$\Psi_{\mathrm{rd_{opt}}} = L_{\mathrm{m}} i_{\mathrm{sd_{opt}}} = \sqrt{|T_{\mathrm{ref}}| \frac{2L_{\mathrm{r}}}{3p} \sqrt{1 + \frac{R_{\mathrm{r}}}{R_{\mathrm{s}}} \frac{L_{\mathrm{m}}^2}{L_{\mathrm{r}}^2}}} \tag{2.36}$$

for a given torque reference value. Another aspect for the calculation of the reference for the flux controller is the voltage limitation of the inverter. For this purpose, a modulation controller is used, which determines a maximum possible adjustable flux. For the detailed functionality and adjustment of the modulation controller, see [4] and [5]. If the magnetic flux is limited, the desired torque can not always be achieved. For this reason, the maximum possible torque is calculated on the basis of the reference value of the flux controller. The reference torque is limited to this value.

After the flux controller has determined a reference for the i_{sd} current and a reference for the i_{sq} current has been calculated by using the torque equation and the estimated flux, the magnitude of this current vector must be limited to the maximum current. The i_{sq} current is limited before, because if the i_{sd} current is limited first, the required flux for a torque may never be reached and thus the control will be stuck. Figure 2.8 shows the entire torque control stage including the modulation controller.

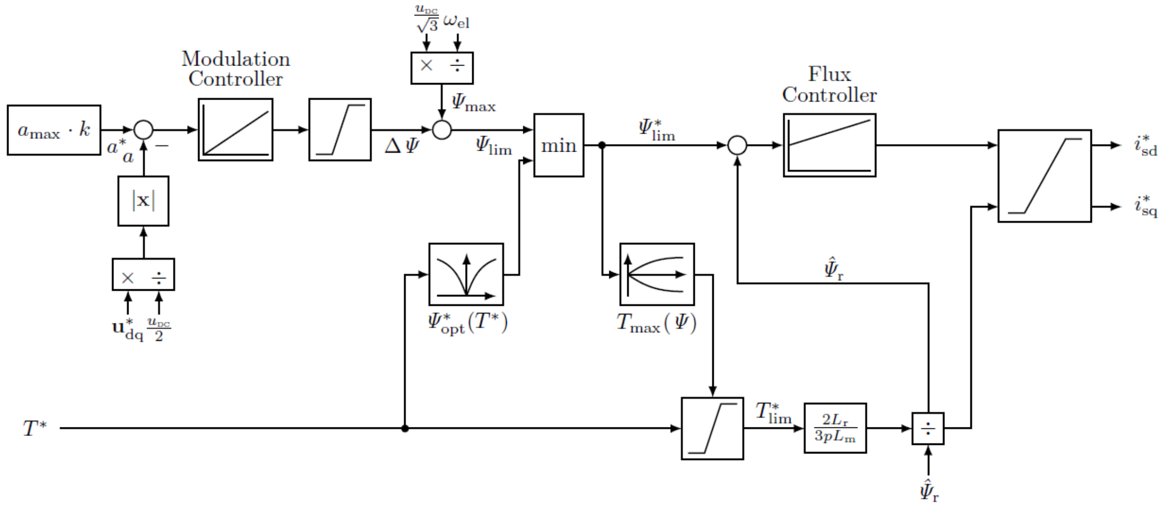


Fig. 2.8: Torque control stage of the FOC

As have been seen, the knowledge about the magnitude of the rotor flux is needed for the two control stages. Furthermore, the angle of the rotor flux is to be used for the coordinate transformation into the flux-oriented coordinate system. It is theoretically possible to use a sensor inside the motor which measures this magnetic flux, but for cost reasons this is typically not done. It is therefore necessary to implement a flux observer, which estimates the rotor flux from the measured variables.

There are different approaches for this task, such as open-loop observer which work with a voltage model, a current model or a combination of both models. In addition, closed-loop observers could also be used, which have an error feedback path. The various observers differ primarily in the speed range in which they provide accurate estimates of rotor flux. A detailed overview of the different observer types can be found in [6].

The flux observer used here is based on the current model. Looking at the differential equations for the magnetic rotor flux from equations (2.12) and (2.13), the observer's approach is very intuitive. The measured stator current is multiplied by the corresponding inductances and resistance and is integrated or respectively is summed up. In addition, there is a feedback path to take into account the influences of the magnetic fluxes, so that in the end the derivative is simply integrated. From the estimated rotor flux in $\alpha\beta$ coordinates, the magnitude and angle can be determined. Figure 2.9 shows the structure of the flux observer based on the current model.

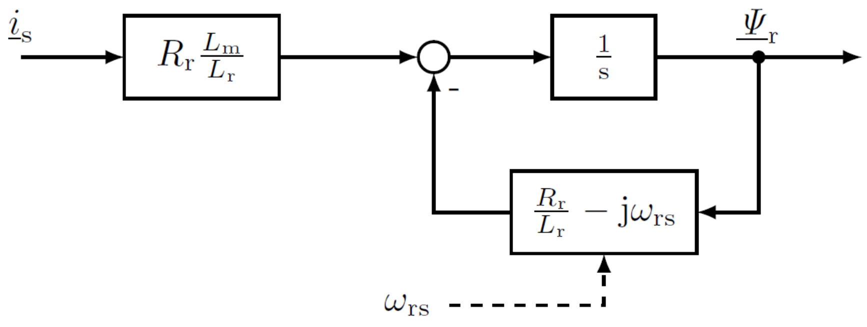

$\underline{i}_\mathrm{s}$
$R_\mathrm{r}\frac{L_\mathrm{m}}{L_\mathrm{r}}$
-
$\frac{1}{\mathrm{s}}$
$\underline{\Psi}_\mathrm{r}$
$\frac{R_\mathrm{r}}{L_\mathrm{r}} - \mathrm{j}\omega_\mathrm{rs}$
ω_rs

Fig. 2.9: Structure of the current model flux observer [6]

2.3 Simulation of the drive system

The control of the motor is implemented simulatively. The Python toolbox gym-electric-motor (GEM) is used to simulate the entire drive system. The documentation of this toolbox can be found on the Github page [7].

This toolbox simulates all components of the drive system from figure 2.1. The toolbox uses the structures of the gym toolbox [8] for the environments, which are actually designed for the implementation of RL controllers. The environments can also be used for any type of controller, such as the FOC that was discussed in the previous section. In addition to the physical system, an environment includes a reference generator, a reward function and a visualization.

For the physical system, it is possible to choose from several different DC motors, synchronous motors and asynchronous motors such as the SCIM that is used here. Depending on the motor, a matching inverter is used to power the motor, which itself is supplied by a simulated voltage supply. Moreover, it is possible to decide whether to use a continuous or a discrete action space. In the continuous action space, the voltages are switched to the motor by using a modulation scheme. Whereas, in the case of a discrete action space only one of the eight possible switching vectors can be applied to the inputs of the motor. Furthermore, different mechanical loads are available and can be selected, for example, a fixed mechanical load or a constant speed load. When initializing the environment, the motor parameters and the limitations of the system are specified. The motor parameters of the SCIM that are used here are listed in Table 2.1.

Parameter	**Symbol**	**Value**
Stator resistance	R_s	1.2878 Ω
Rotor resistance	R_r	1.2878 Ω
Main inductance	L_m	276.2 mH
Stator-side stray inductance	$L_{\sigma\mathrm{s}}$	19.4 mH
Rotor-side stray inductance	$L_{\sigma\mathrm{r}}$	19.4 mH
Pole pair number	p	1

Tab. 2.1: Motor parameters of the SCIM

The reference generator supplies reference values for the corresponding states of the environment. For example, sine functions, triangle functions, step functions, but also stochastic processes such as a Wiener process can be used. To compare different controllers, the environment can be seeded so that the same reference values are provided for an episode. The reward function takes the reference values and the momentary state to calculate a reward. This indicates how well the selected actions have performed. The reward function used here is explained in detail in chapter 3.

For the simulation of the drive system, the input of the inverter is passed in each time step. Internally, the differential equations of the motor are stored in the toolbox, which are solved by an ODE solver. A detailed description of such numerical solution methods of differential equations can be found under [9]. The calculated following states are then output together with the next reference values, the reward and a variable indicating whether the limit has been exceeded. Based on the state and the reference, the controller can calculate the next manipulated variable. Another important quantity in the simulation is the length of a time step τ, which in this case is 50 µs.

3 Reinforcement Learning Agent

The goal of the work is to implement a controller of an IM with unknown motor parameters. For this reason, classical control methods such as an FOC are not applicable, since they require knowledge about the motor parameters for adjusting the PI controllers and the decoupling of the current components. Also for an MPC accurate system knowledge is needed, which is why this cannot be used either. Since several years, the so-called reinforcement learning has been applied for the control of systems of any kind, which is a form of machine learning. This has the advantage that an optimal policy for the determination of the input variable or action is learned during a training process and does not require system knowledge. Nevertheless, expert knowledge about the system to be controlled can be incorporated into the training via the feature engineering and the reward function.

During the training process, the agent selects an action a_k based on the momentary state s_k of the environment, which is then applied to the environment. This results in a subsequent state s_{k+1} of the environment, which is used to calculate a reward r_k for the selected action. The goal of the agent is to maximize this reward, so the reward function must use a metric that measures the performance of the agent. Figure 3.1 illustrates the general structure of this training process.

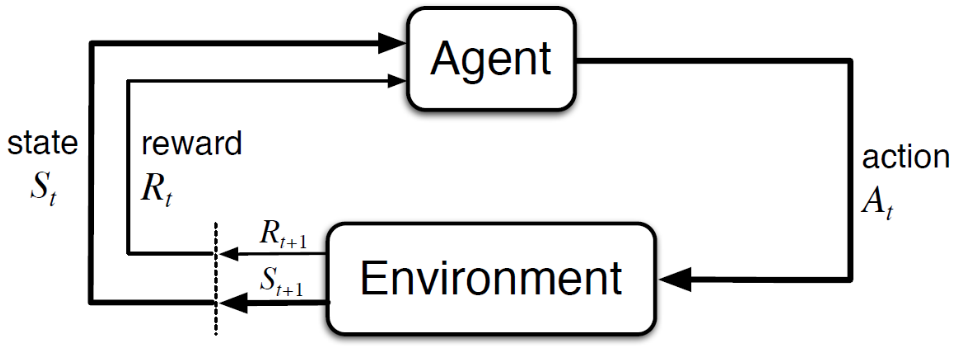


Fig. 3.1: Structure of the training of a RL agent [10]

3.1 Mathematical formulation of the problem

In order to use an RL agent for the control of a system, the state transitions of this system must be able to be described mathematically as an MDP. Therefore, first the question of how to characterize an MDP has to be clarified. An MDP is an extension of a Markov chain. A Markov chain is a stochastic process in which the state transition probabilities depend only on the momentary state. In a Markov chain, there is no possibility to influence the subsequent state. Therefore, the assumption of a Markov chain is extended by an input variable u, so that the transition probabilities of an MDP additionally depend on the input u_k. Mathematically, the state transition probability is expressed as

$$P(s', s_k, u_k) = \mathbb{P}[s_{k+1} = s' | S_k = s_k, U_k = u_k]. \tag{3.1}$$

In particular, the state transition probabilities do not depend on past or future states. For technical processes it is impossible that a state depends on following states, therefore this is also given for the SCIM. On the other hand, it is quite possible that a state depends on more than the previous state s_{k-1}. For the SCIM, the magnetic flux is integrated over time, which is why the state depends on past values in this way. This problem can be avoided if the magnetic flux is also part of the state. If the magnetic flux is part of the state, the problem arises that it is usually not measured. Therefore some parts of the state are not observable. Such a process is called POMDP. Figure 3.2 shows all the processes discussed here.

		All states observable?	
		Yes	No
Actions?	No	Markov chain	Hidden Markov model
	Yes	Markov decision process (MDP)	Partially observable MDP

Fig. 3.2: Overview of Markov chains and decision processes [11]

RL agents are designed to make decisions for MDPs and not for POMDPs. The problem, that arises when not all states are observable, is that different state transition probabilities may hold for the observable part of the state and one action, because the not observable part of the state is different. This causes the agent to be unable to take an optimal action based on the observation. As a consequence, it is necessary to find a method that provides an estimate for at least the missing quantities. This implies for the control of the SCIM that an estimator for the magnetic flux has to be used. This does not necessarily have to provide the exact values. It is sufficient if all information is included in the estimated values. In the following, it is assumed that such an estimator exists and that the observable state is completed to a full state.

The goal of an RL agent is to find an optimal policy, which is a decision rule for choosing actions based on the state in order to maximize the long-term reward. More precisely, a policy $\pi(a|s)$ is a probability distribution for how likely an action will perform, in a given state. A problem arises for continuous tasks, as in this case, because the expected return, which is the sum of all subsequent rewards, can go to infinity if the process does not terminate. Therefore, a discounted return is used, which means that all rewards are weighted with an exponentially decreasing value for future time steps, the so-called discount factor γ, so that the discounted return is calculated as

$$G_\mathrm{k} = R_{k+1} + \gamma R_{k+2} + \gamma^2 R_{k+3} + ... = \sum_{i=0}^{\infty} \gamma^i R_{k+i+1}, \tag{3.2}$$

where the value γ must be greater than 0 and smaller than 1. A small value means that especially the immediate next rewards are considered and therefore the discounted return is shortsighted. On the other hand, a high value close to 1 provides a farsighted discounted return. In rearranged form, the discounted return

$$G_k = R_{k+1} + \gamma(R_{k+2} + \gamma^1 R_{k+3} + ...) = R_{k+1} + \gamma G_{k+1}, \tag{3.3}$$

serves as an approach for many RL algorithms. Often RL algorithms estimate value functions, which indicate how well a policy π performs for a given state. The state-value function v_π is defined as

$$v_\pi(s) = \mathbb{E}_\pi[G_k|S_k = s], \;\; \forall s \in S, \tag{3.4}$$

where $\mathbb{E}_\pi$ is the expectation operator. It is also interesting to look at this function additionally in dependence of the next action. This function

$$q_\pi(s,a) = \mathbb{E}_\pi[G_k|S_k = s, A_k = a] \tag{3.5}$$

is called action-value function. These functions can be used to evaluate a policy, but the task is to find an optimal policy. To obtain the relation between the policy and the state transitions of the environment, the expectation operator of the value function is solved to yield

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')], \;\; \forall s \in S \tag{3.6}$$

as the Bellman equation. In this way, the expected return is separated into a sum of the reward of the present step and the expected return of the subsequent state. For this equation, a policy that maximizes the state-value function

$$v_*(s) = \max_\pi v_\pi(s), \;\; \forall s \in S \tag{3.7}$$

in each state or the action-value function

$$q_*(s,a) = \max_\pi q_\pi(s,a), \;\; \forall s \in S, \;\; \forall a \in A(s) \tag{3.8}$$

in each state for each action should be find. Once an optimal state-value function has been found, it is simple to derive the optimal action-value function

$$q_*(s,a) = \mathbb{E}[R_{k+1} + \gamma v_*(S_{k+1}) | S_k = s, A_k = a] \tag{3.9}$$

from it. This results from the reward for the next time step and the state-value function for the subsequent state when action a is taken. In the opposite way, the optimal state-value function can be represented as a function of the action-value function. If the action-value function is maximized as a function of the next action, assuming that an optimal policy is then followed, this value and the value of the optimal state-value function must be identical for all states. In this way, the optimal state-value function can be expressed independently from a policy. The resulting equation

$$v_*(s) = \max_{a \in A(s)} q_{\pi_*}(s,a) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')] \tag{3.10}$$

is called the Bellman optimality equation for v_*. In analogy, the Bellman optimality equation for q_* can be formulated as

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q_*(s',a')]. \tag{3.11}$$

Figure 3.3 shows the Bellman optimality equations as backup diagrams.

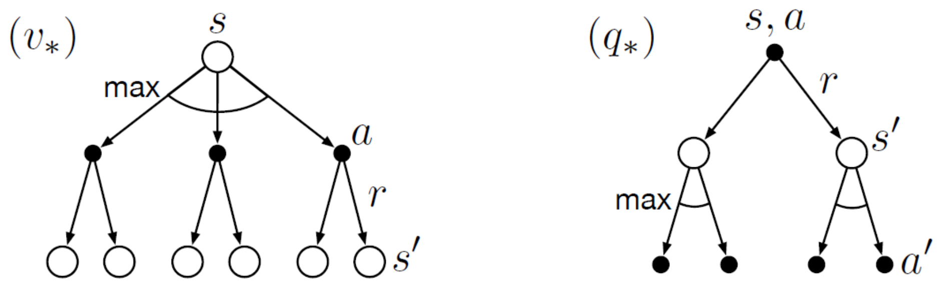


Fig. 3.3: Backup diagrams of the Bellman optimality equations [10]

For MDPs with finite states and actions, these equations can be solved explicitly, leading to an optimal policy. However, the greater the number of states and actions, the greater the computational effort, so that an explicit solution exists theoretically, but cannot be determined practically in a reasonable amount of time. For continuous state spaces, therefore, the problem cannot be solved explicitly, so solutions have to be approximated by using RL algorithms.

3.2 RL algorithms

A variety of RL algorithms are available by now to solve a problem just described. But many of these algorithms are only designed for a discrete or a continuous action space. In this case both options would be possible, because on the one hand the switching vectors can be chosen directly and on the other hand a modulation technique like in the FOC could be used to set intermediate voltage values to create a continuous action space. One problem of RL agents is that they do not explicitly consider system boundaries, so a safety layer should be included when implementing them on a real testbench. To simplify this safety layer, a discrete action space is used here. Further information on the safety layer can be found in chapter 5.

Furthermore, the different algorithms differ in the way which values are approximated and thus how they come to a decision, for example, Q-learning is often used. There, the action-value function is tried to be approximated in order to select the action with the highest q-value. Other algorithms directly optimize a policy that outputs the probabilities resulting in the highest return.

Another difference between the algorithms is how the actions are selected during the training. A general distinction is made between off-policy and on-policy learning algorithms. Off-policy algorithms use a different policy for generating actions during training than is learned, while in an on-policy learning algorithm the policy used during training and the learned policy are the same. Closely related to this aspect is the trade-off between exploration and exploitation. Exploration refers to the fact that as many different areas of the action and state space as possible are studied in order to avoid that the learned policy finishes in a local maximum. On the other hand, the training should be fast, but at the expense of exploration.

All algorithms, which operate with continuous states, have in common that they approximate functions by using artificial neural networks (ANNs). There are different types of ANNs, in this case a multilayer perzeptron used. These consist of different layers with different numbers of neurons. The neurons of one layer are connected with all neurons of the next layer. On the one hand, there is the input layer, which receives the parameters from the outer environment. This is followed by an arbitrary number of hidden layers before the output layer is reached at the end of the neural network. The structure of such a fully connected neural network can be seen in Figure 3.4.

Each neuron contains an activation function. If a signal is applied to the inputs of the ANN, this activation function is evaluated layer by layer and the results are passed on to the next layer until the output layer is reached. These activation functions depend on multiple parameters, the network weights, which determine the behavior of the ANN. During training, these network weights are adjusted so that the ANN performs the desired behavior. The behavior of a neuron is described by

$$y = f_{\mathrm{act}}(\mathbf{A} \cdot \mathbf{x} + b), \tag{3.12}$$

where f_{act} is the activation function, $\mathbf{A}$ are multiplicative network weights, $\mathbf{x}$ is the input vector of a neuron, and b is the bias. As the ANN is organized systematically and hierarchically the gradient can be determined for each neuron. Therefore, gradient-based methods are often used to adjust the network weights in training.

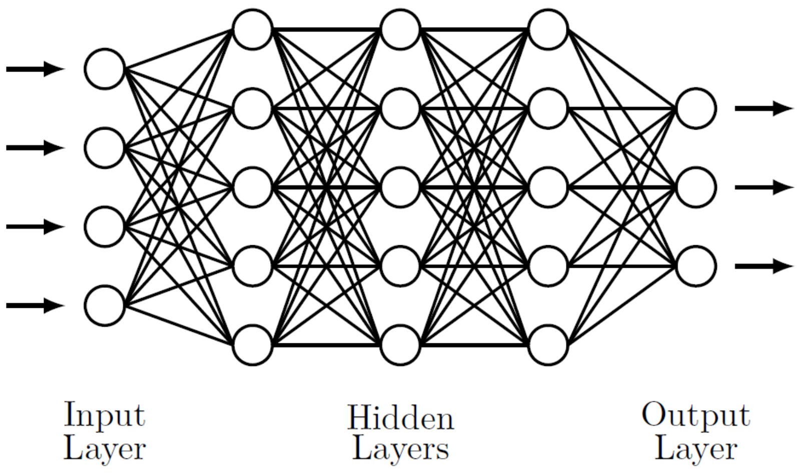


Fig. 3.4: Structure of a fully connected neural network

The number of hidden layers, the number of neurons per layer and the activation function are examples of hyperparameters. Hyperparameters are all parameters that can be adjusted in an RL algorithm before starting the training to optimize the learning process. As it is usually not known which set of hyperparameters leads to the best results, a hyperparameter optimization is performed. Moreover, it is unclear which RL algorithm is most suitable for the problem. In this case the hyperparameter optimization additionally optimizes the RL algorithm itself as a parameter. The implementation and the results of the hyperparameter optimization are presented in Chapter 6. In the following, the algorithms considered there are explained.

3.2.1 Deep Q Networks algorithm

The deep Q networks (DQN) algorithm is based on approximating the action-value function from equation (3.11). As already shown in subsection 3.1, the action-value function can be used to determine an action by selecting the action with the highest q-value, which is called greedy strategy. To train the ANN to estimate q-values as accurately as possible, the loss function

$$L_\mathrm{i}(\Theta_\mathrm{i}) = \mathbb{E}_\mathrm{s,a}[(y_\mathrm{i} - q(s,a;\Theta_\mathrm{i}))^2] \tag{3.13}$$

with the target

$$y_\mathrm{i} = \mathbb{E}_\mathrm{s'}[r + \gamma \max_{a'} q(s',a';\Theta_\mathrm{i-1})|s,a] \tag{3.14}$$

is used, where Θ_i are the network weights in iteration i. To adjust the network weights so that the loss function is minimized, a stochastic gradient descent (SGD) step with step size α, which is called learning rate in the context of RL, is performed. This requires the derivation of the loss function according to the network weights, which is

$$\nabla_{\Theta_\mathrm{i}} L_\mathrm{i}(\Theta_\mathrm{i}) = \mathbb{E}_{s,a;s'}[(r + \gamma q(s',a';\Theta_\mathrm{i-1}) - q(s,a;\Theta_\mathrm{i}))\nabla_{\Theta_\mathrm{i}} q(s,a;\Theta_\mathrm{i})]. \tag{3.15}$$

In order to ensure that the agent explores a large part of the state and action space during training, the greedy policy is not sufficient, so the so called epsilon greedy policy is used to determine the actions during learning. In contrast to the greedy policy, a random action is drawn from the action space with probability ϵ instead of choosing the action with the highest q-value. Since a different policy is used for training and for later operation, this is an off-policy algorithm. This allows to use an experience replay buffer for storing the experiences $e_k = (s_k, a_k, r_k, s_{k+1})$ of the last N time steps. If this replay buffer is not included, the agent can only be trained with contiguous sequences, which is why the data is highly correlated. To avoid this, the transitions of a minibatch are randomly drawn from the replay buffer. Algorithm 1 shows the pseudocode of training the DQN agent.

Algorithm 1 Deep Q-learning with Experience Replay [12]

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $q$ with random weights $\Theta_0$
for episode= $1, ..., M$ do
    Initialize state $s_1$
    for $k = 1, ..., K$ do
        With probability $\epsilon$ select random action $a_k$
        otherwise select $a_k = \max_a q_*(s_k, a, ; \Theta_\mathrm{i})$
        Execute action $a_k$ in simulation and observe reward $r_k$ and state $s_{k+1}$
        Store transition $e_k = (s_k, a_k, r_k, s_{k+1})$ in $D$
        Set $s_k = s_{k+1}$
        Sample random minibatch of transitions $(s_\mathrm{j}, a_\mathrm{j}, r_\mathrm{j}, s_{j+1})$ from $D$
        Set $y_\mathrm{j} = \begin{cases} r_\mathrm{j} & \text{for terminal } s_\mathrm{j} \\ r_\mathrm{j} + \gamma \max_{a'} q(s_{j+1}, a'; \Theta_\mathrm{i}) & \text{for non-terminal } s_\mathrm{j} \end{cases}$
        Update weights $\Theta_{\mathrm{i}+1} \leftarrow \Theta_\mathrm{i} - \alpha \nabla_{\Theta_\mathrm{i}} L_\mathrm{i}(\Theta_\mathrm{i})$
    end for
end for
```

3.2.2 Rainbow algorithm

The Rainbow algorithm is an extension of the DQN and thus also tries to estimate the q-values. The name Rainbow is based on the fact that many different extensions, that are independent, are applied when the agent is learning. These enhancements are aimed at speeding up the training as well as improving the final results. In the following, the improvements compared to the DQN algorithm are explained briefly.

In Q-learning, the q-value is often overestimated due to the maximization in equation (3.14). To overcome this problem, two different estimators for the q-value are used. At each training step, one of the two q-functions is randomly selected and the other q-function is used in the maximization from equation (3.14). In this way, the two q-functions are decoupled to avoid an overestimation of the q-values. For selecting an action both learned q-functions can be used. In order for the learning process to be as data efficient as the DQN algorithm, the average of both q-values can also be considered for decision finding. The mathematical details and the corresponding pseudocode can be looked up in [13].

The Rainbow algorithm also uses a replay buffer, but in an optimized form. For the replay buffer of the DQN, the transitions for training the agent are drawn randomly with equal probability. However, not all transitions have the same influence on the learning. The transitions with a higher effect should be chosen more often to speed up the training. The absolute temporal difference error

$$\delta_k = |r_{k+1} + \gamma \max_{a'} q(s', a'; \Theta_k) - q(s, a; \Theta_k)|^w \tag{3.16}$$

is used as a metric to evaluate the impact of a transition on the learning process, where the parameter w determines the shape of the probability distribution. By this method, all transitions in the prioritized replay buffer are weighted and drawn with the corresponding probability during the training. Nevertheless, the probability of a transition should never be zero, because the change of the q-function can also increase the temporal difference error and this is only updated after the use of a transition. The details of this enhancement can be found in the paper [14].

Another improvement of the DQN algorithm is the use of a so-called dueling network architecture. This special network architecture splits the estimation of the action-value internally into an estimation of the state-value and the advantage

$$a_\pi(s, a) = q_\pi(s, a) - v_\pi(s) \tag{3.17}$$

and merges these two components in an additional layer to the q-value. The separation of the two values should lead to the agent not knowing the value of an action in every state, but learning in general the effect of an action. The implementation and the mathematical background are described in [15].

The DQN algorithm considers only one step forward when calculating the return and estimates the following values with the help of the q-function. To improve the estimate of the expected return, the rewards of further time steps can be explicitly considered. Using the n-step return

$$r_k^{(\mathrm{n})} = \sum_{i=0}^{n-1} \gamma^i r_{k+i+1} \tag{3.18}$$

when minimizing the loss function (3.13) usually speeds up the training. A more detailed description can be read in [16].

In MDPs, the state transitions are characterized by probability distributions. The motivation to estimate probability distributions when estimating q-values arises from here. These, in fact, satisfies all the conditions of the Bellman equation. The distribution

$$d_k = (\boldsymbol{z}, \mathbf{p}_\Theta(s_k, a_k)) \tag{3.19}$$

with a discrete support vector $\boldsymbol{z}$, should be as close as possible to the distribution of the actual return. For the target, the result is also a distribution

$$d'_k = (r_{k+1} + \gamma \boldsymbol{z}, \mathbf{p}_{\overline{\Theta}}(s_{k+1}, \overline{a}^*_{k+1})), \tag{3.20}$$

so that the Kullback-Leibler divergence between these two distributions has to be minimized by adjusting the network weights. To obtain the q-values and take a greedy action the support vector is multiplied by the learned distribution, which depends on the state and the action. A complete derivation of this so-called distributional RL can be seen in the paper [17].

A last extension of the Rainbow algorithm in contrast to the DQN is the use of noisy network parameters. All parameters of the ANN are randomized with noise to increase the exploration of the agent as it is limited by the epsilon greedy policy.

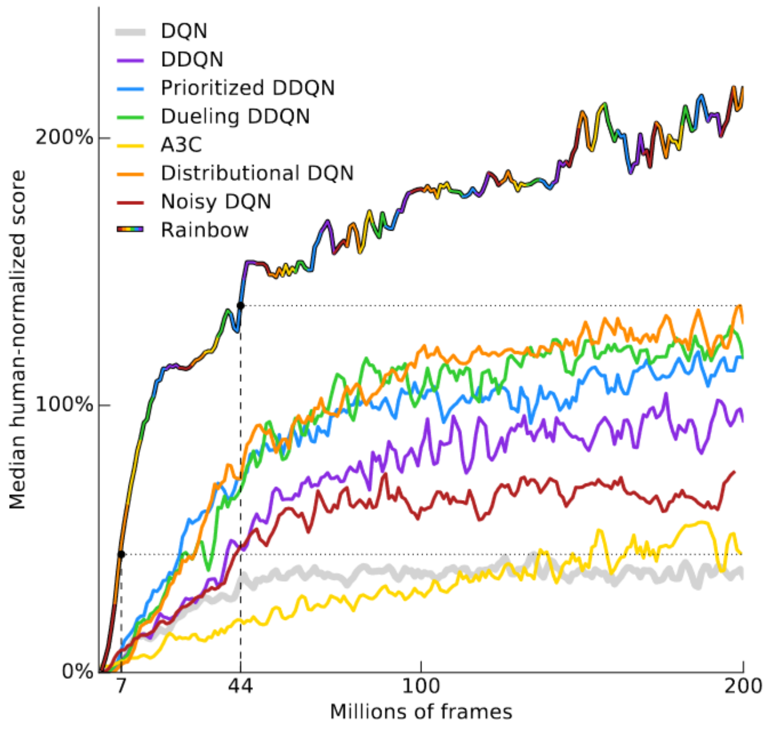


Fig. 3.5: Performance of multiple configurations when controlling 57 Atari games [18]

These extensions to the DQN algorithm are intended to speed up training and increase cumulative returns, as mentioned earlier. Figure 3.5 shows the learning curves of agents with different combinations of the extensions described here. The performance to be seen was recorded during the training of 57 different Atari games and is presented in dependence of an average human performance. It can be seen that all enhancements lead to an improvement of the performance in the long run and a large part as well as to a faster learning process. Moreover, the performance of the Rainbow algorithm exceeds all other configurations by far and especially with respect to the DQN the performance is more than four times higher. Whether this significant improvement also holds true with respect to the control task posed here will become apparent during the hyperparameter optimization.

3.2.3 Proximal Policy Optimization algorithm

The PPO algorithm was developed by OpenAI [19]. Similar to the DQN and Rainbow algorithms, this is a gradient-based learning method, but a stochastic policy is explicitly learned. In the PPO algorithm, the same policy is used for training as it is used in later operation. Because of this fact it is an on-policy training. Therefore, it is not possible to use a replay buffer, so the algorithm is not as data efficient as the previously discussed algorithms. Another difference is that this algorithm is suitable for a discrete as well as a continuous action space.

Gradient-based estimators for a policy typically use an objective function to be maximized of the form

$$L^{\mathrm{PG}}(\Theta) = \hat{\mathbb{E}}_k[\log \pi_\Theta(a_k|s_k)\hat{A}_k] \tag{3.21}$$

where π_Θ is the stochastic policy and $\hat{A}_k$ is an estimation of the advantage function. The advantage is calculated as the expected return subtracted by the momentary state value. One problem with the use of this objective function is that the large policy updates drastically change the policy and often destroy it. Therefore, these policy updates are limited in the PPO algorithm as it is done for example with the Trust Region Policy Optimization (TRPO) by an surrogate objective function [20]. In this objective function

$$r_k(\Theta) = \frac{\pi_\Theta(a_k|s_k)}{\pi_{\Theta_{\mathrm{old}}}(a_k|s_k)}, \tag{3.22}$$

the logarithm of the policy is replaced by the ratio of the old to the new policy. Furthermore, the change of the policy should be limited, because a maximum for the Kullback-Leibler divergence between the old and the new policy is specified in a constraint, since the Kullback-Leibler divergence is a metric for the difference between two probability distributions. Thus, the optimization problem

$$\max_\Theta = \hat{\mathbb{E}}_k[r_k(\Theta)\hat{A}_k] \tag{3.23}$$

$$\text{subject to } \hat{\mathbb{E}}_k[\mathbf{KL}[\pi_{\Theta_{\mathrm{old}}}(\cdot|s_k), \pi_\Theta(\cdot|s_k)]] \leq \delta \tag{3.24}$$

results for the adjustment of the network parameters Θ. Although this optimization problem can be approximated by an unconstrained optimization problem, which is performed in the TRPO method, but the solution is neither trivial nor optimal.

The PPO algorithm simplifies the limitation of the policy updates by clipping the objective function, resulting in

$$L^{\mathrm{CLIP}} = \hat{\mathbb{E}}_k[\min(r_k(\Theta)\hat{A}_k, \mathrm{clip}(r_k(\Theta), 1-\epsilon, 1+\epsilon)\hat{A}_k)] \tag{3.25}$$

as the objective function of the PPO. The motivation to use the minimum of the unlimited and clipped surrogate objective function is not intuitively, but can be explained with the help of a graph of the clipped objective function, which can be seen in figure 3.6.

The left graph shows L^{CLIP} for a positive advantage. In this case the probability of an action will be increased. As a consequence, the ratio $r_k(\Theta)$ will be greater than one. Thus,

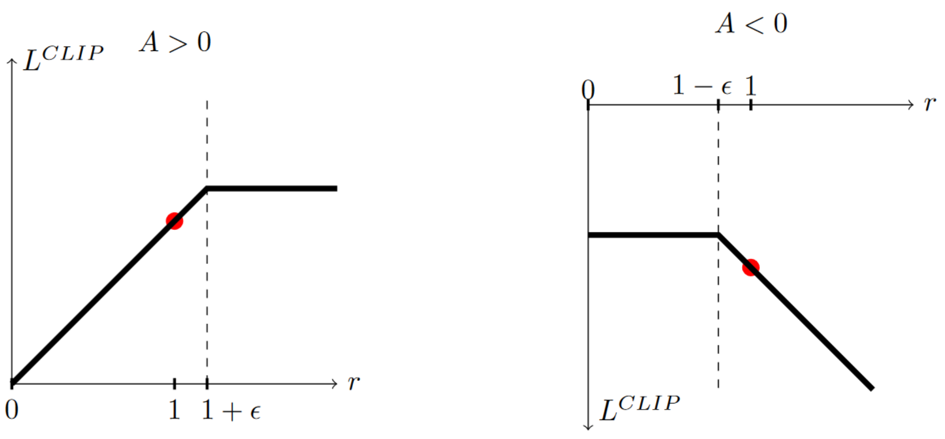


Fig. 3.6: Clipped objective function L^{CLIP} for positive and negative advantage [19]

the objective function must be clipped upwards. On the right side you can see the graph for the reverse case, i.e. that the advantage is smaller than zero. Analog to the other case, the probability will be decreased, the ratio $r_k(\Theta)$ is less than 1 and thus the objective function must be clipped downwards to avoid the policy updates becoming to large. To calculate the advantage

$$\hat{A}_k = -v(s_k) + r_k + \gamma r_{k+1} + ... + \gamma^{K-k+1} r_{K-1} + \gamma^{K-k} V(s_K) \tag{3.26}$$

for a policy that collects K timesteps for an update, an estimate of the state-value $V(s)$ is needed. Since an ANN is used for the policy, it can be used to estimate the state-value function as well. An additional output of the ANN provides the state-value. To train this output suitably, the squared error of this estimate to the target state value is included in the objective function with a negative sign, as can be seen in equation (3.27).

During training, for exploration, the action is drawn from a categorical distribution with the probabilities of the stochastic policy. After training in the normal operation, the action with the highest probability is selected. To improve exploration an entropy bonus S is also added to the objective function. More details about this entropy bonus can be found in [21]. This results in the objective function

$$L_k(\Theta) = \hat{\mathbb{E}}_k[L_k^{\text{CLIP}}(\Theta) - c_{\text{VF}}(V_\Theta(s_k) - V_k^{\text{targ}})^2 + c_{\text{entr}} S[\pi_\Theta(s_k)]] \tag{3.27}$$

which is used to train the PPO agent by a gradient ascent method. The coefficients c_{VF} and c_{entr} are further hyperparameters of the PPO agent, which can be used to adjust the influences of the two additional terms of the objective function. Algorithm 2 shows the pseudocode of training the PPO agent.

3.2.4 Evolution Strategies

The RL algorithms of the class of Evolution Strategies (ES) operate on the principle "survival of the fittest". An agent starts the training process with a population of random

Algorithm 2 Proximal Policy Optimization [19]

```
for episode$k = 1, ..., N$ do
    Run policy $\pi_{\Theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, ..., \hat{A}_T$ by equation (3.26)
    Optimize $L$ wrt $\Theta$, with $K$ epochs and minibatch size $M \leq T$
    $\Theta_{\text{old}} \leftarrow \Theta$
end for
```

parameter vectors, which determine the behavior of a stochastic policy. These are used to determine the actions performed by the environment during an episode. Afterwards, the individual parameter vectors are evaluated by an objective function F. Depending on the values of this function, the parameter vectors of the previous generation are combined, resulting in a new parameter vector. A new population is formed by adding a random component to the new parameter vector. This procedure can be repeated again and again to train the agent.

An agent of the class of ES learns the policy by using an ANN. But the exploration is done by using different policies and not by randomly drawing an action from a probability distribution. Instead, even during training, the action with the highest probability is executed for any policy.

A major difference to the previous algorithms is that no gradient of the policy or, in the case of the DQN and Rainbow, of the q-function is involved in the update of the network parameters. Therefore, no backpropagation through the ANN is required to calculate the gradient. Only a policy is trained, means no value function has to be learned. For the other algorithms, the reward function should have a gradient pointing to the optimum. This is not necessary with the ES algorithm, so that one has more degrees of freedom in the definition of the reward function.

The gradient for the updates of the network parameters by using the stochastic gradient ascent method are calculated by the expected value of the random changes of the parameter vector weighted with the corresponding return. The expectation operator simplifies to the average operator divided by the standard deviation σ, since all returns and the changes in the parameter vectors are independent of each other. Thus, the gradient used for the gradient ascent step results in

$$\nabla_{\Theta}\mathbb{E}_{\epsilon}F(\Theta + \sigma\epsilon) = \frac{1}{n\sigma}\sum_{i=1}^{n} F(\Theta + \sigma\epsilon_{\text{i}})\epsilon_{\text{i}}, \tag{3.28}$$

where the return is the sum of the rewards and not a discounted return as in the other algorithms.

One advantage of this algorithm is that the individual parameter vectors are evaluated independently. Therefore, the individual workers can be executed in parallel, so that the training can be accelerated significantly. Furthermore, only the scalar returns have to be passed, since the parameter vectors are known. In algorithm 3 the pseudocode of the training of the ES agent can be seen.

Algorithm 3 Parallelized Evolution Strategies [22]

```
Input: learning rate $\alpha$, noise standard deviation $\sigma$, initial policy parameters $\Theta_0$
Initialize: n workers with random seeds, and initial parameter $\Theta_0$
for $k = 1, ..., N$ do
    for each worker $i = 1, ..., n$ do
        Sample $\epsilon_\mathrm{i} \sim \mathcal{N}(0, I)$
        Compute returns $F_\mathrm{i} = F(\Theta_k + \sigma\epsilon_\mathrm{i})$
    end for
    Send all scalar returns $F_\mathrm{i}$ from each worker to every other worker
    for each worker $i = 1, ..., n$ do
        Reconstruct all perturbations $\epsilon_\mathrm{i}$ for $j = 1, ..., n$ using random seeds
        Set $\Theta_{k+1} \leftarrow \Theta_k + \alpha\frac{1}{n\sigma}\sum_{j=1}^{n} F_\mathrm{j}\epsilon_\mathrm{j}$
    end for
end for
```

3.3 Implementation of the RL agent

For the implementation of the RL agents presented before, the Python toolbox Ray RLlib is used. This toolbox offers predefined versions of the algorithms. A detailed documentation is available on the homepage [23]. The toolbox is designed to train RL agents for a gym environment. To initialize a trainer the corresponding environment and a collection of hyperparameters of the given algorithm are passed. The selection of hyperparameters is discussed in more detail in chapter 6.

Other important aspects of implementing an RL agent is the selection of the states that are passed. On the one hand, all required information should be available for the agent, on the other hand, the number of states should not be too large in order to simplify the training. In this work, four different configurations are compared in the context of a hyperparameter optimization.

In addition, a reward function needs to be specified. In this case it is a torque control, so the deviation of the reference torque from the momentary value should be the determining factor in the reward function. At this point it has to be mentioned that the torque is usually not measured during the operation of a SCIM. However, it is possible to measure the torque during the training process of the agent by an additional sensor, which is attached to the motor. Since the reward function is only needed for training, this is no problem for defining the reward function. The reward of an RL agent should be maximized during training, so

$$r_k = 1 - |\tilde{T}_k - \tilde{T}_{\mathrm{ref}_k}| \tag{3.29}$$

is initially used as the reward function for training, where the torque and the reference are in the normalized form.

Besides following the reference, the selected current operating point of the motor should have the highest possible efficiency. To guarantee that the tracking of the reference is always in the focus, the reward is only extended by an additional term, that depends on the efficiency, if the torque is in a certain range around the reference torque. To define the size of this range, the torque changes while changing the input voltages are taken into account. Since a discrete action space is used here, the change in torque must be taken into account for the entire supply voltage range. As can be seen from equation (2.23), the torque depends on the rotor flux and the stator current in q-direction. The rotor flux is integrated over time and therefore changes slower than the i_{sq} current. The change in stator current depends primarily on the voltage and the stator stray-inductance. Starting from the voltage equation

$$u = L\frac{\mathrm{d}i}{\mathrm{d}t} \tag{3.30}$$

an inductor, the change of the current i_{sq} in one time step can be calculated as

$$\Delta i_{\mathrm{sq}} = \frac{u_{\mathrm{sup}}}{\sqrt{3}}\frac{\Delta t}{L_{\sigma_{\mathrm{s}}}} = \frac{380\,\mathrm{V}}{\sqrt{3}}\frac{50\,\mathrm{\mu s}}{19.4\,\mathrm{mH}} = 0.565\,\mathrm{mA}. \tag{3.31}$$

With a maximum current of 9.15 A, the selected range of 10 % is a bit wider. If the torque is within the 10 % range of the reference torque, the term

$$r_{\mathrm{eff}} = c_{\mathrm{eff}}\left(1 - \frac{|P_{\mathrm{in}} - P_{\mathrm{out}}|}{|P_{\mathrm{in}}| + |P_{\mathrm{out}}|}\right) \tag{3.32}$$

is added to the reward, where c_{eff} is a constant weighting factor. To prevent states of the system from exceeding the limits, the reward is set to -1 in that case.

4 Flux Estimator

As shown in chapter 3, an RL agent requires knowledge of all system states. Since the magnetic flux is part of the system state of the SCIM, but is usually not measured, at least an estimate of the magnetic flux is needed. An estimate of the rotor flux is also used for an FOC, but this is obtained using a flux observer, which requires knowledge of the motor parameters. Consequently, it is not possible to apply one of the well-known structures of a flux observer. In the following, three different approaches for estimating the magnetic flux are discussed and compared afterwards.

4.1 Lowpass integrator

Typically, the rotor flux of the SCIM is estimated for the FOC by utilizing a flux observer. Nevertheless, both the differential equations of the system and the torque equation can be represented equivalently to the formulation with the rotor flux as well as with the stator flux. Even when transforming the currents and voltages into the flux-oriented coordinate system, it is irrelevant whether the angle of the rotor flux or the stator flux is used. Therefore, the stator flux also contains all the relevant information for the state of the system.

The magnetic flux is building up over time, which can be described mathematically by an integration. The derivative of the stator flux is given by the equation

$$\underline{\dot{\mathit{\Psi}}}_{\mathrm{s}}^{\mathrm{s}} = \underline{u}_{\mathrm{s}}^{\mathrm{s}} - R_{\mathrm{s}}\underline{i}_{\mathrm{s}}^{\mathrm{s}}. \tag{4.1}$$

The voltage $\underline{u}_{\mathrm{s}}^{\mathrm{s}}$ is the clamping voltage of the motor, which is also the manipulated variable and is therefore known. The second term of the derivative describes the ohmic voltage drop. The stator current is measured, but the rotor resistance is unknown, therefore the value for the ohmic voltage drop cannot be determined.

The ohmic resistance of the winding should be as small as possible for efficiency reasons, which is why the ohmic voltage drop is normally significantly smaller than the input voltage. As a result the ohmic voltage drop in the derivative can be neglected in a first approximation. So an estimation of the magnetic stator flux can be obtained by integrating the input voltages. Since the control is performed by a digital controller, the integration is approximated by a summation.

An issue with open integration of an approximated derivative is a potentially unstable output value. To avoid this problem, a lowpass integration is used, which leads to a further difference of the estimated values from the actual flux in phase and amplitude, but prevents the estimated value from oscillating. The estimated value for the stator flux in $\alpha\beta$ coordinates is

$$\hat{\varPsi}^{\mathrm{s}}_{\alpha\beta}[k] = \tau \mathbf{u}^{s}_{\alpha\beta}[k] + c_{\mathrm{lp}}\, \hat{\varPsi}^{\mathrm{s}}_{\alpha\beta}[k-1] \tag{4.2}$$

where c_{lp} is a constant between 0 and 1. Figure 4.1 shows a block diagram of the flux estimator.

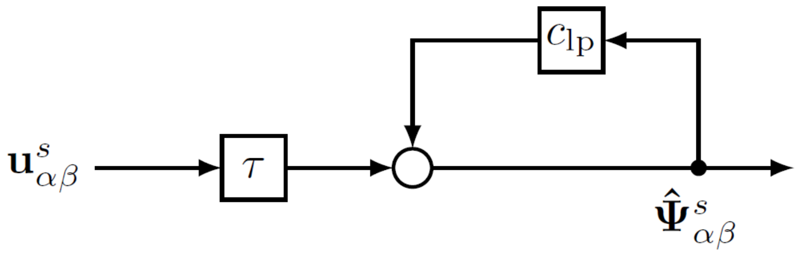

$\mathbf{u}^{s}_{\alpha\beta}$
τ
c_{lp}
$\hat{\mathbf{\Psi}}^{s}_{\alpha\beta}$

Fig. 4.1: Block diagram of the lowpass integrator flux estimator

4.2 RL estimator

Since the motor parameters are unknown, an RL agent is used to control the motor. This is a model free approach for learning a policy. A similar problem arises when estimating the magnetic flux, since the well known flux observers require the motor parameters. Therefore, an approach for estimating the flux with an RL agent can also be used for this task. This RL estimator is trained in parallel to the training of the RL agent for the control of the system. An advantage of using an RL estimator is that nonlinearities can be learned and make the estimates more flexible in general.

First, a suitable RL algorithm must be selected for this task. For the estimation of the magnetic flux, the measured values of the relevant quantities from the differential equation of the flux are passed to the agent, which are continuous. In contrast to the RL controller, the outputs of the RL estimator are also continuous, which is why, for example, the DQN algorithm discussed earlier and the Rainbow algorithm are not applicable. In this case, a Deep Deterministic Policy Gradient (DDPG) algorithm is chosen because it is a well-established RL algorithm for the control of technical systems with a continuous state and action space.

A DDPG agent consists of two separate ANNs. On the one hand, a network is trained as in the DQN and Rainbow algorithm so that the q-function is estimated, known as the critic network. On the other hand, a policy, respectively the actor, is trained separately. The training of the critic is similar to that of the DQN algorithm, while the outputs of the actor are trained to maximize the q-values computed by the critic. The two networks are trained in parallel, but the learning process of the actor requires a well-estimated q-values, in other words, a well-trained critic. Because of that, a higher learning rate is often used for the updates of the network weights of the critic than for the actor. As the name of the algorithm implies, the training is performed with the help of a gradient ascent method. The algorithm is an off-policy algorithm, so a replay buffer is also used. The exploration is implemented by a noise process, which is superimposed on the outputs of the actor. A detailed description of the DDPG algorithm can be found at [24].

When calculating the estimated values of the magnetic flux, the derivative needs to be estimated and integrated. To simplify the task of the RL estimator, the integration is performed explicitly after the estimation, so that the RL estimator only has to find the derivative of the flux. As in the lowpass estimator discussed previously, this is an open integration, thus a lowpass is also used for this integration.

One of the most important components in training an RL agent is the definition of the reward function. Since the magnetic flux is not measured, the agent must be evaluated in a different way. Considering the torque equation (2.22) in $\alpha\beta$ coordinates, we obtain a function of the torque

$$T = c(\Psi_{\mathrm{r}\alpha} i_{\mathrm{s}\beta} - \Psi_{\mathrm{r}\beta} i_{\mathrm{s}\alpha}) \tag{4.3}$$

depending on the measured currents and the constant $c = \frac{3}{2}p\frac{L_\mathrm{m}}{L_\mathrm{r}}$. If the deviation of the measured torque from the estimator of the torque obtained by equation (4.3) is used as a reward function, where the magnetic fluxes are the outputs of the RL estimator, the problem arises that the constant c is unknown. The pole pair number is a motor parameter, but it can be assumed that it is known, because it can be exactly identified by counting. Thus, only the ratio of the two inductances $\frac{L_\mathrm{m}}{L_\mathrm{r}}$ is missing, but for physical reasons it is smaller than 1. If this reward function is used, the estimated values for the magnetic fluxes are reduced by this factor compared to the true values. This constant deviation is not a problem, since a data-based method is used for the control of the motor. It is only important that the angle between the two flux components is estimated correctly, otherwise a possible coordinate transformation into the flux-oriented coordinate system would be incorrect. This condition is fulfilled because the multiplicative factor is constant and positive. For these reasons,

$$R_\mathrm{est} = 1 - |\tilde{T} - \frac{\frac{3}{2}p(\hat{\Psi}_{\mathrm{r}\alpha} i_{\mathrm{s}\beta} - \hat{\Psi}_{\mathrm{r}\beta} i_{\mathrm{s}\alpha})}{T_\mathrm{lim}}| \tag{4.4}$$

is the reward function of the training of the RL estimator. Figure 4.2 shows the whole control structure including the RL estimator.

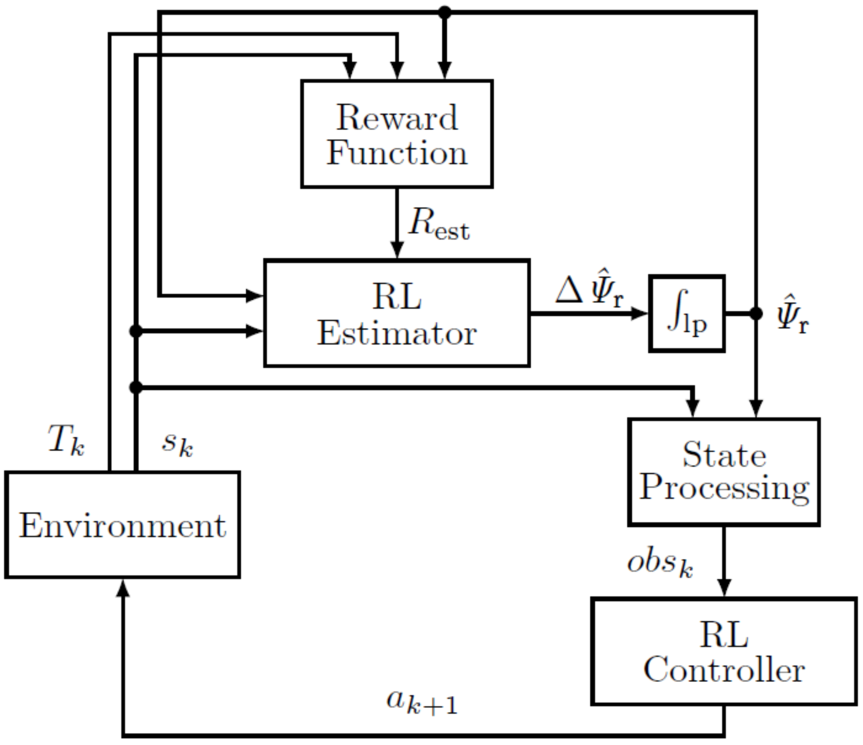


Fig. 4.2: Control structure including the RL estimator

4.3 Derivative identification

Another way to estimate the magnetic flux is to identify the differential equations of the flux, which are given by the equations (2.12) and (2.13). With the knowledge of the derivatives of the two magnetic flux components, these can be integrated to get an estimate value. In contrast to the other two methods, no lowpass integration is required, because there is a feedback of the respective flux component in the derivative.

For the identification of the parameters the torque equation

$$T = \frac{3}{2} p \frac{L_{\text{m}}}{L_{\text{r}}} (\Psi_{\text{r}\alpha} i_{\text{s}\beta} - \Psi_{\text{r}\beta} i_{\text{s}\alpha}) \tag{4.5}$$

is taken into account. The magnetic fluxes are integrated as a function of the motor parameters. With the help of these fluxes, the torque can be determined, also as a function of the motor parameters. By minimizing the error between the measured torque and the estimated torque, the motor parameters are identified. When estimating the motor parameters, some parameters can be combined so that the optimization problem is reduced and simplified. This results in the torque equation

$$T_{\text{est}}[k] = \frac{3}{2} p c_3 (\Psi_{\text{r}\alpha}[k] i_{\text{s}\beta}[k] - \Psi_{\text{r}\beta}[k] i_{\text{s}\alpha}[k]) \tag{4.6}$$

and the recursive equations

$$\Psi_{\mathrm{r}\alpha}[k+1] = \Psi_{\mathrm{r}\alpha}[k] + \tau(c_1 c_3 i_{\mathrm{s}\alpha}[k+1] - c_2\,\Psi_{\mathrm{r}\alpha}[k] - p\omega_{\mathrm{me}}\,\Psi_{\mathrm{r}\beta}[k]) \tag{4.7}$$

and

$$\Psi_{\mathrm{r}\beta}[k+1] = \Psi_{\mathrm{r}\beta}[k] + \tau(c_1 c_3 i_{\mathrm{s}\beta}[k+1] + p\omega_{\mathrm{me}}\,\Psi_{\mathrm{r}\alpha}[k] - c_2\,\Psi_{\mathrm{r}\beta}[k]) \tag{4.8}$$

for the magnetic fluxes, which depend on the parameters $c_1 = R_\mathrm{r}$, $c_2 = \frac{1}{\tau_\mathrm{r}}$ and $c_3 = \frac{L_\mathrm{m}}{L_\mathrm{r}}$, where again the pole pair number can be assumed to be known. When considering the estimated parameters, boundaries can be specified, which have to be considered when solving the minimization problem. Due to physical reasons, all parameters are greater than 0. Furthermore, the parameter c_3 is always less than 1 and greater than 0.5 since $L_{\sigma_\mathrm{r}} < L_\mathrm{m}$. Therefore, the minimization problem

$$\min_{c_1,c_2,c_3} \sum_{k=0}^{n} (T[k] - T_{\mathrm{est}}[k])^2 \tag{4.9}$$

$$\text{with } c_1 > 0,\ c_2 > 0,\ 0.5 < c_3 < 1 \tag{4.10}$$

results.

This minimization problem is solved for each episode using a minimization algorithm. To obtain the estimated values for the parameters, the average of the parameters of all episodes is calculated. This average value can be used as the starting value for the minimization in each episode. So only at the beginning initial values for the parameters have to be selected. The parameter c_1 is the rotor resistance R_s. For technical reasons c_1 will be greater than 0, which is why 1 is chosen as the initial value. The parameter c_2 shows a similar condition, but it is usually a bit larger, so 2 is set as the initial value. About the parameter c_3 there is more known, since L_m is significantly larger than L_{σ_r}, it results that c_3 is slightly smaller than 1, so 0.9 is used as the initial value.

In the figures 4.3, 4.4, and 4.5, the trajectories of the estimated parameters for 100 episodes with a length of 0.1 s are shown. The input voltages of the motor were randomly drawn from the action space of the inverter and the speed of the motor is also random. These conditions correspond to those at the beginning of the training process of an RL agent.

It seems that there is a small systematic error in the estimation of c_1, but it is less than 1 %. Furthermore, the error can be influenced by an incorrect estimation of the other parameters. It is remarkable that a very accurate estimate is available after only a few time steps.

On the other hand, the deviation of the parameter c_2 is already more significant. The parameter c_2 describes the feedback of the flux on the derivative, because a small value can lead to an oscillation of the estimated value, similar to the open loop integration. Therefore, it is expectable that this value tends to be overestimated. Moreover, it should be noted that the magnetic flux is low for random input voltages, which is why in particular this value is difficult to identify at the beginning. This observation is also valid for all

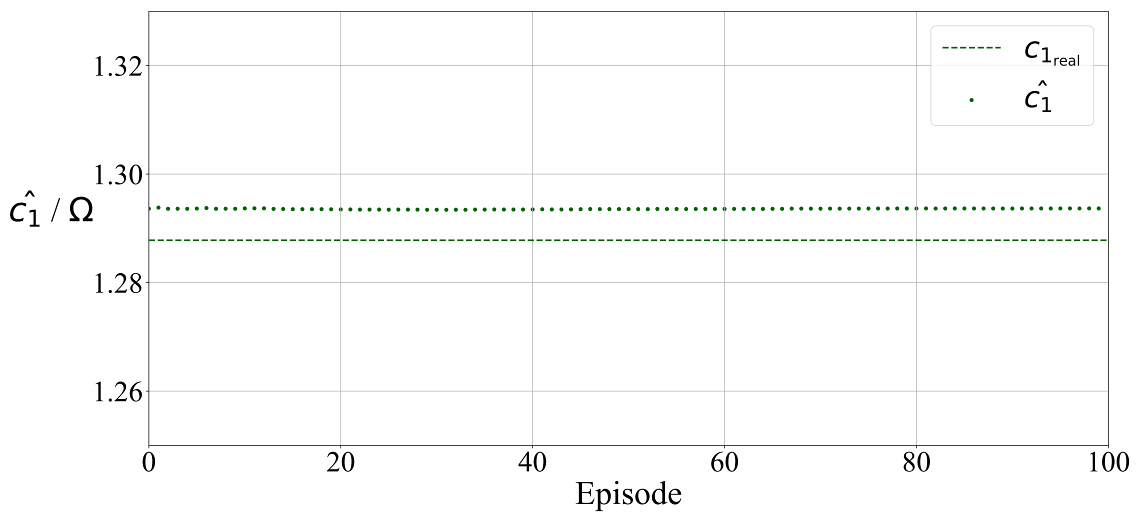


Fig. 4.3: Trajectory of the parameter c_1 for 100 episodes

parameters in general, because the calculated flux and therefore the torque are small at first, which makes the identification of the parameters more difficult. In the later training, it is expected that the magnetic flux will be much larger, which should improve the estimation of the parameters.

The estimation of the parameter c_3 shows a similar behavior as c_1, there is a slight deviation, but this is below 0.3 %. In general, the estimations are very accurate after only a few time steps, even though the low magnetic flux. On the other hand, the system is simulated here with linear parameters and exact measured values, which is why the estimated values on the real test bench are expected to fluctuate significantly stronger.

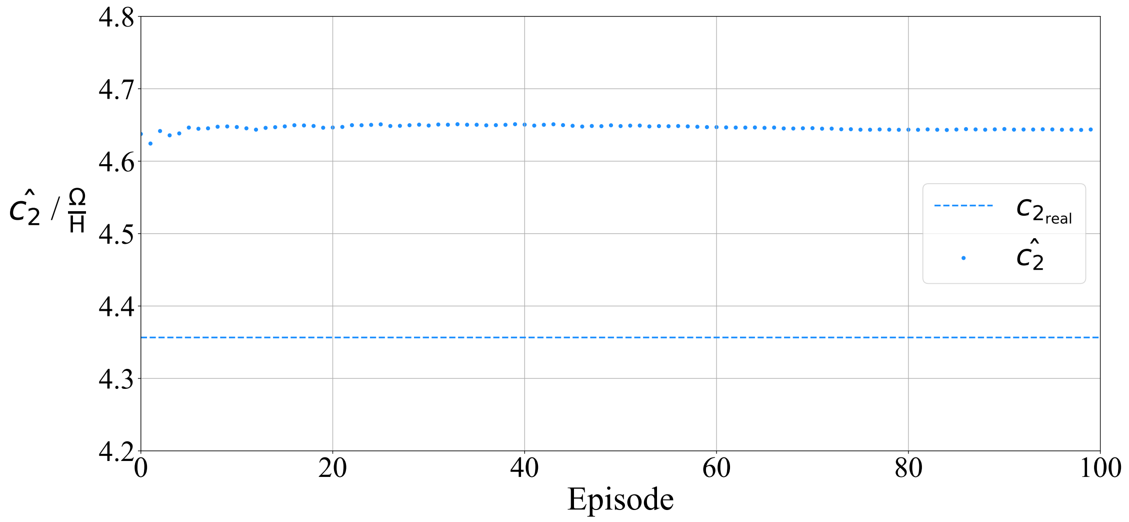


Fig. 4.4: Trajectory of the parameter c_2 for 100 episodes

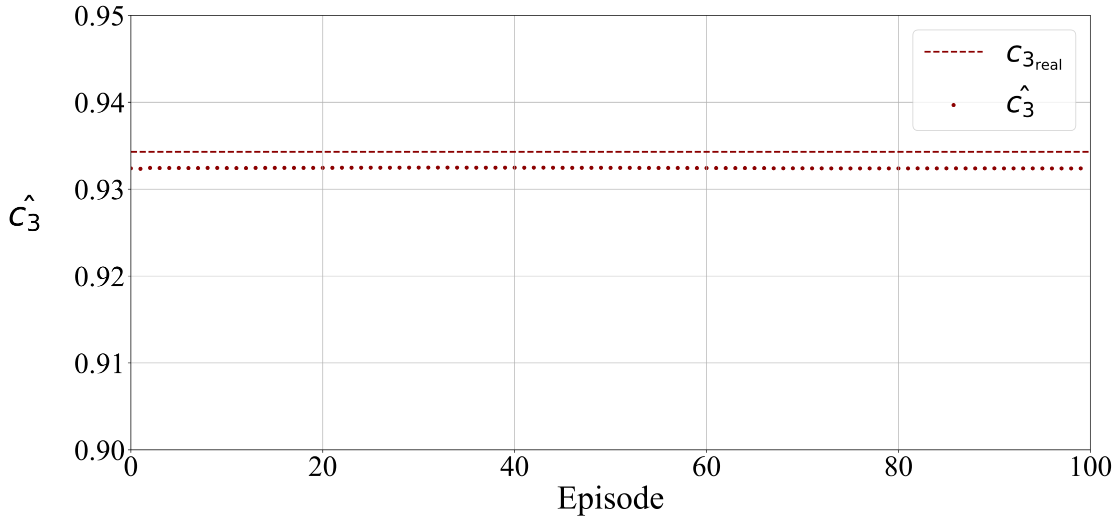


Fig. 4.5: Trajectory of the parameter c_3 for 100 episodes

4.4 Comparison of the flux estimators

The three different estimators are compared in the following in order to select one for the training of the agents afterwards. For comparison, an RL estimator was previously trained in parallel with an agent. The estimator with the identified derivative uses the parameters determined in the previous section. First, the performance of the estimators are evaluated by comparing the estimated values of the α and β flux with the simulated values. Since the lowpass integrator estimates the stator flux and the RL estimator is also magnified by a constant factor, this deviation is compensated with the knowledge of the motor parameters for the comparison. Figure 4.6 shows the graphs of the estimated values for the α flux of the different estimation methods. Also, the simulated magnetic flux can be seen. The graph for the corresponding β flux shows no qualitative differences.

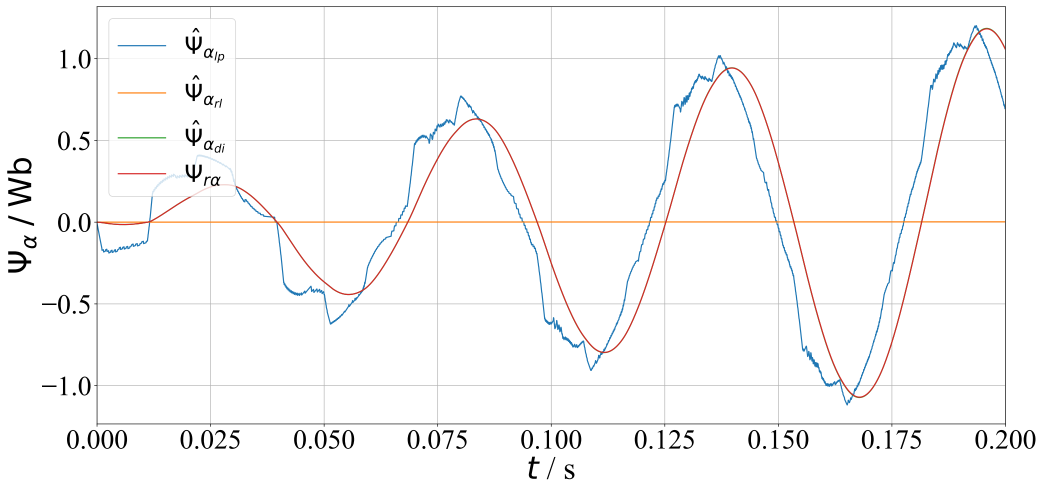


Fig. 4.6: Comparison of the estimated α fluxes

The red course shows the simulated flux, which covers most part of the estimated flux of the integrated identified derivative in green. Although the estimated values of the parameters of the derivative were still partially inaccurate, the estimator already generates very precise values. Therefore, it can be assumed that already at the beginning of the training accurate estimated values for the flux are available and the training of the agent is not slowed down by the parallel identification.

The estimated values of the lowpass integration are shown in blue. It can be seen that the estimated values generally follow the real flux, even though there are larger deviations in some regions. A slight deviation was expected, since the ohmic voltage drop was neglected when integrating the derivative of the stator flux. This has a higher influence at some points than at others, as for example at the beginning of the episode. Furthermore, it can be seen that the estimated value changes stepwise. This is caused by the discrete voltage vectors, since the derivative can only have seven different values. As expected, a phase shift between the estimated value and the simulated value can be observed due to the lowpass filtering.

Lastly, the estimated value of the RL estimator is plotted in orange. Obviously, the RL agent failed to train an adequate policy to estimate the value. Because the estimates for the two components have a direct influence on one component of the reward function. It could be that the agent maximized the reward, but the internal behavior of the magnetic fluxes is not correctly represented. Another problem with the estimation is that due to the structure of the derivative, the estimated fluxes themselves are inputs of the agent. If these estimated values are not accurate enough, the agent will be trained with incorrect data and the next estimated values will be inaccurate as well.

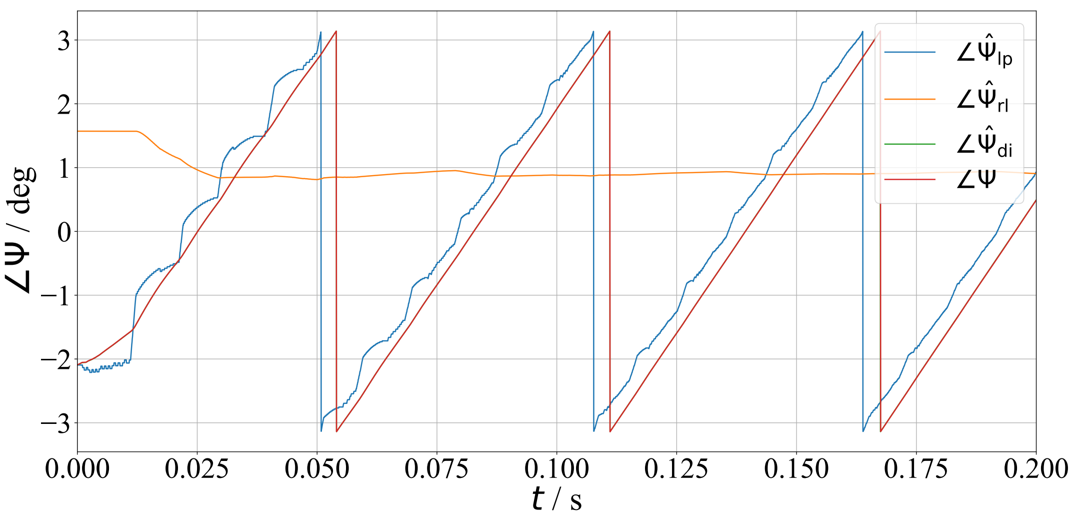


Fig. 4.7: Comparison of the angels of the estimated fluxes

Figure 4.7 shows the angles of the estimated and simulated magnetic fluxes. These angles are of particular relevance, because a deviation of the angle in a potential variable transformation into the flux-oriented coordinate system can lead to large deviations of these quantities. The graphs confirm the observations made in the previous plot. The integration of the estimated derivative returns very accurate estimates of the angle of the fluxes, while the angle of the lowpass integrator varies slightly and is phase-shifted. The angles of the RL estimator are not adequate, as expected.

For the later training of the RL controller, the integration of the estimated derivative is used as the estimation method. However, it remains to be seen whether the excellent results can be reproduced on a real motor, since nonlinearities and measurement noise will make it more difficult to identify the parameters. Alternatively, the lowpass integration can be tested there and the performance then compared again.

5 Model Predictive Safety Layer

When controlling a motor, the system limits have to be considered, because exceeding these limits can lead to destruction of the motor. An RL agent, however, only implicitly considers these limits by the reward function, which is why a violation of the limitations, especially during training, but also in later operation is not guaranteed to be avoided. For this reason, a safety layer must be used, which decides whether an action is safe or leads to an exceeding of the limits. Such a safety layer is based on a system model that can be used to perform a one-step prediction, which is why it is labeled model predictive safety layer. Due to the unknown motor parameters, a model of the system has to be identified first. Finally, the safety layer has to be integrated into the training process of the different RL algorithms.

5.1 Model identification

For the identification of the motor parameters of a SCIM exist several approaches. For example, in [25] a method is presented that identifies the motor parameters using a nonlinear least squares estimator. Another approach from [26] uses a particle swarm optimization to find the parameters of the motor. Both of these methods have in common that they estimate all the motor parameters without the knowledge of the magnetic flux.

Since, as shown in chapter 4, a valid estimate for the rotor flux is also already available in an early phase of the training, it can be used for the model identification. Furthermore, the model that is identified can be simplified in the fact that not all individual motor parameters have to be estimated, just the behavior of the complete system. Knowledge about the behavior is sufficient to perform a one-step prediction and decide whether the system limits are exceeded. In [27], a method is presented that identifies the behavior of a PMSM for an MPC, by using a recursive least squares (RLS) estimator. The method can be adapted to the SCIM, since the knowledge of the magnetic flux can be assumed.

First of all, the structure of the model has to be defined. For this the differential equations (2.10) to (2.13) of the system are considered. Since an estimate of the flux is already available, the two differential flux equations do not need to be identified. The two remaining differential equations can be formulated in the matrix notation and the next states can be calculated by using the Euler forward method. Thus the system for the calculation of the next states is specified as

$$\begin{bmatrix} i_{\mathrm{s}\alpha} \\ i_{\mathrm{s}\beta} \end{bmatrix}[k+1] = \begin{bmatrix} 1-\frac{\tau}{\tau_\sigma} & 0 & \frac{R_\mathrm{r} L_\mathrm{m} \tau}{\sigma L_\mathrm{r}^2 L_\mathrm{s}} & p\omega_\mathrm{me}\tau\frac{L_\mathrm{m}}{\sigma L_\mathrm{r} L_\mathrm{s}} \\ 0 & 1-\frac{\tau}{\tau_\sigma} & -p\omega_\mathrm{me}\tau\frac{L_\mathrm{m}}{\sigma L_\mathrm{r} L_\mathrm{s}} & \frac{R_\mathrm{r} L_\mathrm{m} \tau}{\sigma L_\mathrm{r}^2 L_\mathrm{s}} \end{bmatrix} \begin{bmatrix} i_{\mathrm{s}\alpha} \\ i_{\mathrm{s}\beta} \\ \Psi_{\mathrm{r}\alpha} \\ \Psi_{\mathrm{r}\beta} \end{bmatrix}[k] + \begin{bmatrix} \frac{\tau}{\sigma L_\mathrm{s}} & 0 \\ 0 & \frac{\tau}{\sigma L_\mathrm{s}} \end{bmatrix} \begin{bmatrix} u_{\mathrm{s}\alpha} \\ u_{\mathrm{s}\beta} \end{bmatrix}[k]. \tag{5.1}$$

The rotational speed ω_me is not a constant, so this system is time-variant. On the other hand, ω_me is measured, so the state can be extended by the multiplication of the rotational speed with the magnetic fluxes. Furthermore, the parameters to be estimated are equal in magnitude for the α and β components of the current. Therefore, it would be sufficient to identify the parameters for one of the two components and transfer them to the other component. In the following, all parameters are still identified individually, but separated by the components, so that the equations

$$i_{\mathrm{s}\alpha}[k+1] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \end{bmatrix} \begin{bmatrix} i_{\mathrm{s}\alpha} \\ \Psi_{\mathrm{r}\alpha} \\ \omega_\mathrm{me}\Psi_{\mathrm{r}\beta} \end{bmatrix}[k] + \begin{bmatrix} b_{11} \end{bmatrix}\begin{bmatrix} u_{\mathrm{s}\alpha} \end{bmatrix}[k] \tag{5.2}$$

$$i_{\mathrm{s}\beta}[k+1] = \begin{bmatrix} a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} i_{\mathrm{s}\beta} \\ \Psi_{\mathrm{r}\beta} \\ \omega_\mathrm{me}\Psi_{\mathrm{r}\alpha} \end{bmatrix}[k] + \begin{bmatrix} b_{21} \end{bmatrix}\begin{bmatrix} u_{\mathrm{s}\beta} \end{bmatrix}[k] \tag{5.3}$$

result, where

$$\begin{aligned} a_{11} &= a_{21} = 1-\tfrac{\tau}{\tau_\sigma} \\ a_{12} &= a_{22} = \tfrac{R_\mathrm{r} L_\mathrm{m} \tau}{\sigma L_\mathrm{r}^2 L_\mathrm{s}} \\ a_{13} &= -a_{23} = p\tau\tfrac{L_\mathrm{m}}{\sigma L_\mathrm{r} L_\mathrm{s}} \\ b_{11} &= b_{21} = \tfrac{\tau}{\sigma L_\mathrm{s}} \end{aligned} \tag{5.4}$$

are the parameters, that have to be identified.

For the identification of the parameters an RLS method is applied, which is a recursive form of the ordinary least squares (OLS) method. In contrast to the OLS, where the parameters are estimated offline based on a complete data set, the RLS updates the parameters online in every time step. The OLS estimator minimizes the squared error between the measured value and the estimated value of the model. One problem of this method is that with increasing number of data points the calculation takes more and more time. Another problem with respect to the use of the parameter identification in parallel to the training process of the RL controller is that the estimated values are not available until all data points have been collected. This problem is avoided in the RLS where the estimated values for the parameters are optimized progressively.

The RLS estimator starts with an initial parameter vector θ, which contains all parameters to be estimated. By multiplying this parameter vector with the regressor vector ξ, which consists of the state and the input variables of the system, an estimate for the next state is calculated. Subtracting this estimate from the measured next state φ results in the residuals

$$e = \varphi - \xi^{\mathrm{T}}\theta, \tag{5.5}$$

which indicate the estimation error. Transferred to the SCIM, the estimation of the parameters of the α component results in the variables

$$\theta_{\alpha}[k] = \left[a_{11}\; a_{12}\; a_{13}\; b_{11}\right]^{\mathrm{T}}, \tag{5.6}$$

$$\xi_{\alpha} = \left[i_{\mathrm{s}\alpha}\;\; \Psi_{\mathrm{r}\alpha}\;\; \omega_{\mathrm{me}}\Psi_{\mathrm{r}\beta}\;\; u_{\mathrm{s}\alpha}\right]^{\mathrm{T}} \tag{5.7}$$

and

$$\varphi_{\alpha} = i_{\mathrm{s}\alpha}. \tag{5.8}$$

The variables for the β component are obtained analogously.

An update of the parameter vector is performed in three steps. First, a correction vector

$$\gamma[k] = \frac{\boldsymbol{P}[k]\xi[k+1]}{\lambda + \xi^{\mathrm{T}}[k+1]\boldsymbol{P}[k]\xi[k+1]} \tag{5.9}$$

is calculated for the parameters, which is used in the second step, where the parameter vector is updated. The parameter vector

$$\hat{\theta}[k+1] = \hat{\theta}[k] + \gamma[k](\varphi[k+1] - \xi^{\mathrm{T}}[k+1]\hat{\theta}[k]) \tag{5.10}$$

is adjusted by adding the weighted residuals to the momentary parameter vector. Finally, the matrix

$$\boldsymbol{P}[k+1] = \frac{1}{\lambda}(\mathbf{I} - \gamma[k]\xi^{\mathrm{T}}[k+1])\boldsymbol{P}[k] \tag{5.11}$$

is updated, which is proportional to the covariance matrix of the parameter vector θ. The parameter λ, which has to be between 0 and 1, can be used to adjust an exponential forgetting of the previous parameter vectors. A detailed explanation of the three update rules can be found in [28].

An important aspect of the identification of the parameters is the initialization of the parameter vector θ and the $\boldsymbol{P}$ matrix. Since the motor parameters can vary a lot depending on the motor and therefore no reasonable initial values can be assumed for the parameters, the unit vector is used as initialization for the parameter vector θ. The matrix $\boldsymbol{P}$ is proportional to the covariance matrix of the parameter vector, so high values indicate high uncertainty in the estimation of the parameters. Because in this case there is no prior knowledge about the parameters, the entries of the $\boldsymbol{P}$ matrix should be initialized with high values. This has the consequence that the estimated values become more accurate, because the initial values are considered less trustworthy. Furthermore, it is assumed that the parameters are uncorrelated to each other, so that all entries outside the principal diagonal can be initialized with 0. Therefore, $\boldsymbol{P}[0] = \mathbf{I}\alpha$ with $\alpha = 50$ is used as initialization.

To test the identification of the motor model, first random actions are switched on the drive system, like it is at the beginning of the training. The states of the system are measured and the magnetic flux is estimated by the method explained in chapter 4. In each time step the motor model is updated according to the applicable rules. The following graphs show the trajectories of the estimated parameters of the α component, at a number of 100,000 steps. In addition, the true values of the parameters are plotted, which were calculated using the motor parameters from table 2.1. The graphs of the parameters of the β component, are quite similar, as expected, since the same parameters are estimated. The corresponding figures can be found in the appendix.

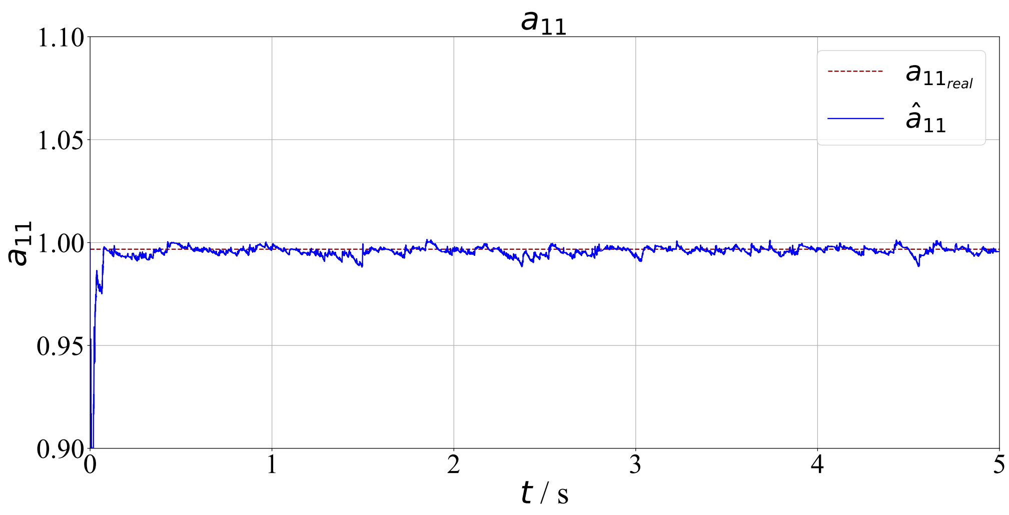


Fig. 5.1: Trajectory of the estimated parameter $\hat{a}_{11}$

The graph in figure 5.1 shows the trajectory of the estimated value of the parameter a_{11}, which indicates the influence of the present $i_{s\alpha}$ current on the following current. The true value of a_{11} is 0.997, which is very close to the initial value 1. Nevertheless, it can be seen that the estimated value increases significantly at the beginning, which is probably due to the high values of the $\boldsymbol{P}$ matrix, because these indicate a high uncertainty of the estimate. In the following procedure, the estimated value oscillates around the actual value, but cannot stabilize.

Figure 5.2 plots the progress of the estimated parameter a_{12}. This describes the effect of the magnetic flux Ψ_α on the next current. As in the previous plot, it can be noticed that the estimated value initially shows a very large fluctuation, in this case downwards. Subsequently, the estimated value oscillates around the true value with an amplitude that exceeds the true value by a large factor. In the progress of time, this amplitude decreases only slightly. The difficulty at this point is probably that, on the one hand, the influence of the magnetic flux is very small and, on the other hand, the estimated magnetic flux is also small, as no significant flux can build up due to the rapidly changing input voltage.

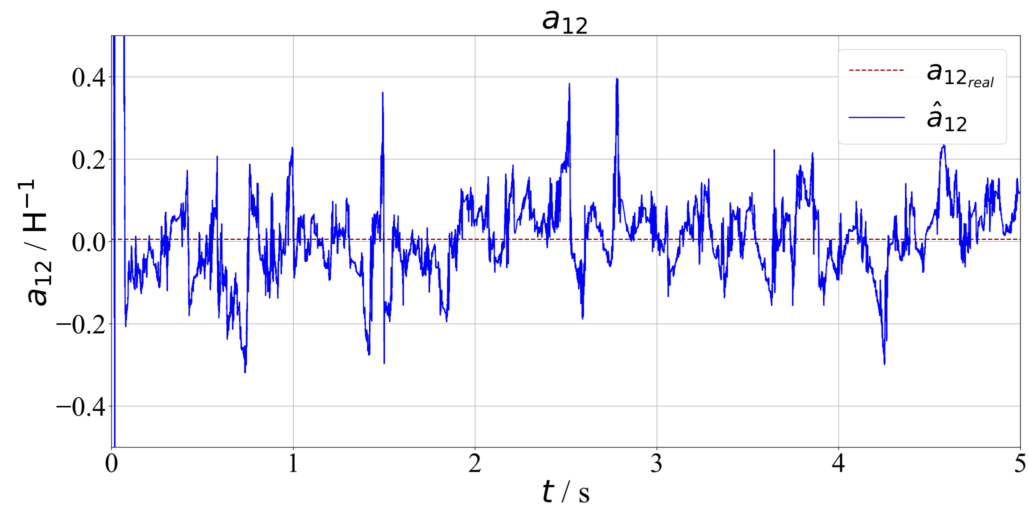


Fig. 5.2: Trajectory of the estimated parameter $\hat{a}_{12}$

The high uncertainty of the estimated value can also be seen in the corresponding entry of the $\boldsymbol{P}$ matrix, since this is by far the largest value.

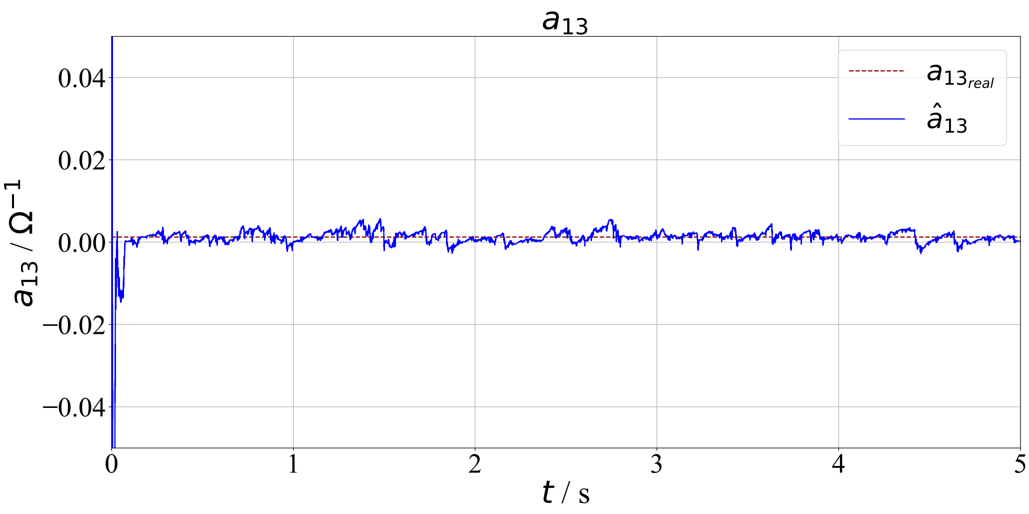


Fig. 5.3: Trajectory of the estimated parameter $\hat{a}_{13}$

In contrast, the estimation of the parameter a_{13} is much more precise, which can be seen in figure 5.3, although it is also describing the dependence on the complementary magnetic flux. Since the entry of the regressor is additionally multiplied by the velocity ω_{me}, the value is a much higher, so it is easier to estimate this parameter. The estimated value for a_{13}, however, still oscillates with a high amplitude relative to the actual value.

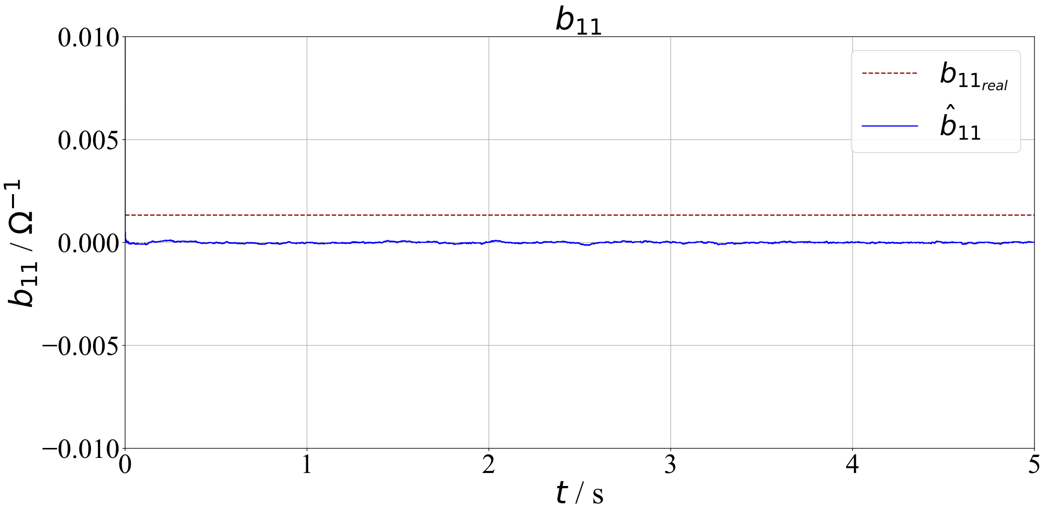


Fig. 5.4: Trajectory of the estimated parameter $\hat{b}_{11}$

The graph in figure 5.4 shows that the parameter b_{11} was obviously not identified adequately. The estimate of this value converges very soon to 0, although the actual value is 0.0013 $\frac{1}{\Omega}$. This is a very small value, but the value does not fluctuate greatly, so it is not expected to stabilize at the optimal value. An explanation for the behavior of the estimator could be again the selection of the input voltages, since this parameter specifies the influences of the input voltage on the next current value. Due to the constant change of the randomly chosen voltage vectors, it may not be possible for the RLS estimator to achieve a correct identification of the parameter.

This inaccurate estimate is problematic for the use of the system model as part of a safety layer, since the safety layer is supposed to predict the following state for the different voltage vectors in order to decide whether the constraints are met. If the influence of the voltage vectors is estimated incorrectly, there is no basis for the decision which action is safe and which is not. This is initially true for the beginning of the training process, since the actions there are chosen randomly. In the later phase of the training, the same actions will be selected more and more frequently in a row, since in this way a current and thus also the torque can be held approximately constant.

In order to evaluate this effect on the identification of the parameters, a second time series is generated, analog to the first one, with the difference that the input voltages are selected by a different metric. A voltage vector is applied to the inputs of the system for 50 consecutive time steps under the new metric, as long as the currents do not exceed a limit of 90 % of the maximum current. In this case, the zero voltage vector is switched on to reduce the currents and prevent them from exceeding the limit. The next voltage vector to be applied after 50 time steps is the counterclockwise next in the voltage hexagon from figure 2.4.

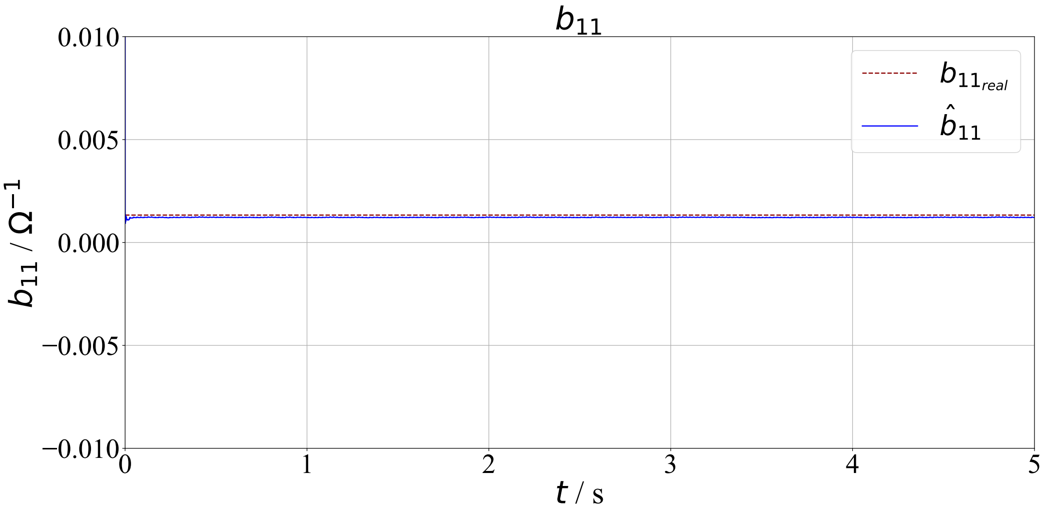


Fig. 5.5: Estimated parameter $\hat{b}_{11}$ selecting a voltage vector for 50 time steps

Figure 5.5 shows the trajectory of the estimated value for parameter b_{11} using the new metric for selecting the actions. As expected, there is a significant improvement compared to the previous estimate. The estimated value converges approximately against the true value, so that this model can be used for a safety layer. The other plots, which can be viewed in the appendix A.1, show that the estimated values oscillate less than before. The estimated values for the parameters a_{12} and a_{22} still fluctuate strongly and this below the real value.

A reason for this could be that the magnetic flux is still too small. Further tests with the same metric, only with a higher number of steps for which a voltage vector is switched on, which should increase the amplitude of the magnetic flux, show that the estimated values for these parameters also nearing the true values at a higher flux. The corresponding plots can also be seen in the appendix A.1. As a result, it has been shown that a system model suitable for the safety layer can be identified in parallel to the training of an RL agent, if the agent has been trained suitable.

5.2 Implementation of the model predictive safety layer

The basis of the safety layer is the system model identified in the previous section. The momentary state variables are measured or respectively estimated, so that these are available for a one-step prediction. The currents predicted in this way can be evaluated to decide which of the actions is allowed to be performed without exceeding the limitations. This step is rather simple, since there are only eight different voltage vectors. For the case of a continuous action space, it would be possible to check whether a chosen action is valid or not, but the set of all valid actions is a continuous space, which is difficult to be calculated. Because, it is necessary for a RL controller to know which actions are allowed to be taken in order to choose the action, a discrete action space is used.

The safety layer checks in a first step by a one step prediction which of the actions are safe by evaluating the equations (5.2) and (5.3). In a second step, the RL agent's policy is reported which of the actions are safe and selects one of these actions. This second step must be adapted to the respective policy of the agent. If the safety layer only forbids an action and independently selects a new action, the agent would be trained with incorrect data, since the agent assumes that the action selected by it has been executed. Thus, the agent has no feedback from the safety layer.

In the paper [29], a safety layer is used for a PPO agent, which computes a mask for the actions so that the unsafe actions cannot be selected by the policy. The probabilities of the actions are multiplied by this mask, so that the probability of unsafe actions are set to 0, while the other probabilities are not changed. This procedure can only be applied to a stochastic policy, which is applicable to the PPO and ES agent.

For the ϵ greedy policy, which the DQN and Rainbow agent use, this procedure is not possible. On the other hand, the computed q-values for the unsafe actions can be set to $-\infty$ so that they are not selected when the action is chosen deterministically. Nevertheless, the problem that the action is randomly selected from the action space during training with probability ϵ remains. In this case, a mask can be placed on the action space in the same way as for the stochastic policy, so that an action is selected only from the action space of safe actions. Figure 5.6 illustrates the structure of the RL agent including the safety layer.

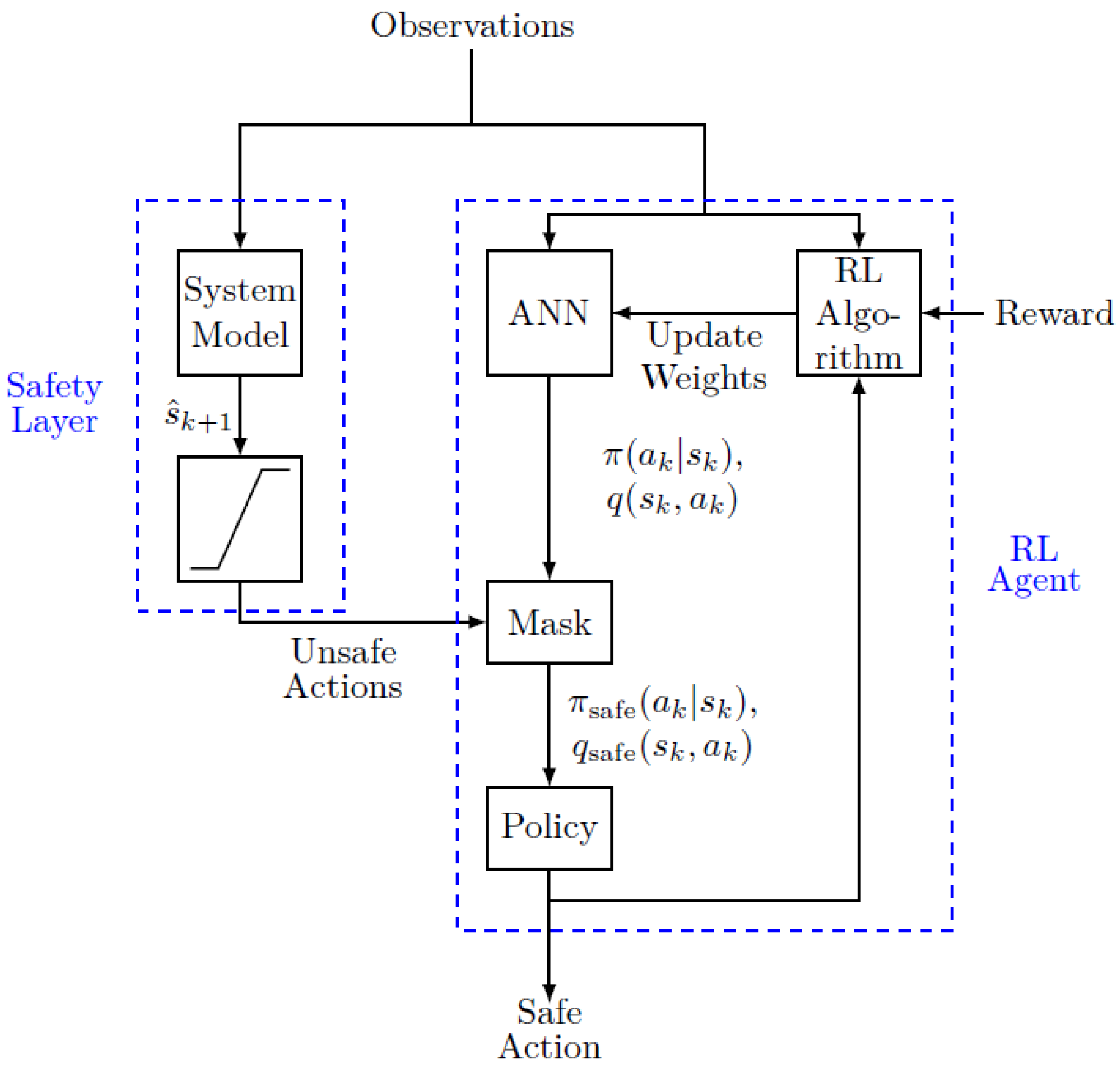


Fig. 5.6: Structure of the RL Agent including the safety layer

6 Hyperparameter Optimization

When using an RL agent, there are many different parameters for adjusting and optimizing the network architecture and the training. For these parameters, which are called hyperparameters, there are no rules or deterministic methods to choose them optimally. Therefore, it can not be avoided to train a large number of agents with different hyperparameters and to compare them. To automate this process, an algorithm specially designed for the optimization of hyperparameters is used, which selects the hyperparameters for a training iteration based on the previous results.

In this case, in addition to the hyperparameters, the RL algorithm itself and the state space should also be optimized, i.e., the state variables which are passed to the RL agent. Four different RL algorithms and state spaces are available for this purpose. Even if these parameters are not hyperparameters in the strict sense, it is no problem to include them in the optimization, because the decision which parameters are selected is only made depending on the results of the previous iterations.

In a hyperparameter optimization, it must be taken into account that the training of an agent also depends on random initialization and, for example, random choice of reference values. For this reason, several agents should be trained for each hyperparameter set and all results should be taken into account when calculating the score of the hyperparameter set. If this is not done, an agent might perform very well for a particular initialization, but not in general, which would mean that this hyperparameter set might perform significantly worse when an agent is trained later on the test bench.

To determine the result of an agent, the reward is measured when controlling a reference trajectory. It is important that this reference trajectory is standardized for all agents, since the reward depends highly on the reference. Furthermore, it should be ensured that the reference trajectory contains as many different operating points as possible. In addition to the reference trajectory, a constant speed is imposed on the motor. This should also be varied during the test. In figure 6.1 a block diagram is shown, which illustrates the procedure of the hyperparameter optimization.

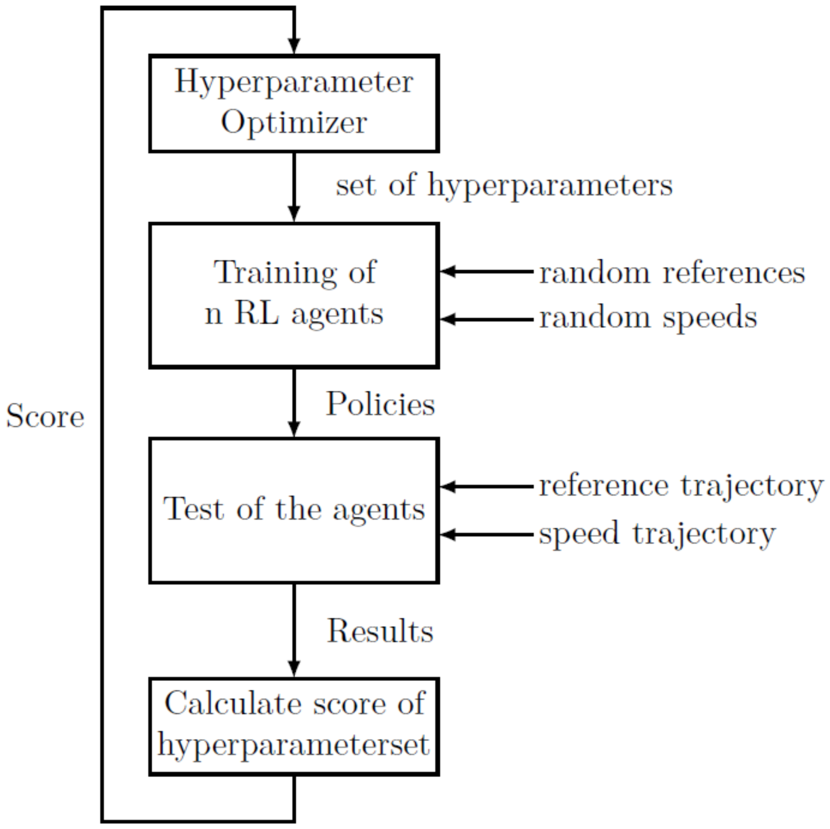


Fig. 6.1: Procedure of the hyperparameter optimization

6.1 Implementation of the hyperparameter optimization

For the hyperparameter optimization, the Python toolbox Optuna [30] is used, which provides various algorithms for the choice of hyperparameters. This toolbox offers the advantage that hyperparameters can be selected hierarchically, so that parameters are selected when another parameter has a certain value. This is important for the optimization of the RL algorithm and the state space, since the algorithms have partially different hyperparameters. The Tree-structured Parzen Estimator (TPE) algorithm is used to determine the hyperparameter sets. Details of this algorithm can be read in [31].

A total of $2,000$ different hyperparameter sets are tested during the optimization. Out of these, the first 250 hyperparameter sets are drawn randomly so that the TPE algorithm has a database for the selection of the hyperparameters. To minimize the influence of the random initialization of ANNs and the random reference values, 10 agents are trained for each hyperparameter set, so that in sum $20,000$ RL controllers are trained. Since the training of a single agent can already take several hours to days, the training of the agents is parallelized. The hyperparameter optimization is therefore performed on the high-performance computer Noctua 2 of the Paderborn Center for Parallel Computing (PC^2). This provides enough computational power and cores to theoretically train all RL agents in parallel. Nevertheless, only up to 100 hyperparameter sets are tested simultaneously, since the hyperparameter optimizer requires the scores of the previous iterations in order to make a decision for the following ones.

Even though the control of a SCIM is a continuous non-terminating task, the training of RL agents is performed in episodes. In addition to the controlling of a reference value, the transient behavior of a controller is often essential for the quality of the control. In particular with the SCIM, a magnetic flux has to be built up first, so that a given torque can be achieved. At the beginning of an episode, all states of the system are set to 0, so this starting behavior has to be experienced in each episode. Instead of a fixed mechanical load, a constant speed is externally applied. The speed is randomly drawn from the maximum speed range in every episode. This enables as many different operating points as possible to be explored during the training. One episode contains $20,000$ time steps, which corresponds to one second for the sampling time $\tau = 50$ µs.

The maximum number of time steps for training an agent is set to $4,000,000$. To speed up the hyperparameter optimization, a validation of the previous training is performed every $500,000$ steps. This is done by running the test episode that determines the result of an agent. If the result of this test fail to improve over the course of the training, the training procedure is terminated prematurely. The test is divided into 10 small episodes, each with $20,000$ time steps, where the speed is increased by 10 % of the maximum speed after each episode. A Wiener process is used to generate the torque reference values, as it is also done in the training. Since the environment, and therefore the reference generator, is seeded before the test, the references are the same for each test sequence. Figure 6.2 shows the reference trajectory of one episode. The result of the test is determined as the average of the rewards. Please note that the undiscounted reward must be used here, because the discounted reward depends on the hyperparameter γ, which is part of the optimization.

In parallel to the training, the derivative of the magnetic flux for the flux estimator could be determined. Furthermore, the system model for the safety layer could be identified. Since both estimations would be very similar for all training processes and these estimations only slow down the training process, the hyperparameter optimization uses pre-estimated values for the parameters. If the system model of the safety layer would be identified during the training, some episodes would terminate at the beginning of the training because the currents could increase above the limitation. By using the already parameterized safety layer, this is completely avoided. The limit value for the currents is chosen to be 90 % of the maximum current, so that all actions are masked by the safety layer which would result in a higher current.

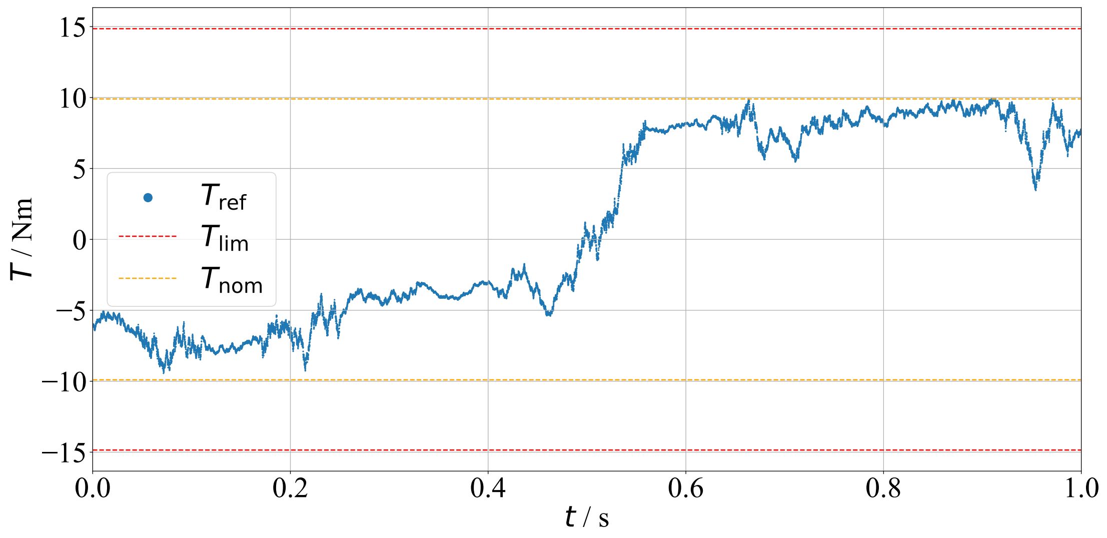


Fig. 6.2: Reference trajectory of a test episode

6.2 Search Space

As mentioned earlier, the hyperparameter optimization is extended to other parameters, which are not hyperparameters, but should be optimized as well. First, the RL algorithm itself is not pre-specified, but the hyperparameter optimizer should detect which of the algorithms is best suited for this problem. The four RL algorithms DQN, Rainbow, PPO and ES are compared, which have already been presented in chapter 3. These algorithms have in common that they use an ANN, but they differ in their policies and their way of updating the network weights.

Furthermore, the state space, that specifies which of the observed parameters of the state are passed to the agents, is investigated. In addition to the observed states, further quantities can be calculated from these, which are also provided to the agent. This is called feature engineering, whereby the agent can obtain expert-based information about the system, which should facilitate the training of the agent. It is important that the agent receives all necessary information about the system state in order to be able to make an optimal decision for the inputs of the system. For this reason, for example, the use of the flux estimator is required. On the other hand, the state space should be reduced as much as possible to avoid giving irrelevant or redundant information to the agent. This would not only slow down the training, but usually also reduce the performance of the agent.

When looking at the differential equations of the system, it can be seen on which variables the state variables are depending. First, all differential equations depend on the rotational speed, which is the reason why this should be part of the state space. The differential equations also contain the currents and voltages. The currents as well as the voltages can be expressed in different coordinate systems. The most straightforward method would be to use the corresponding line voltages and currents. But the quantities in the *abc* coordinate system consist of three components and contain the same information as the quantities in the $\alpha\beta$ coordinate system, which has only two components. To reduce the state space, the currents and voltages in $\alpha\beta$ coordinates, are examined. Because an estimated value for the magnetic flux in $\alpha\beta$ coordinates is available, an estimated angle for the flux can be calculated. Using this angle, it is possible to transform the currents and voltages into the flux-oriented *dq* coordinate system. In contrast to the quantities in stator-fixed coordinate systems, this provides the advantage that the quantities in the *dq* coordinate system are constant for a constant torque. Contrary, the currents and voltages change sinusoidally for the same situation in the stator-fixed coordinate systems.

Furthermore, the agent needs information about the magnetic flux. On the one hand this can be passed in $\alpha\beta$ coordinates. A transformation into the flux-oriented coordinate system is very simple, because the *d*-flux is the absolute magnetic flux and the q-flux is always 0. The information about the angle, however, is lost by this transformation. The angle could easily be passed in a normalized form, but this is not continuous, so unwanted steps in the policy might occur. In addition to the absolute flux, the sine and cosine of the angle is taken, since it is continuous. As in [32], a prefactor w for the sine and cosine is also included to lower the impact of this parameter, which will also be part of the hyperparameter optimization.

The torque is measured only during the training process and can therefore be a part of the reward function, but it is not possible to include it in the state space. Since the magnetic flux is estimated and the currents are measured, a torque estimate is calculated according to equation (2.22). The factor $\frac{L_\mathrm{m}}{L_\mathrm{r}}$ is also estimated when deriving the flux derivative. Otherwise, this could be neglected because it is slightly smaller than 1 and the agent works with normalized data anyway. Finally, the reference value for the torque must be part of the state space, because otherwise the agent does not get the information which torque has to be controlled. In equation (6.1) the four different state spaces are presented, which are tested in the hyperparameter optimization. All quantities, except the rotor flux, are passed in the normalized form.

$$\begin{aligned}
\alpha\beta_1: &\left[\tilde{\omega}_\mathrm{me}\ \tilde{u}_{\mathrm{s}\alpha}\ \tilde{u}_{\mathrm{s}\beta}\ \tilde{i}_{\mathrm{s}\alpha}\ \tilde{i}_{\mathrm{s}\beta}\ \hat{\Psi}_{\mathrm{r}\alpha}\ \hat{\Psi}_{\mathrm{r}\beta}\ \tilde{\hat{T}}\ \tilde{T}_\mathrm{ref}\right] \\
dq_1: &\left[\tilde{\omega}_\mathrm{me}\ \tilde{u}_\mathrm{sd}\ \tilde{u}_\mathrm{sq}\ \tilde{i}_\mathrm{sd}\ \tilde{i}_\mathrm{sq}\ \hat{\Psi}_{\mathrm{r}\alpha}\ \hat{\Psi}_{\mathrm{r}\beta}\ \tilde{\hat{T}}\ \tilde{T}_\mathrm{ref}\right] \\
\alpha\beta_2: &\left[\tilde{\omega}_\mathrm{me}\ \tilde{u}_{\mathrm{s}\alpha}\ \tilde{u}_{\mathrm{s}\beta}\ \tilde{i}_{\mathrm{s}\alpha}\ \tilde{i}_{\mathrm{s}\beta}\ |\hat{\Psi}|\ w\sin(\varepsilon_{\hat{\Psi}})\ w\cos(\varepsilon_{\hat{\Psi}})\ \tilde{\hat{T}}\ \tilde{T}_\mathrm{ref}\right] \\
dq_2: &\left[\tilde{\omega}_\mathrm{me}\ \tilde{u}_\mathrm{sd}\ \tilde{u}_\mathrm{sq}\ \tilde{i}_\mathrm{sd}\ \tilde{i}_\mathrm{sq}\ |\hat{\Psi}|\ w\sin(\varepsilon_{\hat{\Psi}})\ w\cos(\varepsilon_{\hat{\Psi}})\ \tilde{\hat{T}}\ \tilde{T}_\mathrm{ref}\right]
\end{aligned} \tag{6.1}$$

In addition to the agent and the state space, other parameters are considered in hyperparameter optimization. These hyperparameters partly differ from one algorithm to another, but first of all, there are a number of hyperparameters, which are the same for all agents.

The characteristics of the neural networks are one of the most important hyperparameters. This includes on the one hand the number of hidden layers and on the other hand the number of neurons per layer. Combining these gives the number of network weights and thus the number of degrees of freedom of the agent. In addition, the behavior of the ANN is specified by the activation function, which is also a hyperparameter. The three activation functions supported by the Ray toolbox are considered, which include the tangens hyperbolicus (tanh), the Rectified Linear Unit (ReLU) function and the swish function. The three activation functions are shown in figure 6.3.

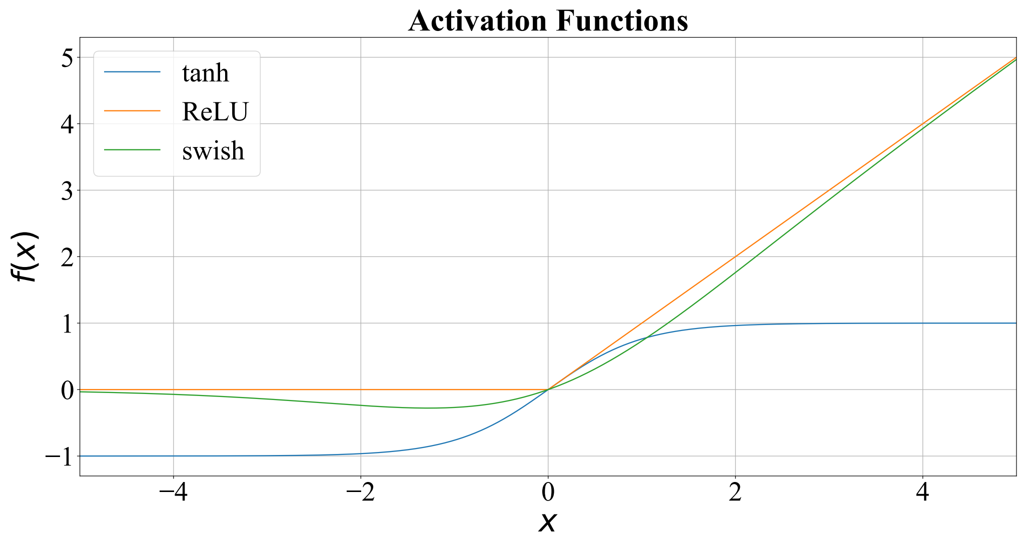


Fig. 6.3: Activation functions

All the RL algorithms use a gradient ascent step or a gradient descent step for the updates of the network weights. Therefore, all algorithms require a learning rate α which scales these updates. Since in the ES algorithm the gradients are calculated in a different way the step size will be in a another range, so this is an additional hyperparameter. Furthermore, the training is performed in batches of state transitions instead of considering only one state transition at a time. The batch size is therefore another shared hyperparameter. In the PPO algorithm, on the other hand, a single batch is divided into minibatches, so in this case the minibatch size is another hyperparameter. This parameter has to be smaller or equal to the batch size. The three gradient-based algorithms, the DQN, the Rainbow, and the PPO, use a discounted return, therefore the discount factor γ is also optimized. The ES algorithm does not need this discounted return, since no q-value is estimated there.

The DQN and Rainbow algorithms use the ϵ greedy policy, where the exploration of the agent can be adjusted by the ϵ parameter. The ϵ here is decreased starting at 1 during training to a final value ϵ_{end}, which is reached after the exploration timesteps. This value ϵ_{end} is a hyperparameter which is being examined. In addition, these two algorithms use a replay buffer, and the size of this replay buffer should be optimized. On the one hand, this replay buffer should be large so that enough different data points can be stored. On the other hand, it should not be excessively large so that it does not increase the memory capacity unnecessarily and slow down the training. Since the Rainbow algorithm is an extension of DQN, it has other unique hyperparameters. These are the number of q-learning steps, the number of atoms, which is the dimension of the support vector $\boldsymbol{z}$, whether to use double q-learning, and whether to use dueling networks.

The coefficient l_2 is another hyperparameter of the ES algorithm. This is multiplied by the momentary network weights when calculating the updates for the network weights and added to the gradient. In the ES algorithm, multiple slightly different policies are evaluated in parallel. To keep the computational effort comparable to that of the other algorithms, the number of parallel evaluated policies is set to 10.

Table 6.1 lists all hyperparameters that are included in the hyperparameter optimization. Also the search spaces, the distributions from which they are drawn and the RL algorithms, where these hyperparameters are used, are specified in the table. The search spaces were derived from previous tests. All other hyperparameters are set to their default values, if they are not mentioned.

Hyperparameter	**DQN**	**RB**	**PPO**	**ES**	**Search Space**
number of layers	✓	✓	✓	✓	uniform int(1, 7)
neurons per layer	✓	✓	✓	✓	uniform int(10, 600)
activation function	✓	✓	✓	✓	categorical([tanh, ReLU, swish])
batch size	✓	✓	✓	✓	categorical([8, 16, 32, 64, 128, 256, 512])
minibatch size			✓		categorical([8, 16, 32, 64, 128])
learning rate α	✓	✓	✓		uniform($5 \cdot 10^{-6}$, $3 \cdot 10^{-4}$)
step size				✓	uniform(0.001, 0.1)
discount factor γ	✓	✓	✓		loguniform(0.8, 0.999)
replay buffer size	✓	✓			uniform int(10000, 1500000)
final epsilon ϵ_{end}	✓	✓			uniform(0, 0.1)
exploration timesteps	✓	✓			categorical([500, 1000, 2000, 4000])
l_2 coefficient				✓	uniform(10^{-4}, 10^{-3})
q-learning steps		✓			uniform int(1, 10)
number of atoms		✓			uniform int(2, 20)
double q-learning		✓			categorical([True, False])
dueling networks		✓			categorical([True, False])

Tab. 6.1: Hyperparameters

6.3 Results of the hyperparameter optimization

In the following, the results of hyperparameter optimization are evaluated in general. Subsequently, the examined parameters and hyperparameters are discussed. Since a total number of 10 agents have been trained and tested for each hyperparameter set, the median of these 10 attempts is used to determine the performance of a hyperparameter set. Figure 6.4 shows the timeline of the rewards per step of the trained agents in the test.

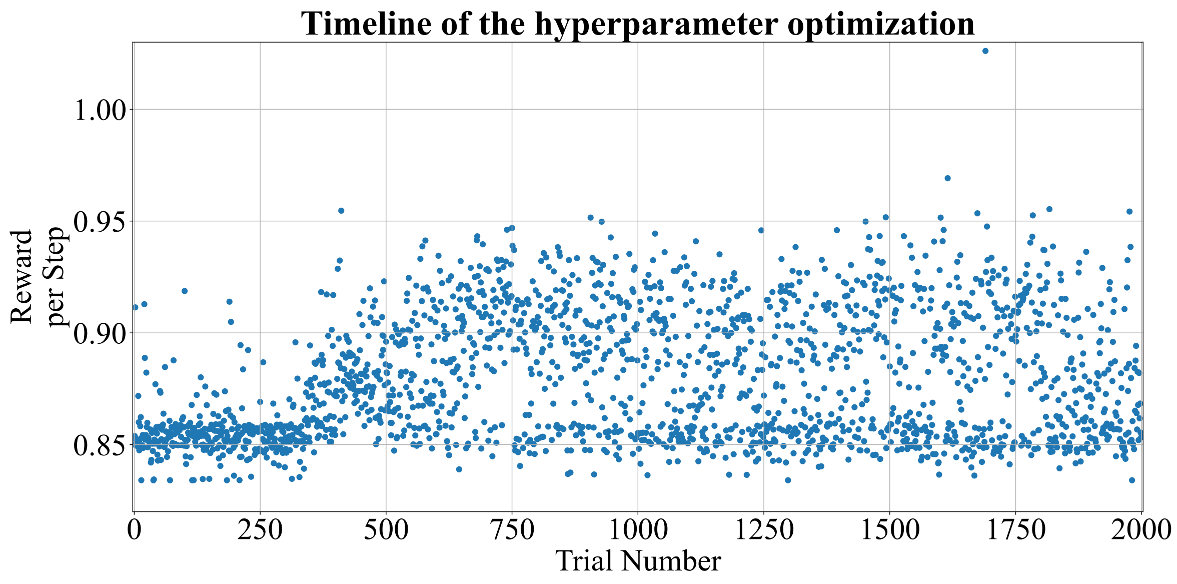


Fig. 6.4: Timeline of the hyperparameter optimization

It can be seen that the rewards in the first phase are at a low level until around the trial 350. There are some outliers on the way up, but most agents have achieved a reward per step of around 0.85. This can be explained by the fact that the initial 250 hyperparameter sets are randomly sampled and only after this the hyperparameter optimizer starts to select the parameters. Between the trials 350 and 750, the maximum achieved reward increases significantly. It can be seen that the TPE algorithm has learned which hyperparameters have to be selected, in order to increase the reward. After the trial 750, the curve flattens quickly and only slight improvements can be seen. There are some slight and a few strong outliers upwards.

Particularly remarkable is trial 1690, which exceeds the other attempts by a reward per step of 0.05 more than the second best trial. The reasons for this high performance will be discussed in the following sections. Furthermore, the rewards of many agents are in the range around 0.85, even at the end of the training. This behavior is also analyzed in more detail in the following.

The agents are tested every 500,000 steps during the training and the reward is compared with the previous reward. If the new reward is smaller than the old one, the training is completed. Therefore, it can be assumed that the reward increases with the training length. Figure 6.5 shows the reward per step over the averaged training length of a trial.

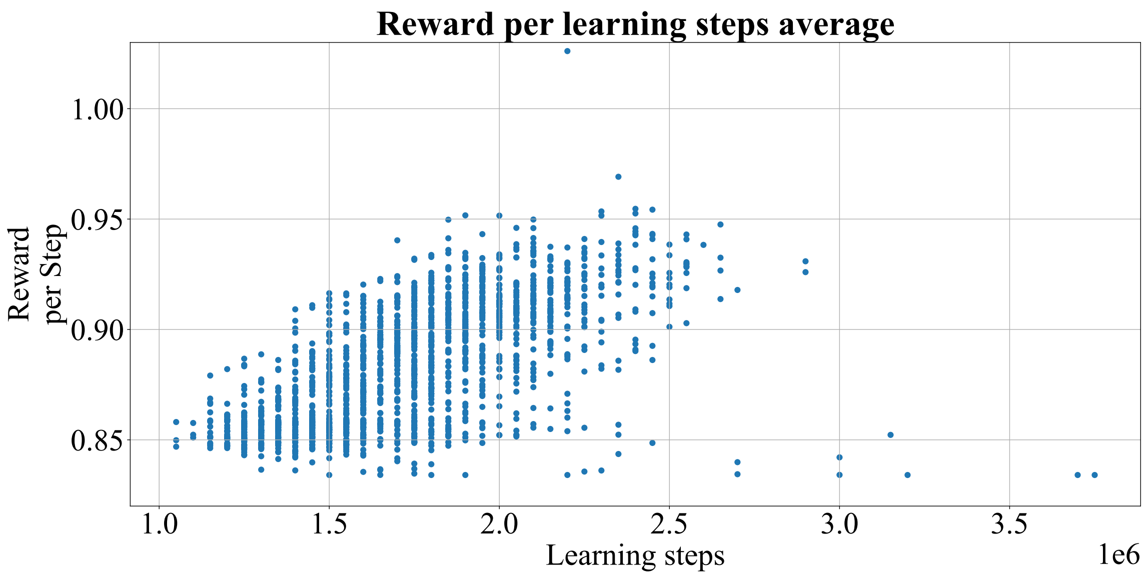


Fig. 6.5: Reward per step over the averaged learning steps

The expected behavior can be recognized in the graph. Up to an average value of about 2.5 million training steps, the reward per step increases almost linearly. However, there are also tests that show a very low reward despite an average number of training steps of more than 3 million. These agents thus improved constantly over a long period of time, but the improvements were not significant. Since no improvements can be seen with an average training length of more than 2.5 million steps, a maximum training length of 3 million steps would have been sufficient. Nevertheless, most of the agents did not reach this number of steps at all, because the training was stopped before. Of course, it would be possible to train all agents for the same number of steps, but no significant improvements in performance would be expected if the reward had decreased, even if only slightly, because the learning curve of an agent can be expected to have already converged in that case.

Figure 6.6 shows the learning curve of an agent, which was recorded during the training. The training rewards are averaged over a period of 10,000 time steps. This agent is a PPO agent, which is why it uses a discounted return during training. The influence of the discount factor γ was removed for this illustration in order to establish comparability. It can be seen that the reward is initially low and increases gradually. From around 2.5 million steps until the end of the training at 4 million steps, no significant improvements can be seen. It is interesting to note, that from 3 million steps onwards, there are no more outliers downwards, which means that the control seems to be quite stable. Although this agent was trained a total of 4 million steps, the learning curve already converges after 3 million steps, thus a training of 3 million steps was sufficient for this agent too.

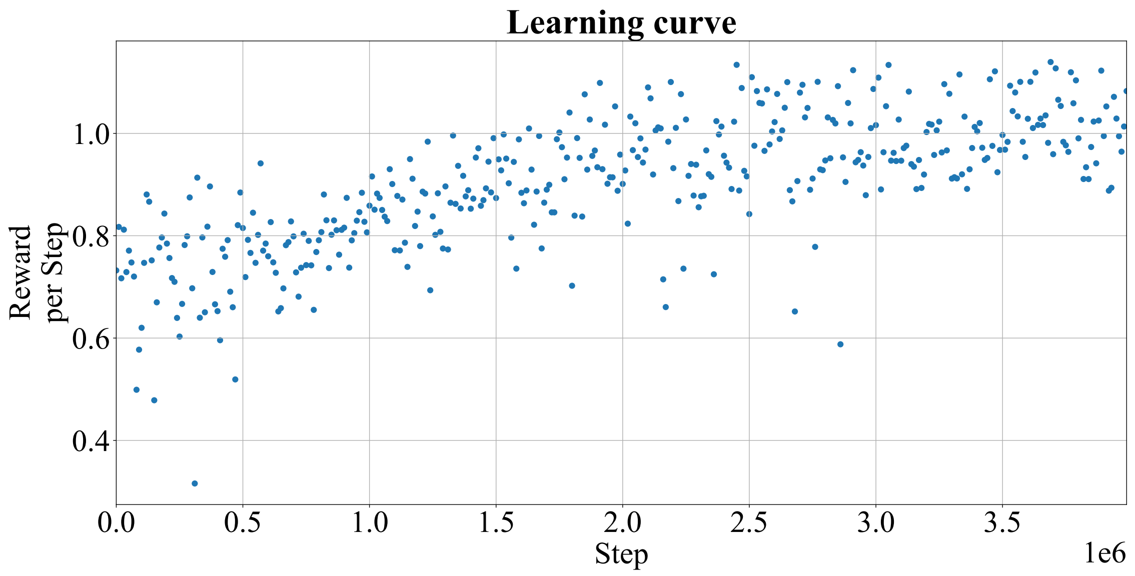


Fig. 6.6: Learning curve of an agent during the training

6.3.1 Analysis of the different RL algorithms

The most important parameter considered in hyperparameter optimization is the RL algorithm itself. In Figure 6.7, a boxplot of the results of the four different RL algorithms is presented. It is immediately noticeable that the best results by far were achieved by the PPO algorithm. The median of the rewards as well as the maximum and the two middle quantiles exceed the maximum of the other algorithms. Furthermore, it is highly notable that the rewards of the PPO algorithm varies significantly more than those of the other algorithms. The one upward outlier is also visible in this boxplot.

In contrast, the results of the DQN, Rainbow, and ES algorithms differ only slightly. The median of the Rainbow is slightly higher than that of the DQN. This was to be expected, since the Rainbow algorithm is an extension of the DQN. In addition, the variations of the Rainbow algorithm are very small. The median of the Rainbow algorithm is higher than those of the DQN and ES, however, the maxima of these two algorithms are in turn higher. Even if the minimum of the PPO algorithm is in the range of the minima of the other algorithms and is actually quite below the minimum of the Rainbow algorithm, only the PPO algorithm will be considered in the further analysis. These results explain why so many agents have a reward in the lower range. This is because only PPO agents were able to achieve a higher reward.

Since the PPO algorithm achieved the best results by far, the question now is why the performance of these PPO agents is so much higher than the others. To address this question, the differences between the algorithms are analyzed. The DQN and the Rainbow algorithm have the same main principles. They are off policy algorithms which approximate the q-function of the MDP. In addition to the q-function, the PPO algorithm estimates

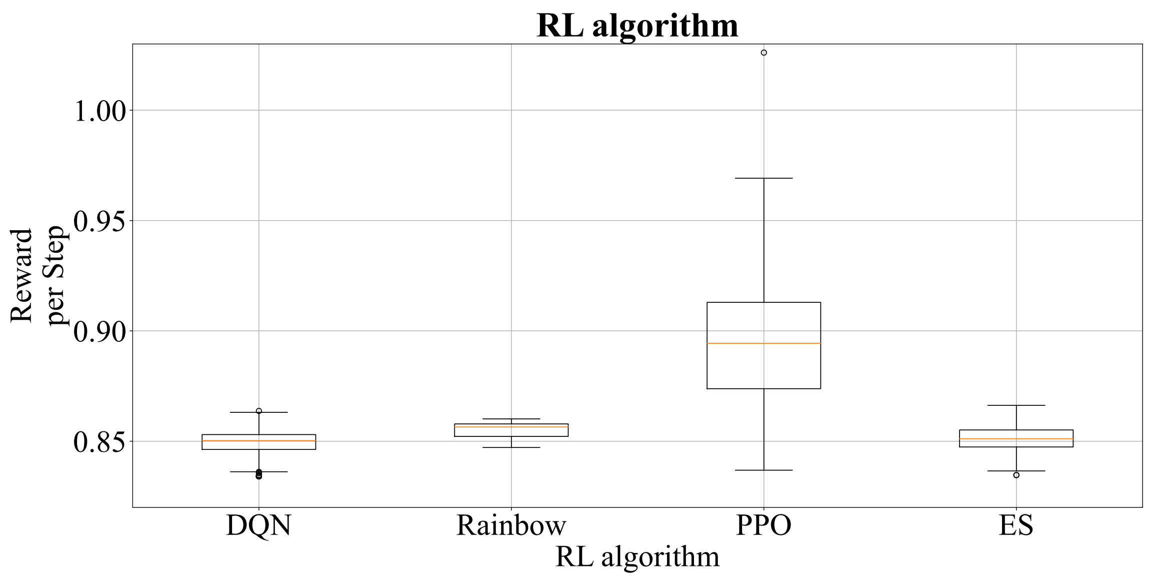


Fig. 6.7: Boxplot of the results of the different RL algorithms

a probability distribution from which the actions are randomly drawn during training. The ES algorithm determines this probability distribution of actions without an estimate of the q-function. For the selection of an action with the ϵ greedy policy, the learned q-function is maximized according to the action. In the other two algorithms, the ANN outputs a probability distribution for the actions that indicates which action will lead to the best result. Thus, not only the system behavior is predicted, but also a guideline for the actions is given. This is potentially an advantage in decision making.

Furthermore, the estimated q-values will not differ as much as the probabilities in a stochastic policy. Since this is a technical process and the influence of a single switching vector is limited to a short period of time, the expected rewards will not differ as much as the probabilities of the stochastic policy. If the system behavior in the learned q-function is now minimally incorrectly estimated, a switching vector that is different from the optimal one will be selected. To verify this, figure 6.8 shows a graph of the estimated q-values and the probabilities for each action, when using an exemplary trained agent for each algorithm. The system is in the same state in each case and the references are also identical. In the appendix A.2 the same plots for different operation points can be found.

As expected, the probabilities of the PPO and ES agents are much higher for one action than for the others. The q-values of the DQN and Rainbow agents are significantly lower and closer to each other. It is striking that all agents would choose a different action in the deterministic choice of the action. It is expected that the q-values of the DQN and the Rainbow agent should approximately match, since the discount factors γ are very similar. For some actions, the q-values are at least similar in their tendency, but for action 2 and 6,

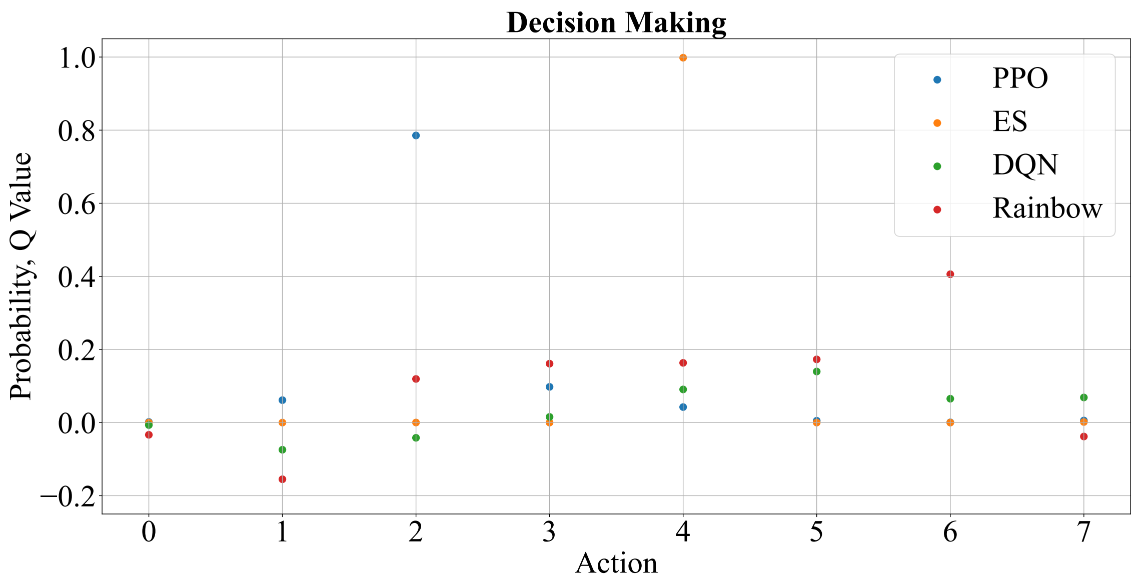


Fig. 6.8: Estimated q-values and probabilities for the actions

the values diverge widely. This effect is also more evident for some of the other operation points, which indicates that at least one of the two agents was not trained well.

For the probabilities of the ES algorithm, it can be seen that in many operating points these go near 1 for one action and near 0 for the remaining actions. This indicates that an overfitting of the network weights has taken place. If the probability values are such high during the training of the ES agent, only few other actions are drawn, so that no further exploration takes place. Only the probability values of the PPO agent show the expected behavior. The probability for one action is increased. The values for the neighboring actions are slightly increased because these voltage vectors are also neighboring in the voltage hexagon from figure 2.4. For this reason the difference of the voltages is smaller compared to the opposite vectors. Therefore the reaction of the system will also be more similar. Another aspect in training of the ES agent is that the updates of the network weights are done on the basis of the results of random changes in the network weights, while in the other algorithms the derivatives of the estimated functions are used. The advantage of using the explicit derivatives in the gradient method is that the change in the output of the ANN is guaranteed to be in the direction given by the discounted return, while this is not the case for the ES algorithm.

6.3.2 Analysis of the different state spaces

In addition to the RL algorithm, the state space was optimized as another parameter. Compared are state spaces containing currents and voltages in dq coordinates on the one hand and in $\alpha\beta$ coordinates on the other hand. Furthermore, there are two different methods to pass the information of the magnetic flux, once in the $\alpha\beta$ coordinates or the magnitude of the flux and the weighted sine and cosine of the angle, so this results in four combinations. The definition of the different state spaces are shown in equation (6.1). The state space results of the hyperparameter optimization are illustrated in a boxplot in figure 6.9.

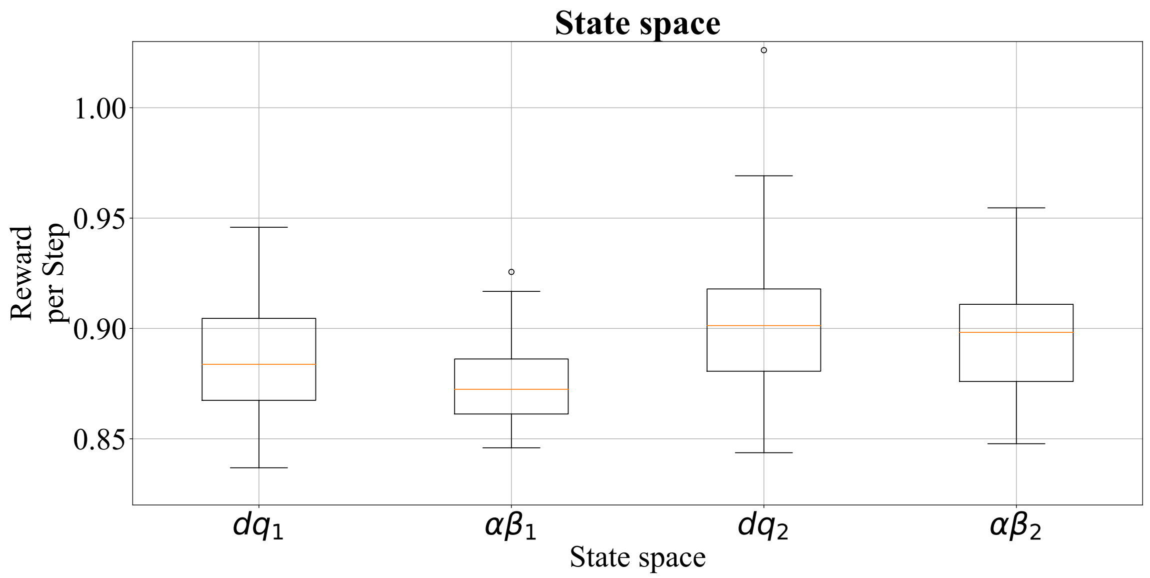


Fig. 6.9: Results of the state spaces

For this parameter, the differences are not as significant as they are for the RL algorithm. Nevertheless, it can be seen that the rewards of the state spaces dq_2 and $\alpha\beta_2$ tend to be higher. If only the two coordinate systems of the currents and voltages are compared, the rewards of the state spaces with the quantities in dq coordinates are slightly higher. It can thus be seen that both variable elements have an influence on the control of the motor.

The greatest influence of the state space on the control performance has apparently the representation of the magnetic flux. For the magnetic flux, two quantities are relevant, the magnitude and the angle of the flux. The magnitude of the magnetic flux needs to be large enough to allow a desired operating point to be reached. The eight voltage vectors are defined in a stator-fixed coordinate system, so that the angle of the flux is needed to determine the optimal voltage vector. Since the magnetic flux rotates, a voltage vector for two opposite angles of the flux at an identical state will have opposing effects on the subsequent state of the SCIM. If a continuous action space would be used, the input voltages could be calculated in flux-oriented coordinates and then transformed into stator-fixed coordinates. Information about the angle of the flux would not be required in the agent. Because in this case a discrete action space with voltage vectors in stator-fixed coordinates is used, the agent has to consider this coordinate transformation implicitly in the policy. This problem is much more complex if just the two flux components are known compared to when the agent is given the sine and cosine of the flux angle.

Moreover, the rewards of the state spaces with the currents and voltages in dq coordinates are quite better. The advantage of using this coordinate system in contrast to the $\alpha\beta$ coordinate system is that the quantities are ideally constant at a constant torque operation point as well. This is not the case in the stator-fixed coordinates, instead the currents and voltages have a sinusoidal shape with the rotational frequency of the magnetic field. The handling of constant input quantities is simpler, which explains the higher rewards when utilizing the dq state spaces. The weight factor w of the sine and cosine has no significant influence on the results of the agents. The corresponding plot can be found in the appendix A.3.

6.3.3 Analysis of the network parameters

In the hyperparameter optimization a total of 16 different hyperparameters were investigated, however, the performance of the PPO agents was shown to be much higher than the others. Therefore, only the hyperparameters that are relevant for the PPO algorithm are analyzed in the following. The plots of the results of the hyperparameters, which are only used for the other algorithms, are presented in the appendix A.3.

All RL algorithms examined here use ANNs for function approximation. The behavior of these ANNs is defined by the number of hidden layers, the number of neurons per hidden layer and the activation function of the neurons. These three parameters were tuned in the hyperparameter optimization. On the one hand, the ANN must be large enough so that the system behavior and a policy can be learned adequately. On the other hand, the network should be as small as possible to reduce the computation time, because these neural networks have to be evaluated within the sampling time in the later operation. In addition, a large ANN holds the risk of memorizing the reference trajectories due to the high number of degrees of freedom instead of learning the system behavior. The consequence would be that although the controller shows good results in the training, the control performance on unseen data is much poorer.

Figure 6.10 illustrates the rewards depending on the number of hidden layers as a boxplot. It can be seen that the rewards show a very similar behavior starting from a number of 3 hidden layers. The maximum is slightly higher at a number of 7 layers and significantly increased at a number of 6 layers. The rewards drop sharply with less than 3 hidden layers.

There is a similar trend in the results of the number of neurons per hidden layer, which can be seen in figure 6.11. Up to a number of about 100 neurons per layer, the reward increases linearly. After that, the variation of the rewards is static. The maximum of the reward is reached at the upper boundary of 600 neurons per layer. This is unfavorable from an optimization point of view, since an increase in the number of neurons could possibly result in a higher reward. In the previous tests, the optimal number of neurons per layer was considerably smaller, which is why the limit was set to 600. On the other hand, this number of neurons already results in a huge ANN with many degrees of freedom, depending on the number of layers. Finally, the number of degrees of freedom, which corresponds to the number of network weights, is the most important factor in the structure of the neural network.

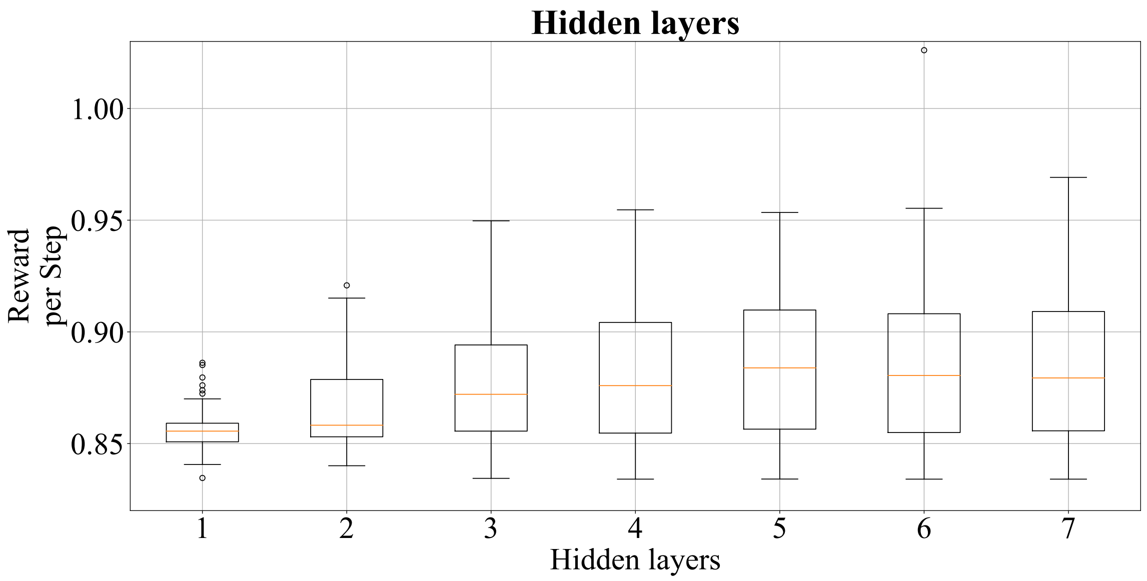


Fig. 6.10: Results of the the hidden layers

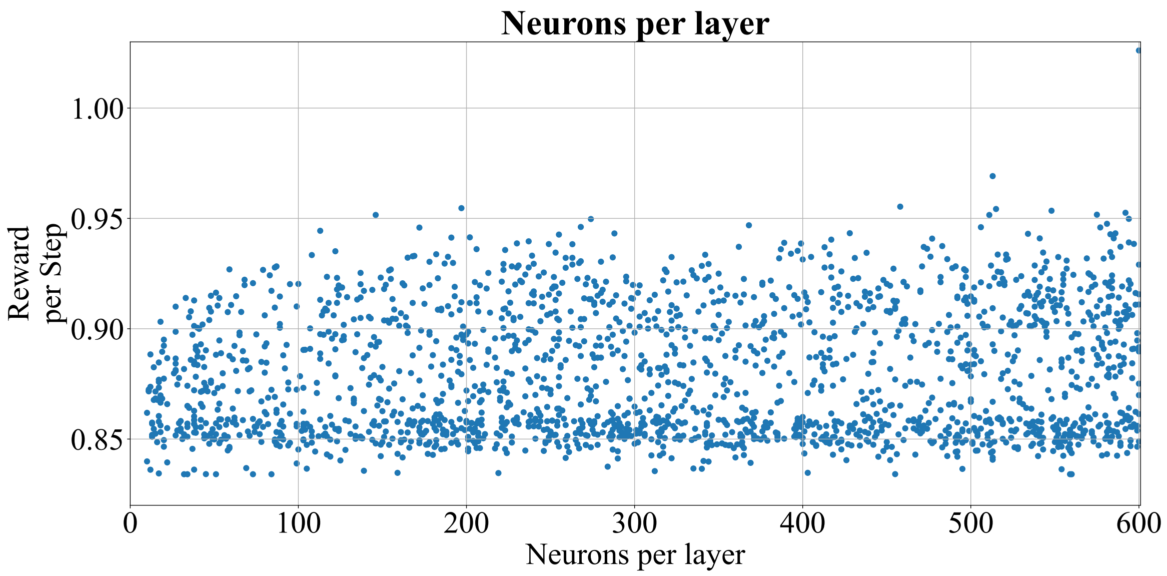


Fig. 6.11: Results of the number of neurons per hidden layer

The number of network weights per layer is divided into the number of multiplicative network weights and the bias. With the knowledge of the network architecture, the number of weights can be calculated. In this way, the achieved rewards can be plotted depending on the degrees of freedom of the ANNs. This can be seen in figure 6.12 in a logarithmic scale. The maximum reward increases linearly in a logarithmic scale up to a number of approximately 100,000 net weights. At a higher number of net weights no significant changes can be observed. Even if the maximum of the reward is reached at a number of more than 2 million network weights, with the exception of this data point, a number of 100,000 network weights seems to be sufficient.

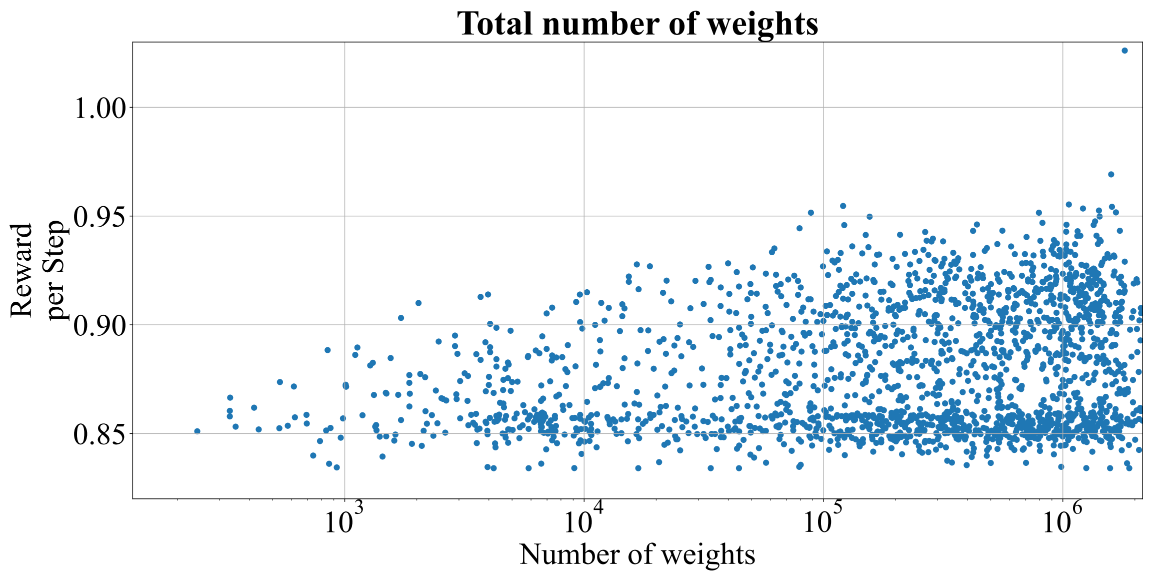


Fig. 6.12: Reward as a function of the number of weights

The last hyperparameter that characterizes the behavior of the ANN is the activation function of the neurons. Three different activation functions have been tested in the hyperparameter optimization, the tanh, the ReLU and the swish function. The results of the activation functions can be seen in the boxplot in figure 6.13. The highest rewards have been achieved with the tanh activation function, however, the rewards of the ReLU function are much higher on average. The swish activation function is inferior to the other two, although this function is very similar to the ReLU function in large parts. However, the behavior around the origin varies strongly, which could explain the different results. Nevertheless, it is interesting that two activation functions that are very different, such as the tanh and ReLU function, can both lead to high rewards.

6.3.4 Analysis of the training parameters

In addition to the hyperparameters that define the structure of the ANN, other parameters that influence the training process are investigated. In the case of the PPO algorithm, these hyperparameters are the learning rate α, the batch size and minibatch size as well as the discount factor γ.

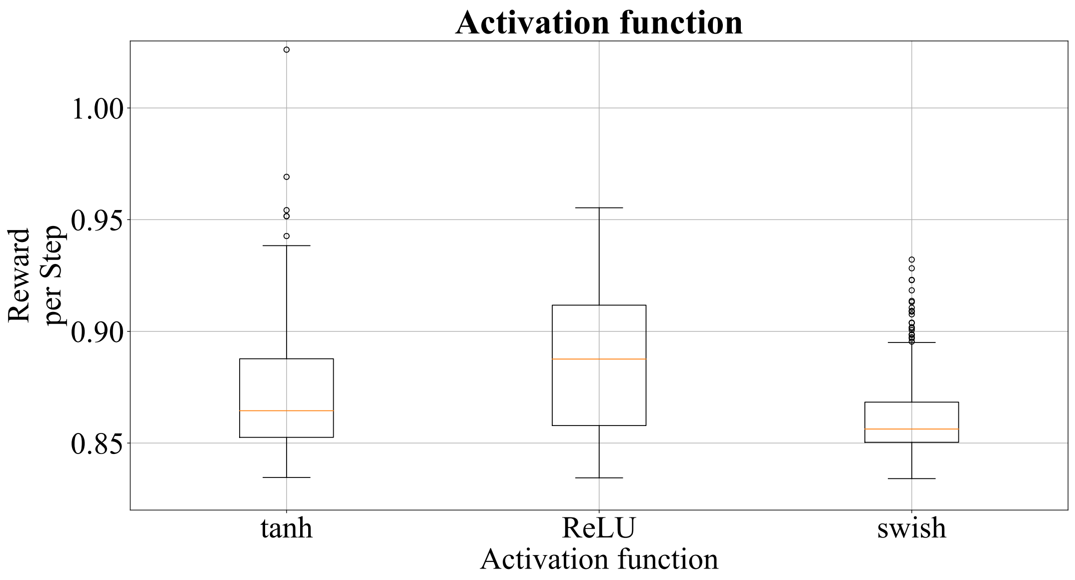


Fig. 6.13: Results of the activation function

The learning rate is used in the updates of the network weights by a gradient method. The higher the learning rate, the larger the changes in the network parameters. In order to speed up the training, the learning rate should be high, but this results in the problem that the network weights change strongly in each iteration. As a result, the network weights may not converge, but alternate over a range permanently. In the PPO algorithm, a gradient that is too high is clipped. But this should prevent individual network weights from changing too fast and not to do so permanently. In the other algorithms there is no such clipping, which is why a too large learning rate has more negative consequences.

Figure 6.14 plots the rewards of the hyperparameter optimization in relation to the learning rate. It can be seen that, as expected, small learning rates lead to a higher reward. If the learning rate is lower than 10^{-4}, the differences are hardly noticeable. The search region for the learning rate was narrowed sharply based on previous test sessions, as this indicated a distinct region where the rewards would be high.

In RL, there are two different approaches of how much data is included in an update of the network weights. On the one hand, online learning is used, which updates the network weights after each time step based on the last iteration. On the other hand, there is the possibility to wait for a batch of samples and determine the loss and the gradient for all of these transitions together. This offline learning is faster than the online learning, but there is a larger delay before the policy is updated. The algorithms applied here use the batch learning, which is why the batch size is studied here as a hyperparameter.

The results of the batch size study can be seen in figure A.18 in the appendix. It is evident that the rewards increase with increasing batch size, in both the average and the maximum. It can also be seen that the rewards for batch sizes below 128 are significantly lower. This

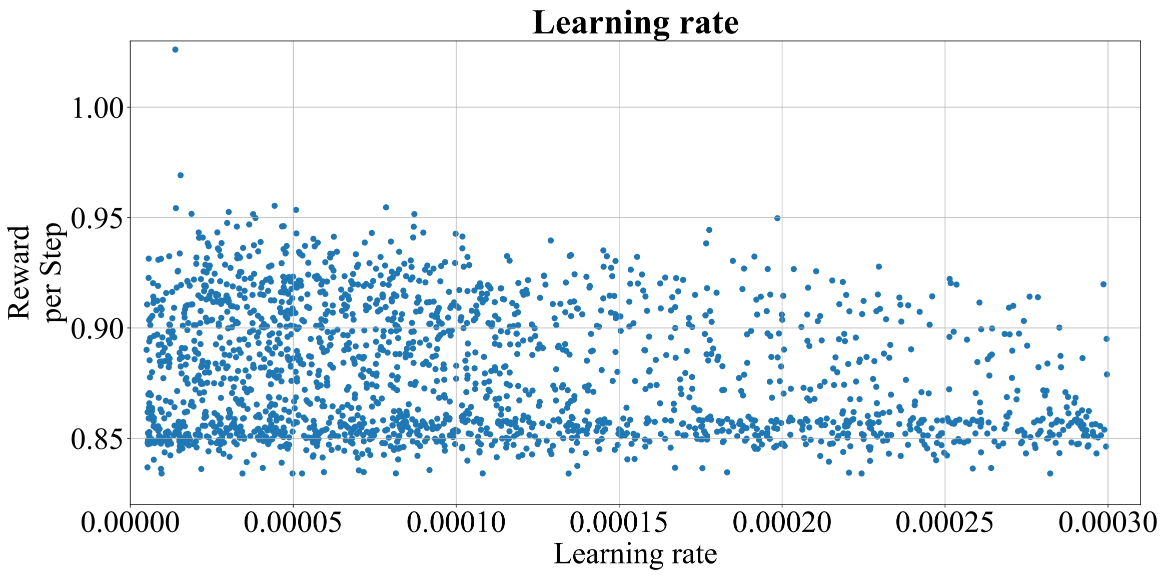


Fig. 6.14: Results of the learning rate

is because the minimum batch size for the PPO algorithm is 128, as it further divides the batches into minibatches. In this way, a variant between online and offline learning should be used. For this reason, powers of two are chosen for the possible values of the batch size and minibatch size, since this ensures that the minibatch size is a divisor of the batch size. The results for the minibatch size are presented in the boxplot in figure A.19 in the appendix, which show a similar trend as the results of the batch size. The maximum and the median value of the calculated rewards increase with an increasing minibatch size. Especially the maximum of the rewards is higher for a minibatch size of 128 than for the other values.

The last parameter that has been explored in the hyperparameter optimization and is involved in the training of a PPO agent is the discount factor γ, which indicates whether the discounted return is shortsighted or farsighted. The results of this discount factor are shown in figure A.20 in the appendix. There is a slight tendency that small values for γ lead to a higher reward. In addition, the rewards for γ above 0.95 drop noticeably. It can thus be concluded that the rewards reached in the tests are higher when the agent was trained with a more shortsighted discounted return.

The analysis of the hyperparameters shows that mainly the used RL algorithm has a major influence on the performance of the RL controller. There is also a clear tendency for many other parameters and hyperparameters indicating to know which values have a positive influence on the training of an agent. However, based on analysis of the hyperparameters studied, no explanation could be found to explain why one of the hyperparameter sets has such a significantly better outcome than the others. Although the hyperparameters of this hyperparameter set are confirmed by the general tendency that they optimize the training of the agent, the difference cannot be justified, which is why further analysis of the results is necessary. The parameters and hyperparameters of the best agent are listed in table 6.2.

Parameter	**Value**
agent	PPO
state space	dq_2
weighting factor w	0.883
number of layers	6
neurons per layer	600
activation function	tanh
batch size	512
minibatch size	128
learning rate α	$1.38 \cdot 10^{-5}$
discount factor γ	0.854

Tab. 6.2: Best hyperparameters

6.4 Statistical analysis of the results

Because one hyperparameter set performed significantly superior to the others in the hyperparameter optimization, and this cannot be attributed to the chosen values, the cause of this outlier is to be investigated. In the hyperparameter optimization, a total of 10 different agents were trained with one hyperparameter set. Looking at the results of these 10 agents, for the agent with the highest median of the individual test results, it is noticeable that while many individual results have high rewards, 4 test rewards in particular are much lower. The highest individual reward of this trial is 1.1 while the lowest is just 0.85. Calculating the standard deviation of these results, we get a value of 0.09, which is almost 10 % of the median. The spread of these values is even higher than that of all the medians.

The variations in the results are caused by the random initializations of the ANN and the random processes used for the generation of the reference values. The effect of these random variables, however, seems to be much larger than expected. To get deeper into these variances, additional agents are trained with the hyperparameter set of the best agent. Figure 6.15 shows a histogram of the rewards of a total of 200 trained agents.

As expected, the rewards are widely distributed, but most of the rewards are between 0.88 and 0.95. The median of these 200 rewards is 0.929, which is about 0.1 lower than in the hyperparameter optimization. The standard deviation is slightly lower with 0.07, but still relatively high. However, there are some upward outliers, but no downward outliers, which is probably caused by the definition of the reward function, as it is discontinuous when

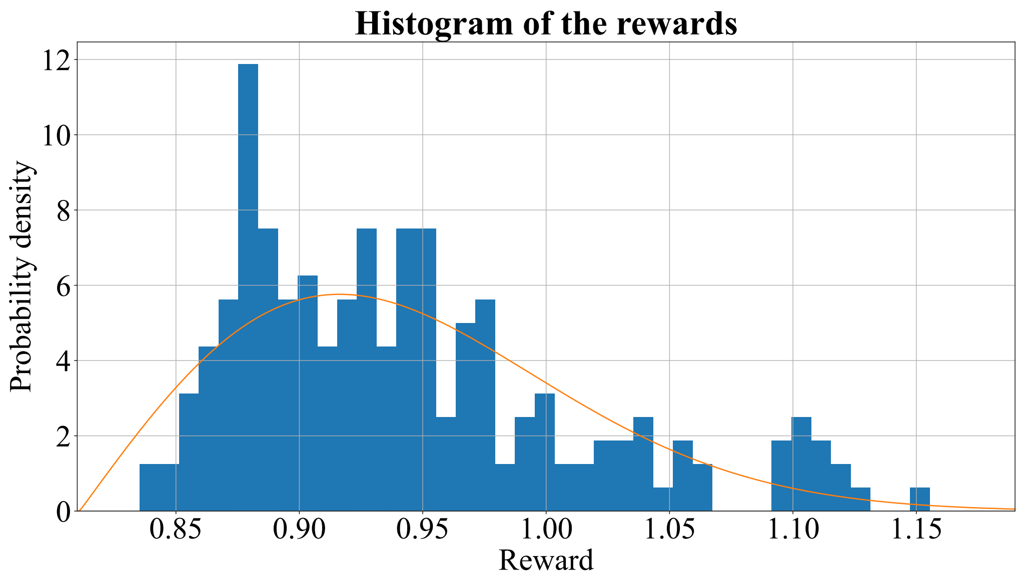


Fig. 6.15: Histogram of the 200 rewards

the control error is within a band of 10 % around the reference. The reward function will be discussed in chapter 7 in more detail.

The distribution of rewards in the histogram has approximately the shape of a Rayleigh distribution shifted to the right, which is a specialized type of the Weibull distribution. The probability density is given by

$$f(x|\sigma, x_{\text{off}}) = \begin{cases} \frac{x - x_{\text{off}}}{\sigma^2} e^{-\frac{(x-x_{\text{off}})^2}{2\sigma^2}} & x \geq 0 \\ 0 & x < x_{\text{off}} \end{cases} \tag{6.2}$$

as a function of the parameter σ, where x_{off} is the offset of the shift. Since the parameters σ and x_{off} are unknown and the expectation value $E(X) = \sigma\sqrt{\frac{\pi}{2}}$ depends on both, σ cannot be determined by the mean value of the test results. However, since the standard deviation and thus the variance of the samples can be calculated, the parameter σ can be determined by using the equation of the variance $\text{Var}(X) = \frac{4-\pi}{2}\sigma^2$ to $\sigma = 0.1053$. With this result, the offset is then calculated to $x_{\text{off}} = 0.811$ using the expectation value and the average of the rewards. The resulting distribution density function can be seen as well in figure 6.15.

The excellent results of the one agent in the hyperparameter optimization are explained by the random distribution of the rewards with a high standard deviation. Thus, it has to be clarified whether all the results of the hyperparameter optimization have such a high standard deviation and whether the number of 10 agents per hyperparameter set was chosen too low. Therefore, in figure 6.16, the standard deviations of the 25 best hyperparameter sets are shown in blue, sorted by the rewards in descending order. It can be seen that the standard deviation of the best agent is the highest. Thus, it is expected

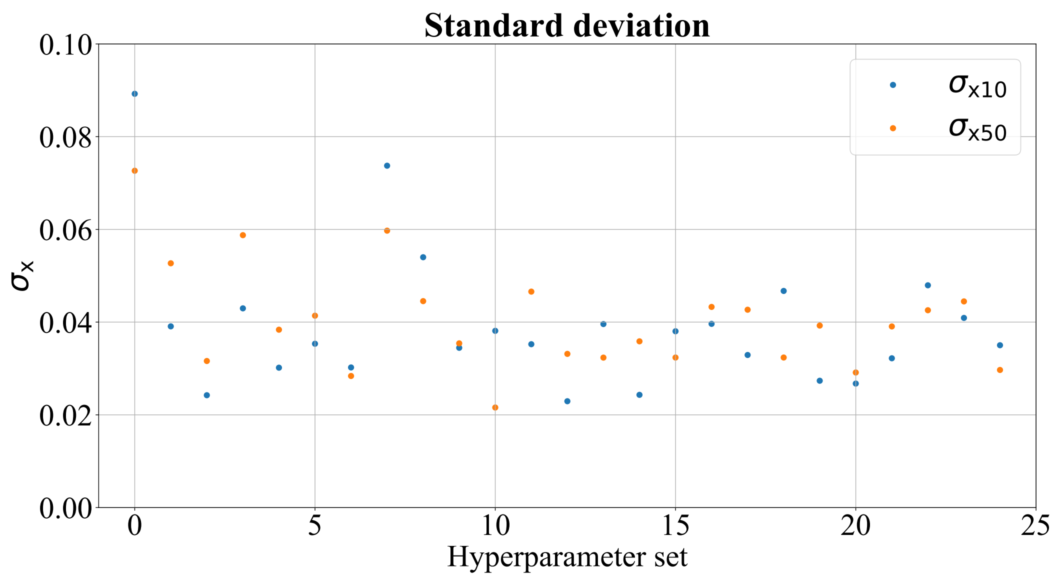


Fig. 6.16: Standard deviations of the rewards

that the spread of the rewards of the other agents is smaller. To verify this, 50 additional agents have been trained with each of the 25 best hyperparameter sets. The standard deviations of the rewards of these agents are also shown in orange in figure 6.16. These confirm the results from the hyperparameter optimization.

Figure 6.17 plots the median rewards of the 50 additionally trained agents compared to the median rewards from the hyperparameter optimization. It is first noticeable that all rewards of the additionally trained agents are below those of the hyperparameter optimization. However, the deviations are significantly smaller for all hyperparameter sets than for the best hyperparameter set. All in all, it can be assumed that the sampling number of 10 agents in the hyperparameter optimization is too small, because the median of the rewards is highly influenced by stochastic variations. Nevertheless, the obtained results of the hyperparameter optimization provide guidance on how to choose these parameters. Furthermore, it has been shown that some agents using the hyperparameter sets were able to increase the reward significantly compared to the beginning of the hyperparameter optimization.

It is obvious that the random processes at the initialization and the training have a strong influence on the rewards of the agents. Therefore, it has to be clarified how many agents should be trained with one hyperparameter set in order to estimate the median accurately. Closely related to the estimation of the median is the estimation of the average. For estimating the average, metrics exist to calculate the standard deviation as a function of the number of samples. The correlation of the standard deviation of the estimated average and the calculated standard deviation is given by

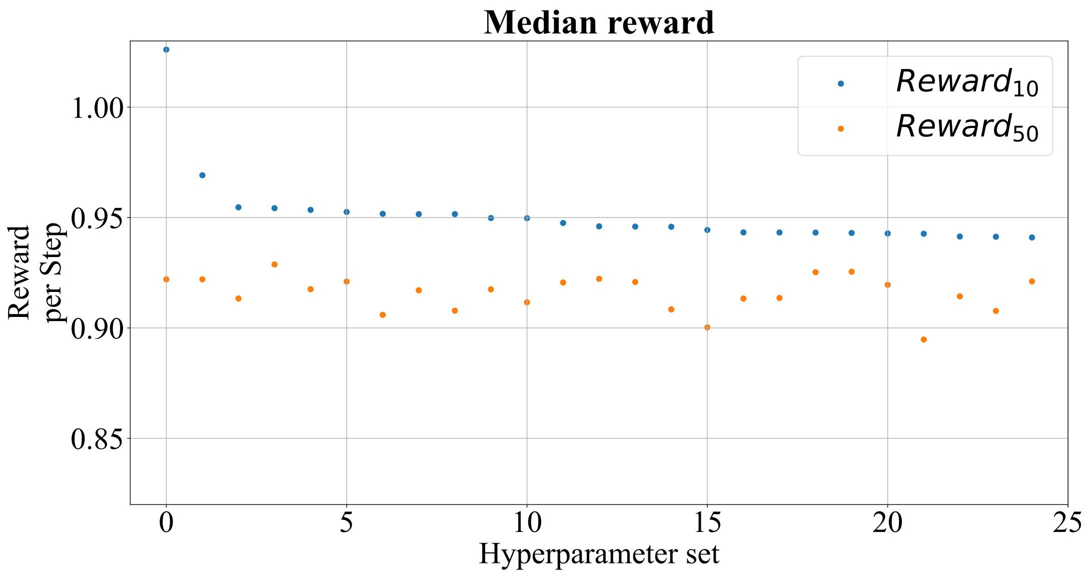


Fig. 6.17: Comparison of the median of the rewards

$$\sigma_{\overline{x}} = \frac{\sigma_{\mathrm{x}}}{\sqrt{n}} \tag{6.3}$$

where σ_{x} is the measured standard deviation and n is the number of samples. To minimize the estimation error of all rewards, the maximum standard deviation of the average should be small. The largest measured standard deviation results from the hyperparameter set with the highest reward, so this is considered in the following. Figure 6.18 plots the standard deviation of the estimated average as a function of the number of samples.

It can be seen that although the standard deviation of the estimated value for 10 samples is only 0.022, the actual deviation can be significantly larger. The curve flattened out further so that increasing the number of samples would only slightly reduce the standard deviation. If the number of samples is doubled up to 20, the standard deviation would still be reduced by about one third. Defining a maximum value of 0.01 for the standard deviation of the estimated value, at least 48 agents have to be trained with one hyperparameter set. However, since the computational resources are limited, a compromise must be found. For example, this could be made at 30 samples, since the curve hardly decreases after this point.

On the one hand, the number of samples can be increased to improve the accuracy of the estimates, but it would be more advantageous if the variance of the agents' rewards would be reduced. For this purpose, it first has to be clarified what causes the variance of the results. This could be reduced by ensuring that the RL algorithm converges to a global optimum rather than a local optimum, so that the training becomes more stable.

In the training process, the reference values for the torque and the rotational speed of the motor are randomly chosen. Furthermore, the ANNs of the RL agents are initialized with

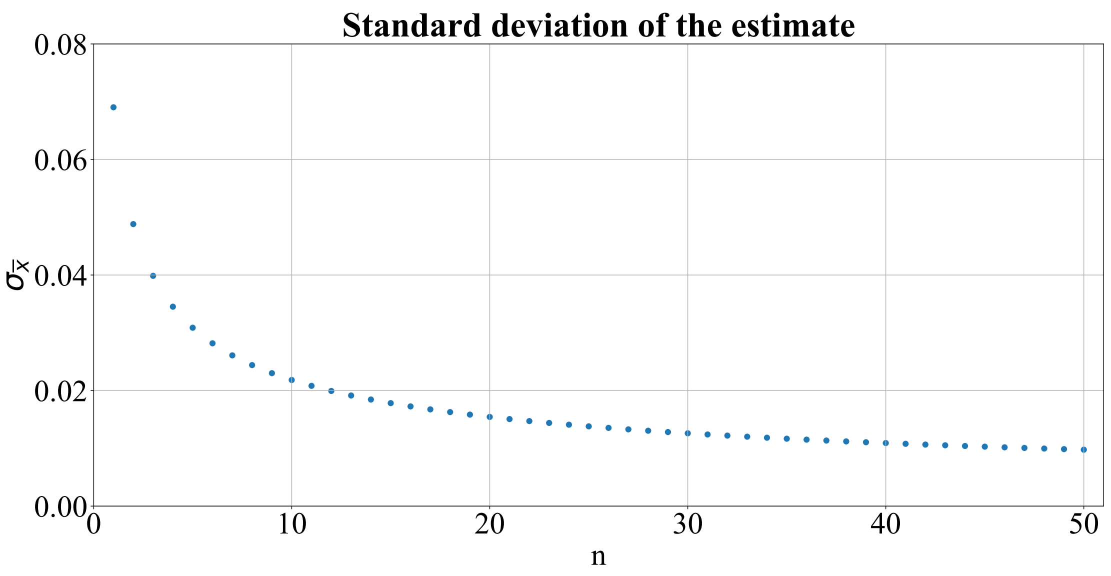


Fig. 6.18: Standard deviation of the estimated average

random network weights before the training. Both of these factors increase the variance of the results, but it is important to verify how strong these influences are in order to select suitable techniques to reduce the variance. For this reason, two sets of 100 agents have been tested. In one of the test series, all agents are trained with the same reference values and at the same velocities. In contrast, in the other test series, the network weights are initialized identically.

The histograms of the two tests, which can be found in the appendix A.4, do not differ qualitatively from the histogram shown in figure 6.15. The variance of the rewards of the test series with equal reference values and varying initial network weights is 0.071, which is comparable to the variance of the previous test. The variance of the results with identical initial network weights is also only marginally lower with 0.067. These results suggest that both random variables have a considerable influence on the variance of the rewards. Furthermore, the median of the rewards can be evaluated. This median is in the case of equal reference values during the training at 0.94 and therefore higher than that of the test with identical initial network weights, which has a median of 0.918. This difference is larger than the standard deviation of the estimated average for 100 samples, which leads to the hypothesis that this deviation is systematic. This difference could be explained, for example, by the assumption that the initial weights were chosen in such a way that the policy started near a local optimum and often converged into that local optimum instead of converging into a better optimum.

In [33], several problems related to the learning stability of RL algorithms are discussed. Accordingly, a major problem is said to be the high correlation of the data used to train the agents. This problem can be easily solved in off-policy learning by a replay buffer, however, this possibility does not exist in on-policy algorithms. It is observed that the network architecture has a strong influence on the learning stability. This is also observed in the results of the hyperparameter optimization. Therefore it could be advantageous to use a network architecture which performs slightly poorer in average, but has a lower variance in the rewards. Another approach to minimize the variance is to scale the rewards. This scaling of rewards yielded different results. On the one hand, this only showed a positive effect for some environments, and on the other hand, the optimal scaling values were inconsistent. Nevertheless, this method could be applied to the problem at hand.

Another approach to improve the learning stability is presented in [34]. This approach uses a supervised pre-training for the ANN of a PPO agent. The supervised pre-training is intended to maximize the entropy of the action probabilities so that all actions are drawn with equal probability at the beginning of the training. Thus, the decision which actions are selected more often in which region of the state space is left to the algorithm. The objective of the supervised pre-training is to have the output vector of ANN be

$$\boldsymbol{a}^* = \frac{1}{d}[1, 1, ..., 1]^{\mathrm{T}} \in \mathbb{R}^d \tag{6.4}$$

where d is the number of available actions. This pre-training is intended to stabilize the training on the one hand, but also to increase the exploration on the other hand. Finally, the performance of the PPO agent should be improved, since the agent does not converge to local optima close to the initialization of the network. The stability could be significantly improved using this method in the experiments with various Atari games.

7 Performance of the controllers

7.1 Performance of the RL controller

Even though the rewards during the hyperparameter optimization vary to some extent, some of the trained agents have achieved a high reward. In the following, the performance of an exemplary agent with an excellent test reward is analyzed. The most important task of the controller is to select the voltage vectors in such a way that the torque follows the reference with minimal error.

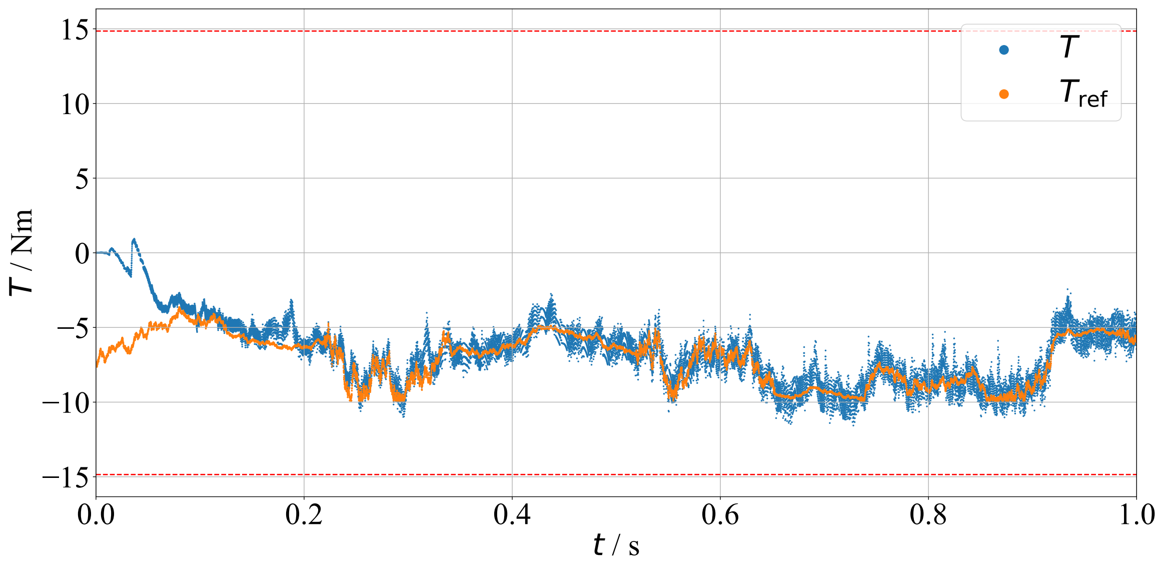


Fig. 7.1: Torque and reference torque

Figure 7.1 shows a plot of the torque and reference when controlling the motor with one of the RL agents, where the reference is randomly generated by a Wiener process. At the beginning, the RL controller needs a short time until the torque has increased up to the reference value, since a magnetic flux has to be built up first. After that, the torque follows the reference relatively well, but the torque oscillates slightly around the reference values. This is firstly because of the discrete action space, as this causes not only the voltage values to jump but also the current values to change quickly. Secondly, this is a result of the definition of the reward function, since it is incremented in a 10 % range around the reference value. The average of the absolute deviation of the torque from the reference is around 6 %. Thus, the RL agent has learned in the training to follow the reference trajectory very accurately.

The reference values used here are randomly drawn, but from a Wiener process, as it was the case in the training of the agent, so that the agent may have seen a similar reference trajectory in training. Since in training as many different operation points as possible should be passed through, this Wiener process was used, but often in later operation the reference torques are stepwise constant. Therefore, the RL controller should also be able to control these torques.

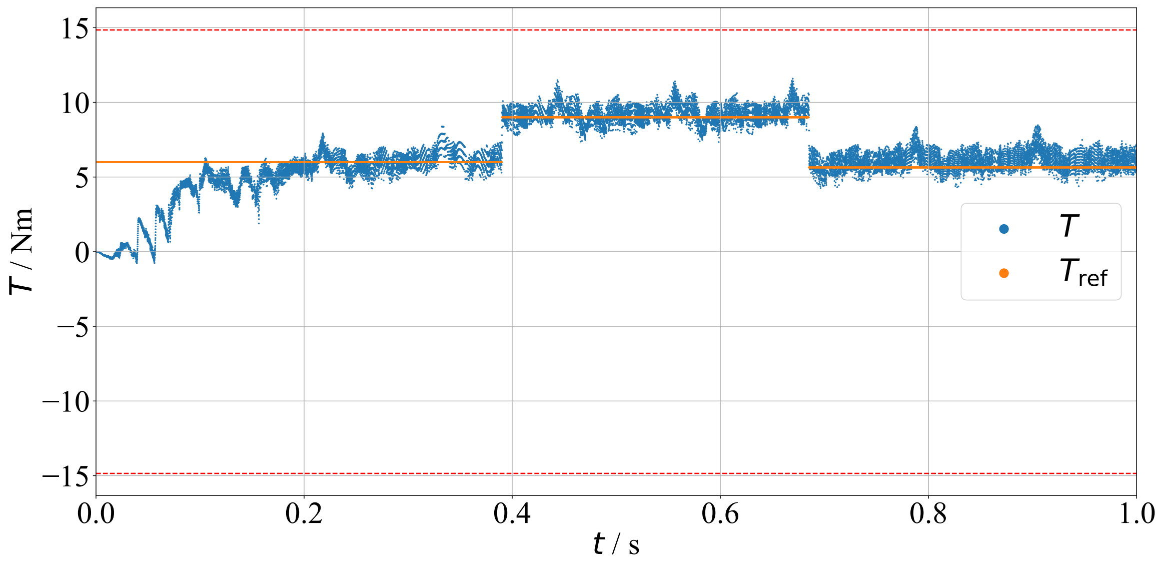


Fig. 7.2: Torque and a stepwise constant reference torque

Figure 7.2 plots the torque and a reference trajectory, which is stepwise constant. There is no qualitative difference seen between this and the previous plot. The absolute deviation of the torque to the reference is only slightly higher, with an average of 6.8 %. But this can be explained by a higher torque reference value at the beginning, which is why the RL controller needs a bit more time to reach this value. This allows the conclusion that the control works not only for Wiener references, but also for other types of reference trajectories.

Besides following the reference trajectory, the selected current operating point should also be efficient. The graph in figure 7.3 shows the stator currents of the motor when controlling a Wiener reference trajectory. At the beginning the i_{sd} current is close to the limit of 90 % of the maximum current. This is as expected, since a magnetic flux must first be built up by a positive i_{sd} current so that the required torque can be achieved. After

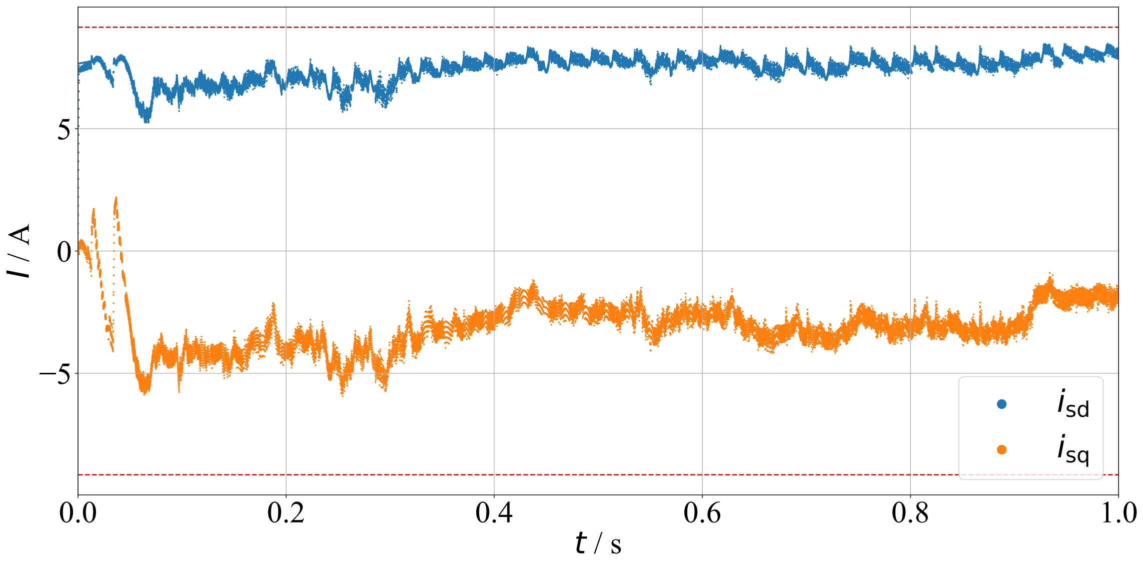


Fig. 7.3: Stator currents of the controlled motor

0.05 s, the i_{sd} current drops slightly and the i_{sq} increases to the negative, since the desired torque is negative. When the torque has reached the reference, both current components oscillate slightly. This oscillation can be attributed to the discrete voltage vectors at the input of the motor. For reference values that are high in magnitude, the i_{sd} current is slightly higher in magnitude than the i_{sq} current. However, if the torque reference value is closer to 0, such as around 0.4 s or at the end of the episode, the magnitude of the i_{sq} current decreases but the i_{sd} current increases. Expectedly, the i_{sd} current should also be reduced in order to reduce the input power. Instead, the magnitude of the input currents is close to the maximum current all the time.

Since the reference values change quickly due to the Wiener process, it might be advantageous for the RL controller to maintain a high magnetic flux in the motor, which is why the i_{sd} current is controlled to the maximum. In this way, only the i_{sq} current must be controlled according to the reference. This operates much faster compared to building up a magnetic flux. Figure 7.4 shows the magnitude of the magnetic flux. As already seen from the i_{sd} current, the magnetic flux increases rapidly at first, but then flattens out and converges to the steady-state flux for the current. Towards the end of the episode, the flux increases slightly again due to the increased i_{sd} current. To test the influence of the reference, the currents are examined at stepwise constant reference values. The measured currents show the same behavior as before. The i_{sd} current is at a maximum and the i_{sq} current controls the required torque.

Furthermore, this problem could happen due to the reference values used during the training process of the agent. Therefore, additional agents are trained with constant or stepwise constant reference values. The performance of these agents is significantly

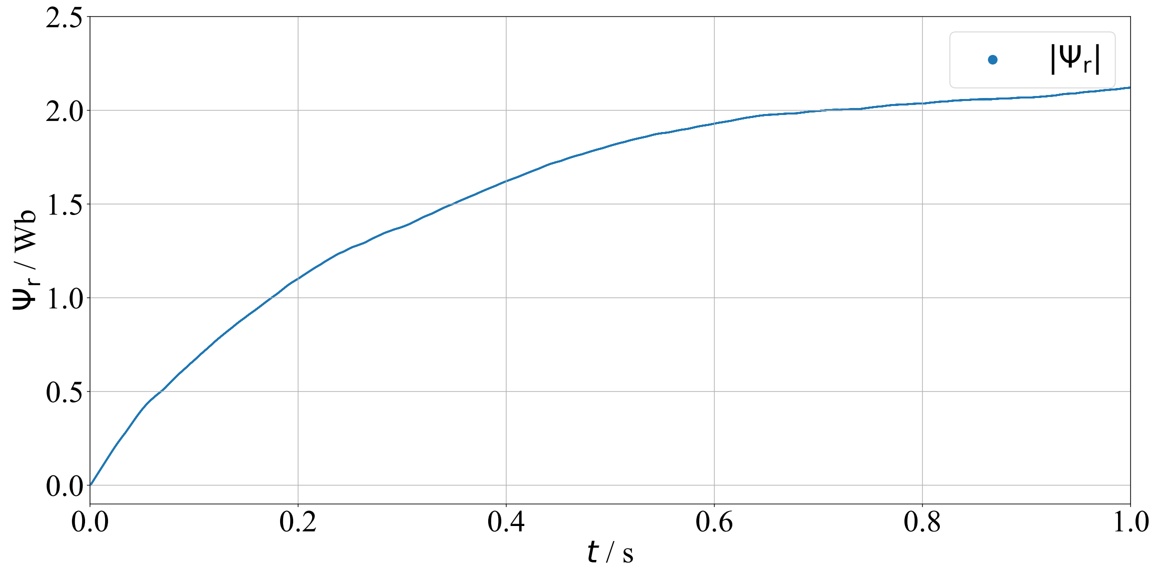


Fig. 7.4: Magnitude of the magnetic flux

lower than those trained with reference values generated by a Wiener process. The agents have problems to build up a magnetic flux and to control the torque. For this reason, the assumption that as many different operating points as possible should be seen during training is confirmed. The choice of the operation points could therefore not be improved.

Another element for the choice of the operation points is the reward function from equations (3.29) and (3.32). The reward is divided into two parts. First, a small deviation of the torque from the reference is rewarded. If the torque is in a 10 % range around the reference, the reward is increased by an efficiency term. In case of the input power and the output power being equal, the efficiency of the motor is 1. This theoretical maximum cannot be reached, because there are always ohmic losses in the motor, nevertheless the goal of a controller should be to maximize the efficiency.

Figure 7.5 shows three different graphs. The blue line indicates the total reward, the orange one the part of the reward resulting from the difference of the torque compared to the reference, and in green the part of the efficiency term. At the beginning, the total reward and the part due to the deviation are equal, since the deviation of the torque is greater than 10 % and therefore the efficiency part is 0. After about 0.05 s, the deviation of the torque is within the 10 % band, so the efficiency term has an influence on the total reward as well. The fraction of the deviation varies in the following most of the time between 0.9 and 1. In some cases, slight downward deviations can be seen. This was to be expected due to the definition of the 10 % band around the reference value. This observation confirms that the RL controller succeeds in making the torque follow the reference adequately.

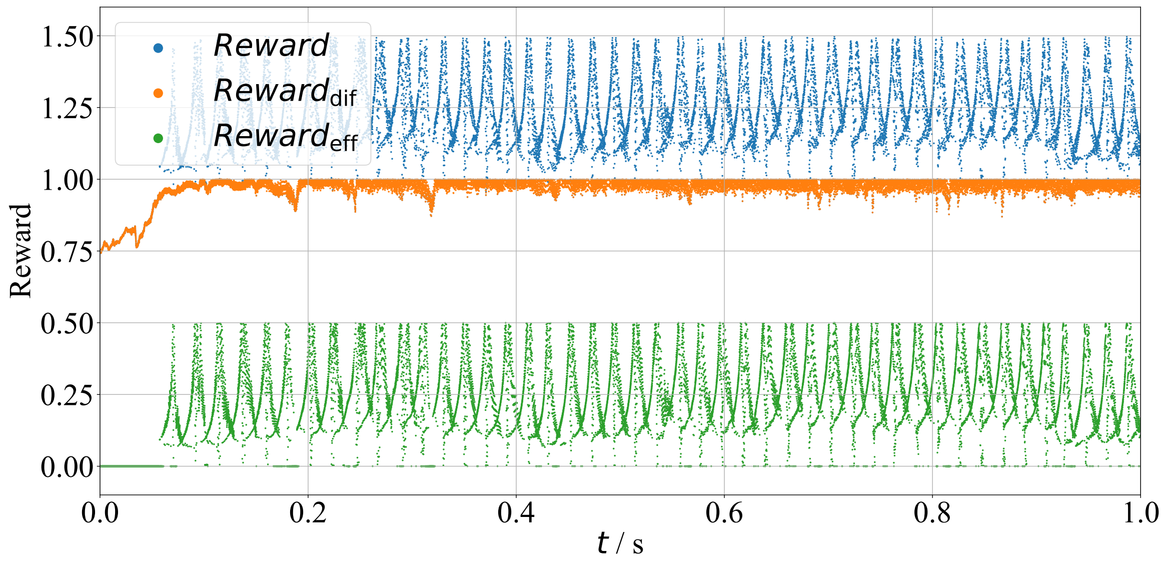


Fig. 7.5: Rewards of the episode

The total reward shows considerable oscillations, which are caused by the efficiency term. The reward of the efficiency term oscillates between 0.1 and 0.5, whereby 0.5 is the maximum due to the weighting factor. It can be seen that the minimum of the oscillation is lower for low reference values. This can be explained by the lower output power at these operating points, while the input power is constantly high. This high oscillation of the reward is disadvantageous for the training of an agent. Therefore, it must first be clarified what causes the oscillation. The efficiency term has a maximum when the input power and the output power of the motor are equal. The output power is the product of the rotational speed and the torque. The rotational speed is constant during an episode and the torque changes due to the changing reference, but not to the same extent as the efficiency term. For this reason, the oscillation of the efficiency term must be caused by the input power, which is the product of the input currents and input voltages. The power in the two-phase system is proportional to the power in the three-phase system, so it is sufficient to consider the power in dq coordinates. The currents in dq coordinates are approximately constant as seen in figure 7.3. The input voltages can only have 7 different values in the stator-fixed coordinate system. Due to the rotation of the dq coordinate system, intermediate values can also result in this coordinate system. Figure 7.6 shows the voltages of the episode in dq coordinates.

The u_{sq} voltage alternates between a positive and negative value. This causes the i_{sq} current to change quickly so that the torque also changes fast and follows the reference. In contrast, the u_{sd} voltage varies over the entire voltage range. This results in an oscillating input power. As expected, the frequency of the oscillating voltages corresponds to the frequency of the efficiency term and thus also to the frequency of the total reward. When

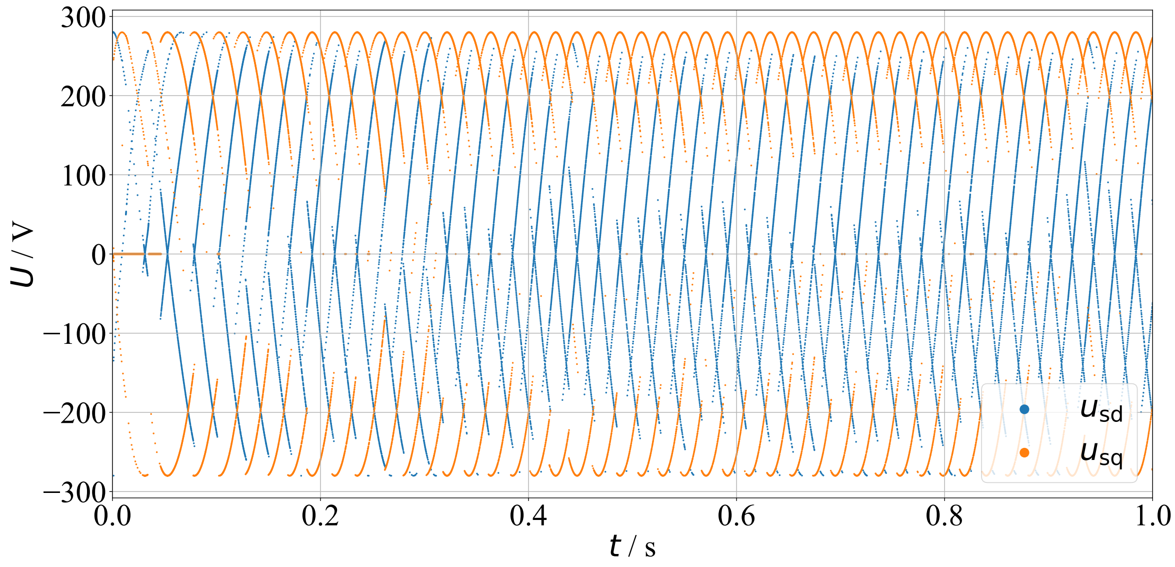


Fig. 7.6: Voltages of the episode in dq coordinates

considering the quantities in stator-fixed coordinates, the voltages only jump between the 7 different vectors, but the currents change sinusoidally.

To improve the training of the agent, a modified reward function is tested. The voltages oscillate strongly, but on average they go to 0. Therefore, the currents are the determining factor for the efficiency of the motor. Consequently, the magnitude of the current vector should be minimized in order to increase the efficiency of the motor. The efficiency term is changed to

$$r_{\mathrm{eff}} = c_{\mathrm{eff}} \left(1 - |\tilde{i}_{\mathrm{total}}|\right), \tag{7.1}$$

so that a small total current maximizes this term. The total current is normalized to the limit of the current so that the efficiency term is restricted to c_{eff}.

Using this adjusted reward function and the best hyperparameter set, 50 agents are trained. Figure 7.7 shows the torque and currents when controlling the motor with one of these RL agents. As before, the controller ensures that the torque follows the reference, with the exception of two short time periods. However, the main focus is on the selected current operating points. It can be seen that the i_{sd} current again reaches very high values and the i_{sq} current is adjusted in such a way to achieve the desired torque. The difference to the previous RL agent is that the i_{sd} current goes towards 0 phase-wise. This increases the reward for a short moment, but it would have been desirable for the i_{sd} current to be permanently lower. This stepwise decreasing i_{sd} current causes the magnetic flux to be reduced so that the agent is no longer able to adjust the reference torque by increasing the i_{sq} current. Therefore, the performance of this agent is ranked lower than the performance of the previous agent. Because of this, the previous agent is used for comparison with the FOC. There, the efficiency of the two controllers is compared besides other aspects.

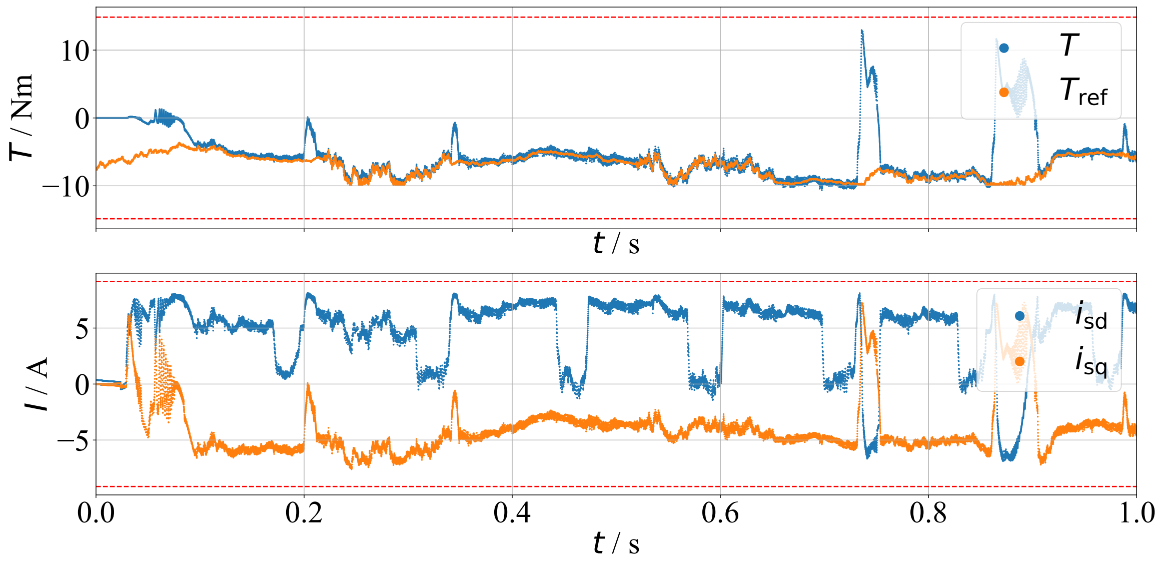


Fig. 7.7: Torque and currents of the trained agent

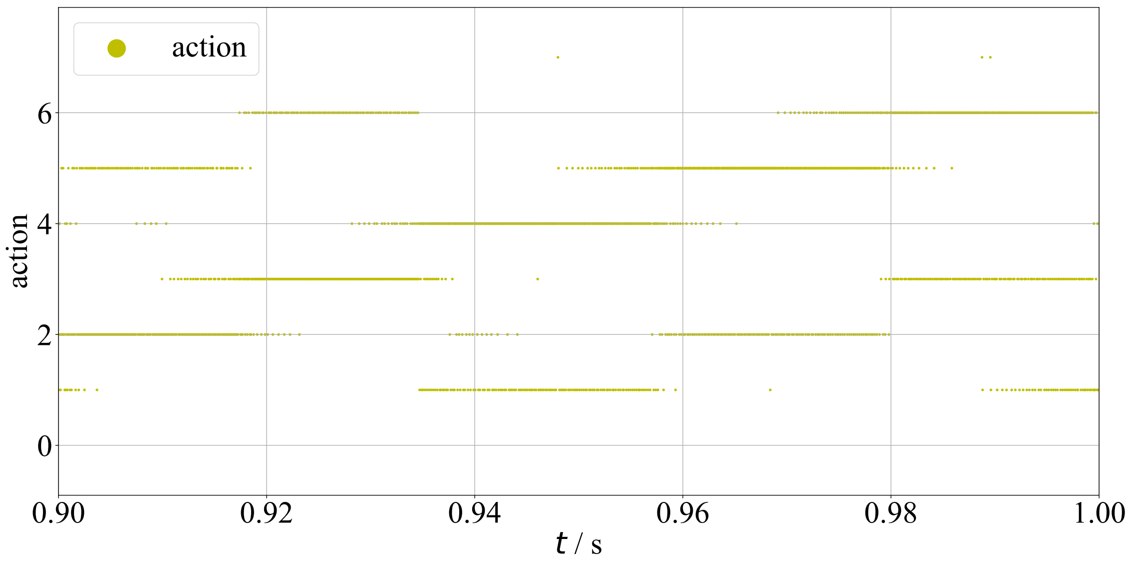


Fig. 7.8: Input voltage vectors

Lastly, the voltage vectors chosen by the agent are analyzed. These can be seen for a short period of time in figure 7.8. The actions always alternate for a short period between 0.02 s and 0.03 s across two different voltage vectors. Only rarely a different switching vector is selected in such a period. The frequency of the switching corresponds to the frequency of the change of the magnetic field. During one period, there is always switching between two opposite voltage vectors in the voltage hexagon. By switching between opposite voltage vectors, the current is held at a constant value. If the magnetic field has rotated further, the switching is performed between the next two opposite voltage vectors in the hexagon. This results in a staircase of actions, as expected, since such a switching scheme maximizes the magnetic flux. By switching on a zero voltage vector, the currents and the input power could be reduced, but a zero voltage vector is selected very rarely.

7.2 Comparison with a field-oriented controller

The state of the art for the control of a SCIM is the FOC. Therefore, it is reasonable to compare the performance of the RL controller with the FOC. The implementation details of the FOC are already explained in chapter 2. In contrast to the RL controller implemented here, the FOC requires knowledge about the motor parameters in order to set the parameters of the controller and the flux observer. Furthermore, the FOC uses a continuous action space. Intermediate voltage values are normally switched to the inputs of the motor by a modulation method such as pulse width modulation. The GEM toolbox, which is applied for the simulation of the drive system, uses an average modulation method instead. Therefore, there may be differences between the simulation and the behavior of a real motor. The FOC with the continuous action space thus has a small advantage compared to the RL controller, which uses a discrete action space. The comparison of the two controllers is consequently not fair in some aspects.

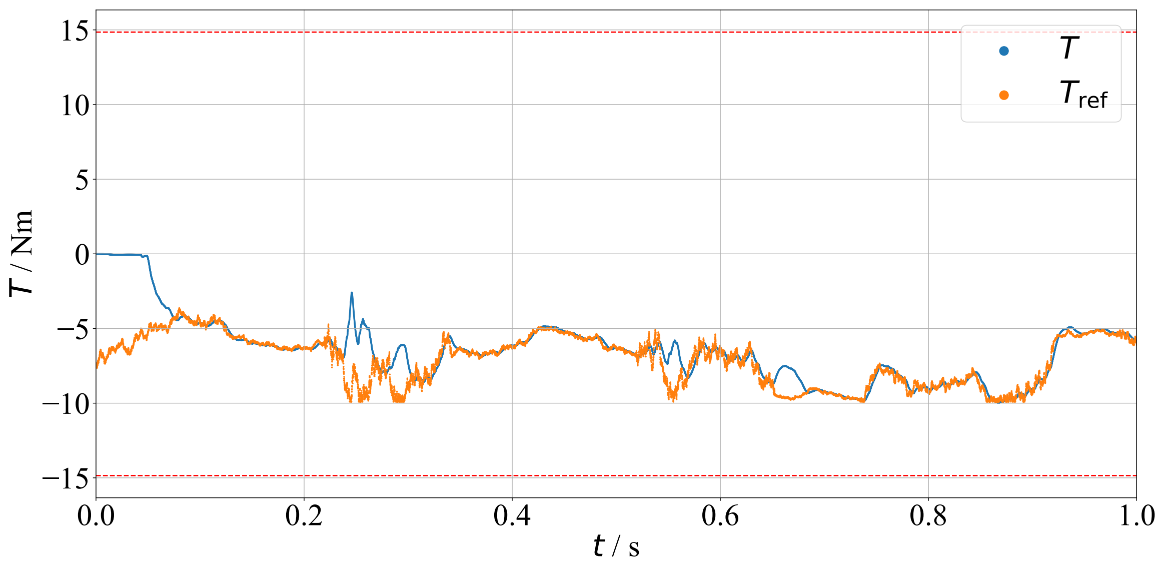


Fig. 7.9: Torque and reference of the field-oriented controller

For the implementation of the FOC, the Python toolbox gem-control is used, which provides classic controllers for various motors of the GEM toolbox. A detailed description can be found on the GitHub page [35]. Before the controllers are compared by different metrics, the performance of the FOC is briefly presented. Figure 7.9 shows the torque of the motor when using the FOC and the same reference trajectory as for the RL controller. At the beginning of the episode, there is no torque because a magnetic field has to be built up first. After about 0.05 s, the torque drops rapidly towards the reference. In the further process, the torque follows the reference trajectory with only slight deviations for most of the time. However, it can be seen when the reference value increases sharply in magnitude, the torque initially decreases. This is caused by the fact that the FOC keeps the flux in the motor relatively low for efficiency reasons. Thus, for steps in the reference, the magnetic flux first needs to be increased. To do this, the reference value of the underlying controller of the i_{sd} current is increased. Since this current is prioritized compared to the i_{sq} current during limitation, the torque initially drops even though the reference value increases. If this prioritization would not happen, the case could occur that the required magnetic flux could not be adjusted and thus the reference torque would not be reached.

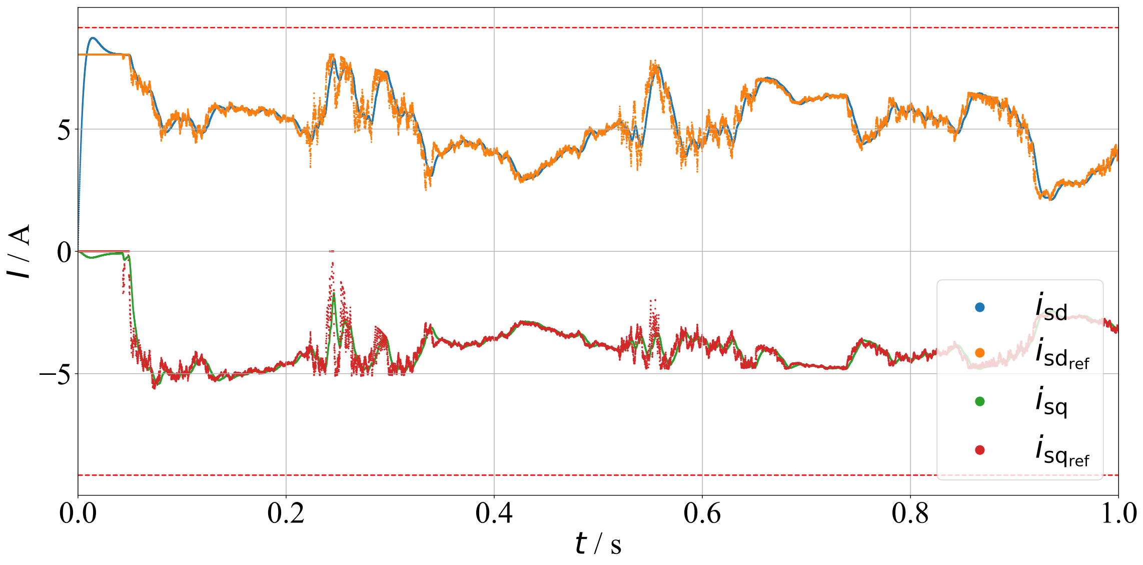


Fig. 7.10: Currents and current references of the field-oriented controller

The absolute magnetic flux is also lower compared to using the RL controller. A graph of this flux is plotted in figure 7.11. At the beginning, the flux of the IM also increases, however, it decreases again when the torque reference value is lower like at 0.4 s and at the end of the episode. The flux also increases very rapidly in some cases. Since the flux is too low to adjust the desired torque, the i_{sd} current is greatly increased. This results in the sharp increase of the magnetic flux.

The disadvantage of a small magnetic flux in the motor is that it needs more time to react to an increase of the reference value. In the following, the performance of the two controllers is compared by the squared control error. The quadratic control error indicates how closely the torque follows the reference. The lower this value, the better the torque

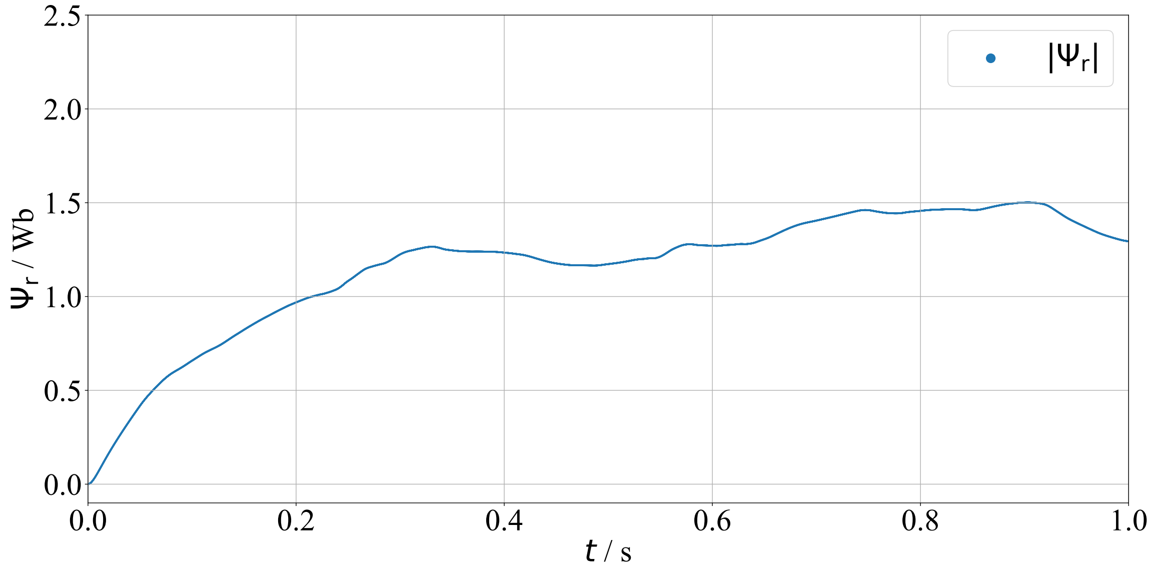


Fig. 7.11: Absolute flux of the field-oriented controller

follows the reference. Different types of reference trajectories are evaluated at different speeds. For the generation of the reference trajectories, on the one hand, a Wiener process is used again. On the other hand, step references, constant references and sinusoidal references are applied. Tables 7.1 and 7.2 show the mean squared control errors of the RL controller and the FOC for the different references and rotational speeds.

Speed	**Wiener**	**Step**	**Constant**	**Sinusoidal**
200 $\frac{1}{\text{min}}$	2.812	1.438	2.675	1.199
400 $\frac{1}{\text{min}}$	2.532	1.162	2.268	0.874
600 $\frac{1}{\text{min}}$	2.391	1.088	1.851	0.67
800 $\frac{1}{\text{min}}$	2.262	1.071	1.758	0.586
1000 $\frac{1}{\text{min}}$	2.212	1.174	2.152	1.167
Average	2.442	1.187	2.141	0.899

Tab. 7.1: Mean squared control errors of the RL controller in Nm^2

The average of the mean squared control error of the RL controller is lower than the mean squared control error of the FOC for Wiener references as well as for step references. In contrast, for continuous reference trajectories such as constant and sinusoidal references, the FOC performs better than the RL controller. These results are consistent with the observations from the analysis of the performances of the two controllers. In the case of the Wiener reference and the step reference, the reference values change sometimes very rapidly. As the magnetic flux in the motor is much higher when using the RL controller

Speed	**Wiener**	**Step**	**Constant**	**Sinusoidal**
200 $\frac{1}{\text{min}}$	2.608	1.668	0.786	0.329
400 $\frac{1}{\text{min}}$	2.579	1.627	0.793	0.329
600 $\frac{1}{\text{min}}$	2.552	1.578	0.802	0.333
800 $\frac{1}{\text{min}}$	2.187	0.933	0.814	0.342
1000 $\frac{1}{\text{min}}$	3.058	1.144	0.493	0.181
Average	2.597	1.39	0.734	0.303

Tab. 7.2: Mean squared control errors of the FOC in Nm^2

compared to the FOC, this RL controller can react faster to an increased reference torque. In the FOC, the i_{sd} current has to be increased first to build up a magnetic flux. In the mean time, the i_{sq} current must be decreased to maintain the current limitation. As a result, the torque decreases and the control error rises significantly. After the flux has been increased, the i_{sq} current and thus the torque will increase too.

In the case of constant and sinusoidal reference trajectories, the FOC has advantages because the reference value changes only slowly. Therefore, the magnetic flux has to change slowly as well to enable the torque to follow the reference. Furthermore, the FOC benefits especially from the continuous action space for these references. While the torque of the RL controller can only oscillate around the reference value, the currents and thus the torque can change continuously, so that the control error in steady state can be close to 0. In addition, the I-component of the PI controller ensures that there is no steady-state control error. This explains the large differences between the two controllers, in particular at constant reference values.

The speed of the rotor was also varied in addition to the reference trajectories. First of all, it is pleasant to see that there is no large deviation in any of the measured values. This indicates that the controllers work suitably at several different speeds. Nevertheless, slight tendencies can be seen, especially for the RL controller. The smallest control errors are observed in the middle speed range. This is to be expected, since the speed was randomly selected in the entire speed range during the training. Since one policy is used to select the actions, it is optimized for a wide range, with the optimum being approximately in the middle. There is no distinct trend for the FOC. For the Wiener reference, the mean squared control error is the largest at the highest speed, while these are the lowest for the continuous reference trajectories.

In summary, the RL controller has been shown to follow the torque reference quite well. Although the comparison is not fair in favor of the FOC due to the different action spaces, the mean squared control error of the RL controller was lower for fast changing references. The FOC, on the other hand, has its strengths for slowly changing and constant reference trajectories, since the magnetic flux needs to change only slowly there.

Another metric for the comparison of controllers is the step response and especially the rise time. This rise time indicates how long a controller needs to react to a step in the reference. When analyzing the control error, it was already observed that the RL controller was better in handling steps in the reference than the FOC. However, for the investigation of this rise time some differentiation is necessary. At a step change in the reference, most of the rise time is needed to build up a magnetic field. The RL controller already builds up a magnetic field when the reference is 0. In contrast, the FOC does not build up a magnetic field until the torque reference is unequal to 0. Thus, on the one hand, the response behavior can be analyzed if the step in the reference occurs at the beginning. On the other hand, the response is analyzed if the step in the reference happens after a certain period of time, when there is a stationary magnetic field in the rotor. The step height and the rotational speed are set to a medium value for the experiments. No qualitative differences could be observed when varying these parameters. Furthermore, the rise time is defined as how long the controller needs until the torque reaches the 10 % band around the reference value to ensure a fair comparison.

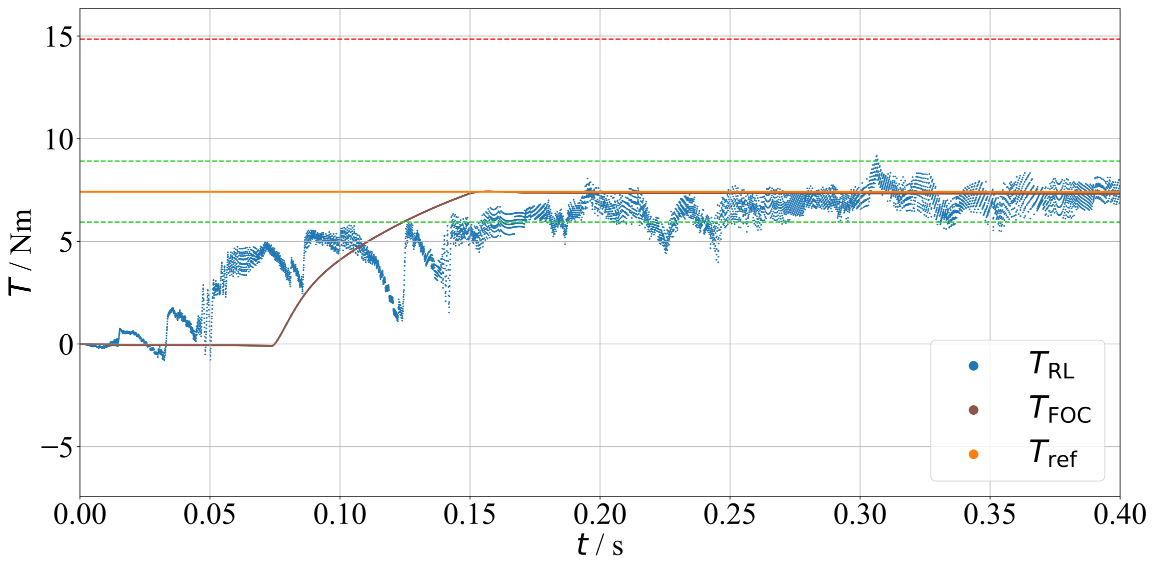


Fig. 7.12: Step response of the RL controller and the FOC

Figure 7.12 shows the step response of the RL controller in blue. The graphs of the corresponding currents can be found in figure A.32 in the appendix. It can be seen that the RL controller initially has slight problems reaching the reference torque. The i_{sd} current is high at first to build up a magnetic flux. As the i_{sq} current starts to increase significantly after about 0.05 s the i_{sd} current must be decreased to maintain the limitations. However, since the magnetic flux in the motor at this point is obviously not yet high enough to reach the reference value, the torque fluctuates widely. After 0.1428 s, the torque is within the 10 % band for the first time. In the following, the torque oscillates around the reference value and leaves the 10 % band for short time periods.

Figure 7.12 also presents the corresponding graph for the FOC in brown. The plots of the currents can be seen in figure A.33 in the appendix. The torque is initially zero for 0.08 s before it increases sharply and then flattens out. After approximately 0.14 s, the torque reaches the reference value and does not deviate from it afterwards. Because the input voltages are constant there are no oscillations in the torque. At the beginning of the episode the i_{sd} current is at a maximum to build up a magnetic flux. If a sufficient flux has been built up, the i_{sq} current is increased to increase the torque to the reference value. The 10 % band is reached after 0.1169 s. Thus, the rise time of the FOC is less than for the RL controller in this case. In addition, the torque reaches the reference on a more stable path than with the RL controller.

As the reference value often makes a step at a later time during the operation, a further test is executed. The reference value is zero for 0.3 s and then makes a step to a constant value. Within this time, the controller has time to build up a magnetic flux in the motor. As the FOC does not adjust any magnetic flux at a reference value of zero for efficiency reasons, the response time of the FOC does not change. However, it is possible to provide a minimum flux in the motor to reduce the rise time. For a large step, as in this case, the reduction would be small, because the required flux would be significantly higher than the minimum flux.

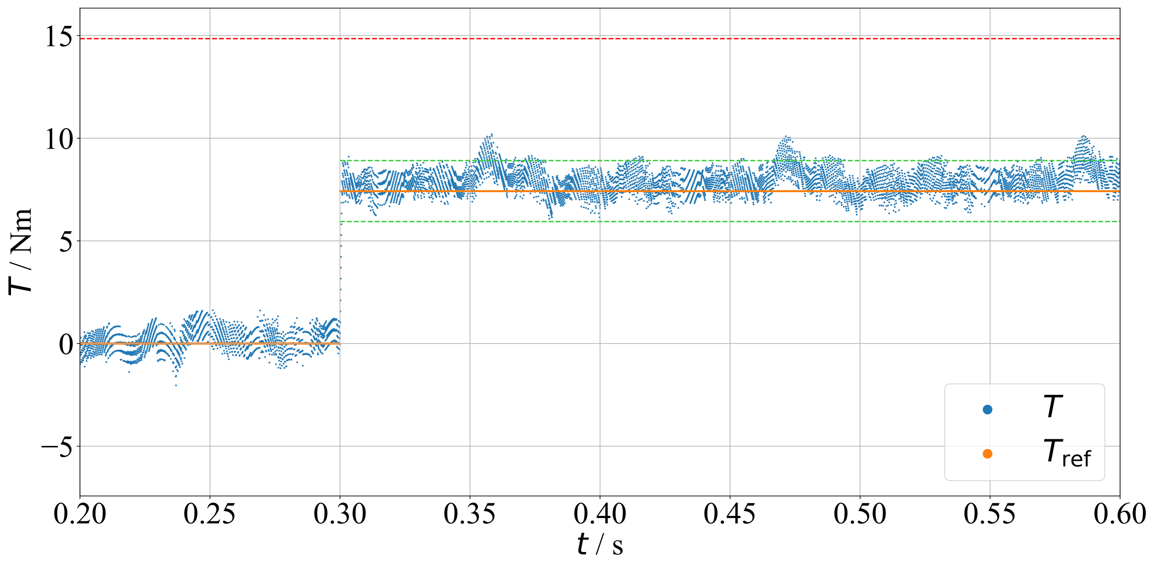


Fig. 7.13: Step response of the RL controller after 0.3 s

In contrast to the FOC, the step response of the RL controller is changed if the reference is initially zero and the step occurs after 0.3 s. Figure 7.13 shows the torque and the reference trajectory. It can be seen that the torque fluctuates around the value 0 at the beginning. After 0.3 s the reference value increases. The torque needs only a few time steps to reach the 10 % band around the reference. The rise time is only 0.6 ms, which is several orders of magnitude smaller than for the FOC. This behavior can be explained by the different time constants of the SCIM. The time constant of the current is much smaller than the time constant of the magnetic flux. Both, a magnetic flux and a i_{sq} current are needed to generate a torque. In the plot of the corresponding currents, which can be found in figure A.34 in the appendix, it can be seen that the i_{sd} current is close to the limitation while the reference is 0. Thus, a magnetic flux is built up. If the reference value now increases, only the i_{sq} current has to increase. This current can increase very quickly due to the small time constant, so that the rise time is short.

The rise time of the RL controller depends strongly on the time the controller has to build up a magnetic flux before the reference changes. The control of a motor normally happens time-continuous and not episodic. Therefore, the RL controller usually has enough time to build up a magnetic flux and reduce the rise time. Otherwise, the step response of the FOC is superior to that of the RL controller. However, for efficiency reasons, it is not advantageous to permanently maintain a high i_{sd} current to induce a high magnetic flux. For this reason, the efficiency of the SCIM is examined in the following when using the two controllers.

Besides following the reference, maximizing efficiency is also an aspect of a controller's performance. A high efficiency is especially advantageous for mobile applications, but also for stationary applications the efficiency should be as high as possible. The efficiency

$$\eta = \frac{P_{\mathrm{out}}}{P_{\mathrm{in}}} \tag{7.2}$$

is calculated as the quotient of the output power P_{out} and the input power P_{in}. The input power corresponds to the supplied electrical power from equation (2.18). At the output of the motor, the power is absorbed by the mechanical load. The output power is therefore the mechanical power, which is the product of the rotational speed and the torque. In chapter 2 it has already been shown how the optimum operating point is calculated in respect of the efficiency. Thus, the maximum efficiency of the motor can be determined. However, this maximum efficiency is only valid for the steady state, i.e. at constant speed and constant torque. Therefore, such an operating point is selected for the measurement of the efficiency. The theoretical maximum efficiency only depends on the speed. The efficiency increases with increasing speed. Since the required currents are constant at the same torque and the voltages are only partly dependent on the speed, the required electrical power increases more slowly than the mechanical power. The mechanical power increases linearly with increasing speed. In contrast, the maximum efficiency is independent of the torque. The currents and thus the input power increase linearly with higher torque, but the output power also increases linearly. Therefore, the quotient of the two quantities is constant.

To measure the efficiency of the two controllers, a constant reference with a value of half the torque maximum is used. As the efficiency depends on the rotational speed, this was varied over the entire speed range. The input power and the output power can be determined directly from the measured quantities. To ensure that all quantities are in a steady state, a start-up phase is waited to stabilize all quantities before measurements are started. Figure 7.14 illustrates the measured efficiencies of the two controllers and the optimal efficiency.

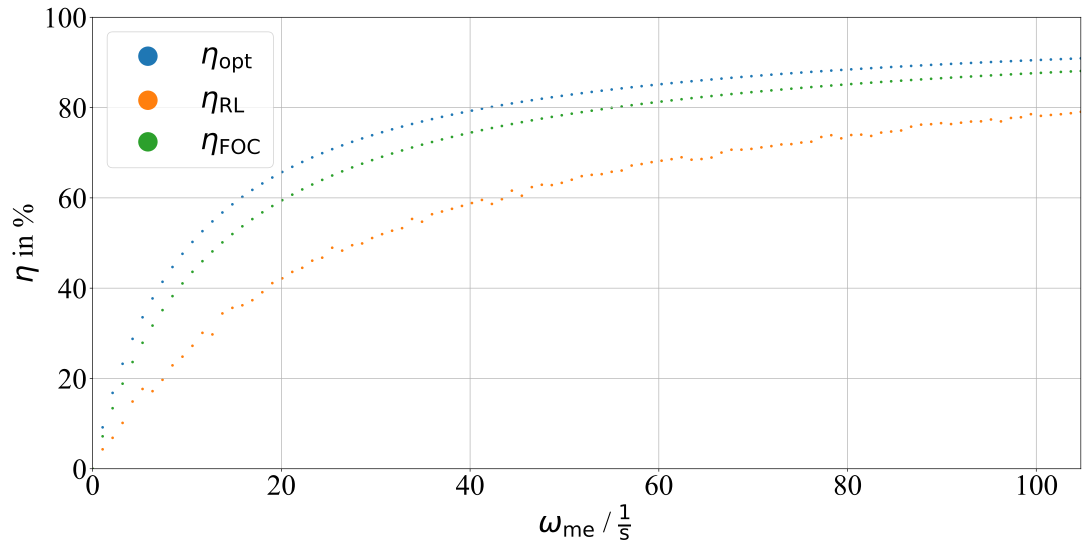


Fig. 7.14: Efficiency of the controllers

The optimum efficiency is plotted in blue. It starts from the origin and rises sharply at first. Subsequently, the optimal efficiency flattens out continuously. The measured efficiency of FOC, which is displayed in green, shows a similar curve. However, the efficiency at each point is slightly below the theoretical maximum efficiency. In principle, the FOC tries to adjust the optimal steady-state operating point. As, for example, look up tables are used in the operation point selection of the FOC, slight deviations from this optimal operation point can result. This explains the deviation of the efficiency. As expected, the efficiency of the RL controller is significantly lower. Over large parts of the speed, the efficiency is more than 20 % lower than the optimal efficiency. This can be explained by the fact that the selected i_{sd} current is very high when using the RL controller. Nevertheless, in principle, an increase in the efficiency with increasing speed can be seen. The higher the speed gets, the smaller becomes the difference between the optimal efficiency and the efficiency of the RL controller. Looking at the currents at high speed, it can be seen that the i_{sd} current drops temporarily. It can be observed that the safety layer is repeatedly active, but not more frequently than at other speeds. In addition, the current drops to such an extent that the safety layer no longer intervenes. The agent has apparently learned that a temporary reduction of the i_{sd} current leads to a higher reward caused by the efficiency term. However, a permanently lower i_{sd} current would be preferable to improve the efficiency constantly.

The efficiency of the optimal operating point is independent of the torque value, since the input power increases linearly with the currents and the output power increases linearly with the torque. To check whether this is also the case for the efficiency of the two controllers, two additional test series were measured for each controller. The reference value was chosen at 25 %, 50 % and 75 % of the maximum torque. As expected, the efficiencies of the FOC at different speeds are almost identical. This is not the case for the RL controller. As can be seen in figure 7.15, the efficiency increases with an increasing torque reference. For higher reference values, the i_{sq} must also be higher. Consequently, the i_{sd} current has to be smaller in order to maintain the limitation of the currents. As this i_{sd} current is normally higher than the optimal value when using the RL controller, it is forced to get closer to the optimal value. Therefore, the efficiency of the RL controller is improved at higher torques.

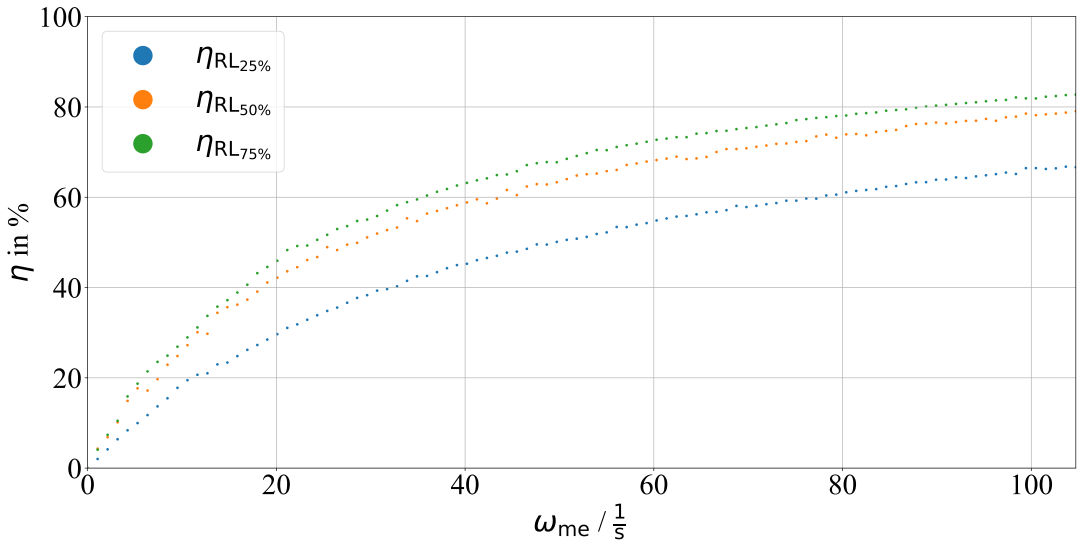


Fig. 7.15: Efficiency of the RL controller at different torques

It can be summarized that the RL controller is able to follow a torque reference quite well. Especially in the case of fast changing references, it reacts considerably faster than the FOC. The rise time of the RL controller is thus short. The reason is that the i_{sd} current is adjusted to the maximum at each time, resulting in a large magnetic flux. Because of the smaller time constant of the currents, the torque can be adjusted very quickly. However, this high current causes the efficiency of the motor to be much lower when using the RL controller compared to the FOC. This increases forcibly with increasing torque. Nevertheless, it has been shown that a functioning RL agent has been trained to control a SCIM.

8 Conclusion and Outlook

8.1 Conclusion

The goal of the thesis was to develop a method for the control of an IM, which operates without the knowledge of the motor parameters. A data-driven controller is trained to learn an appropriate policy for the determination of the input voltages. The problem was divided into three subtasks. First, since the magnetic flux of the SCIM is not measured, it had to be estimated. The classical flux estimators need the motor parameters which is why these methods cannot be used. It has been shown that an identification of the parameters of the derivative of the magnetic rotor flux provided excellent results in the simulative tests. By integrating this derivative, very accurate estimates can be obtained. This flux estimation method could also be used for other controllers, such as an FOC or an MPC.

Furthermore, when using an RL agent to control a system, the limitations cannot be taken into account. Therefore, a model predictive safety layer was utilized to prevent unsafe actions from being performed. The underlying system model of the safety layer is identified in parallel with the training of the RL. It has been shown that the accuracy of the estimated parameters of the model increases with the magnetic flux. A system model has been successfully found which estimates the following state sufficiently good to prevent the limits from being exceeded. The identified system model could also applied in an MPC.

The third subtask was the optimization of the RL agent. For this purpose, a hyperparameter optimization was performed. This included the investigation of the RL algorithm. The results show that the PPO algorithm is best suited for this problem. One problem of the hyperparameter optimization was the high variance of the obtained results. This problem is not specific to this control task, but represents a general difficulty of RL agents.

The performance of the best agent of the hyperparameter optimization was compared with that of an FOC. The RL controller achieves a similar performance as the FOC in terms of the control error. However, the RL controller adjusts the magnetic flux to the maximum at each time, which is why the efficiency of this controller is around 20 % lower than that of the FOC.

In conclusion, a functioning RL agent has been trained to control a SCIM. The RL controller follows the reference very well, but the efficiency needs to be improved before the control method can be used in economic applications. A model predictive safety layer has been implemented to ensure that the system limitations are respected when using an RL agent. Also, a parameter independent method for the estimation of the magnetic flux was presented.

8.2 Outlook

In this thesis a new method for the control of a SCIM was presented. Until now, this method has only been tested simulatively. Therefore, the next step is to implement this controller on a real test bench. The implementation of the RL controller can be realized by using edge computing. Figure 8.1 shows a flowchart of this process.

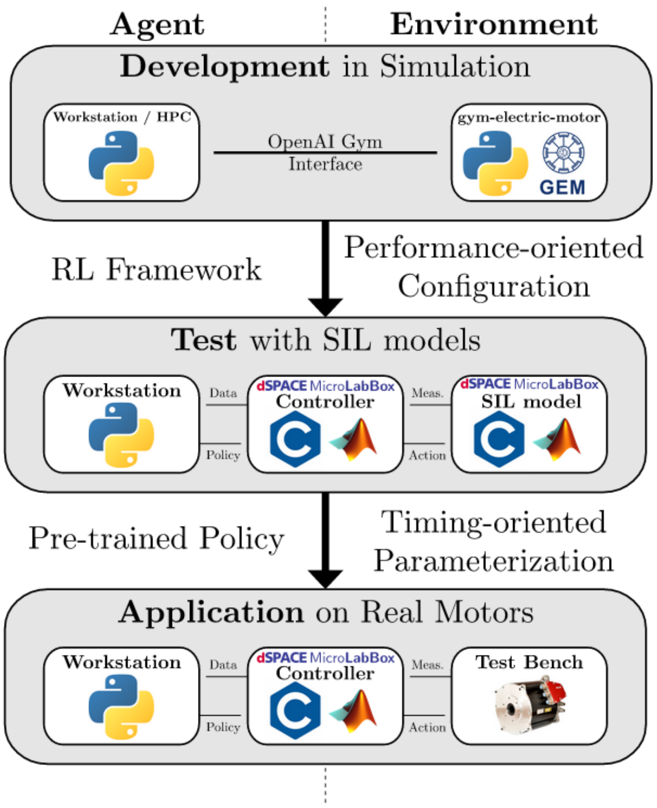

Agent
Environment
Development in Simulation
Workstation / HPC
OpenAI Gym Interface
gym-electric-motor
GEM
RL Framework
Performance-oriented Configuration
Test with SIL models
Workstation
Data
Policy
dSPACE MicroLabBox Controller
Meas.
Action
dSPACE MicroLabBox SIL model
Pre-trained Policy
Timing-oriented Parameterization
Application on Real Motors
Workstation
Data
Policy
dSPACE MicroLabBox Controller
Meas.
Action
Test Bench

Fig. 8.1: Flowchart of the implementation on a real test bench (cf. [36])

In a first step, the performance of the RL controller is optimized simulatively. This task has already been finished by the hyperparameter optimization. The RL controller is then transferred to the actual hardware that should be used. All processes that have to be performed in real time, such as estimating the magnetic flux and calculating the actions, are executed on a dSPACE MicroLabBox. The training can be performed asynchronously on a workstation. Nevertheless, the system is still simulated. In a final step, this simulation is replaced by the real test bench.

In comparison to simulation, there are some additional aspects to be considered on the real system. The simulation uses an idealized system model, but in reality there are deviations due to nonlinearities and parasitic effects. In addition, the measurement signals will be noisy, so the RL controller has to learn to deal with uncertainties. Besides the RL controller, the identification of the system model for the safety layer and the identification of the derivative of the magnetic flux for the flux estimator must also handle non-ideal conditions. Whether these identifications will also work with noisy data has to be checked. The performance of the RL agent can be compared again with the FOC after training.

One problem of the trained RL agents was their low efficiency. The RL agent increases the i_{sd} current to a maximum to be able to react fast to a changing reference trajectory. This increases the reward in the short term. In the long term, the reward could be increased by reducing the i_{sd} current again. The time constant of the magnetic flux is much larger than the time constant of the currents. Thus, a change in the i_{sd} current has a much delayed effect on the torque. To overcome this problem, a cascaded control could be helpful, similar to the FOC. First, an RL agent has to be trained to control the current. The cost function of this agent would be the deviation of the currents from the reference currents. On the one hand, these reference currents can be produced by a reference generator. On the other hand, these reference values can be the outputs of a superimposed torque controller.

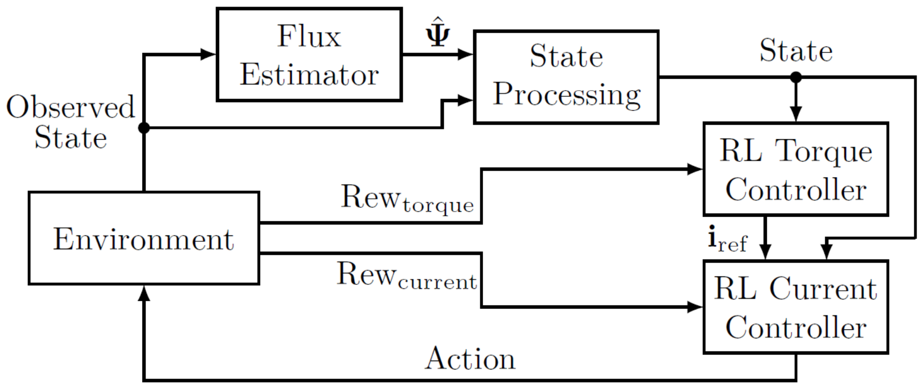


Fig. 8.2: Structure of the cascaded RL controller

The superimposed torque controller is also realized by an RL agent. The reward function of this agent would have a similar form as the one that has already been used here. On the one hand, the operating point should be chosen in such a way that the desired torque is reached, and on the other hand, the efficiency of the operating point should be maximized. By splitting the controller, it is now possible to choose different sampling times for the two controller stages. These should correspond to the time constants of the controlled variables. In this way, the observed data points of the torque controller are less correlated. The question remains whether the current controller should be trained first or if it is possible to train both agents in parallel. The torque controller needs a well trained underlying RL current controller. Moreover, it can be assumed that both agents need a similar number of data points for the training. As the sampling times are different for the same training duration the current controller will get significantly more data points. On the other hand, it can be argued that the current controller is trained significantly faster than the torque controller due to the shorter sampling time, so that a parallel training would result in a small time reduction. Figure 8.2 shows the structure of the cascaded RL controller. Whether such a cascaded structure solves the efficiency problems of the RL controller remains to be shown.

Appendix

A.1 Results of the system model identification

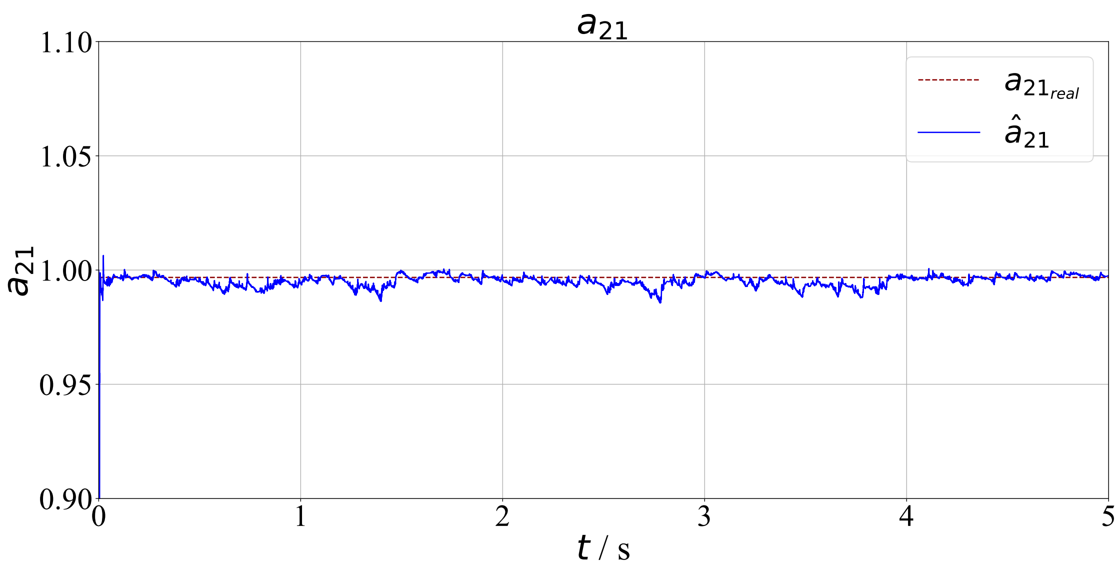


Fig. A.1: Trajectory of the estimated parameter $\hat{a}_{21}$

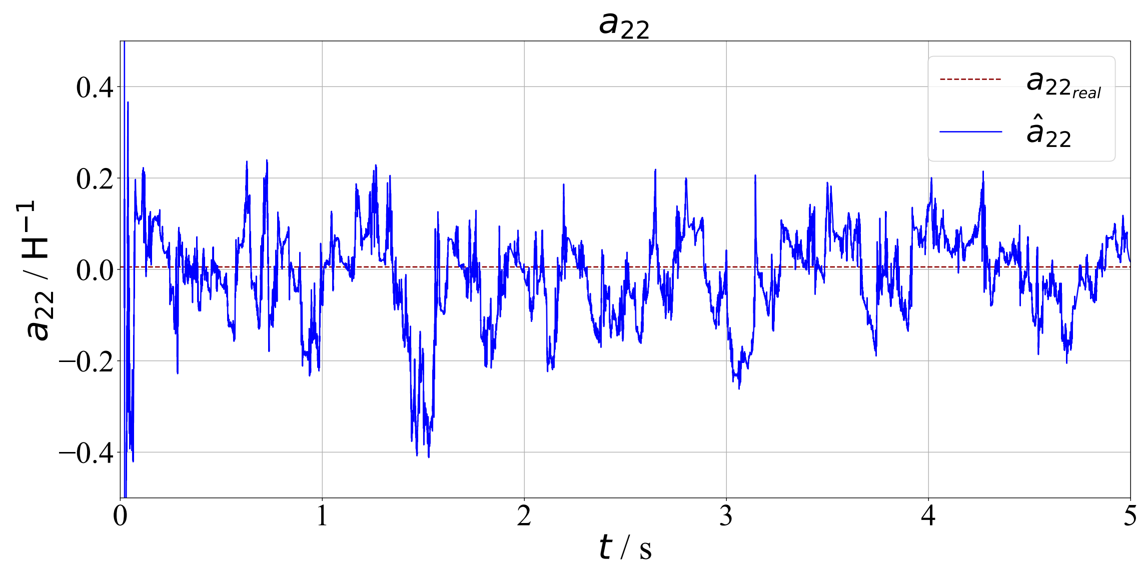


Fig. A.2: Trajectory of the estimated parameter $\hat{a}_{22}$

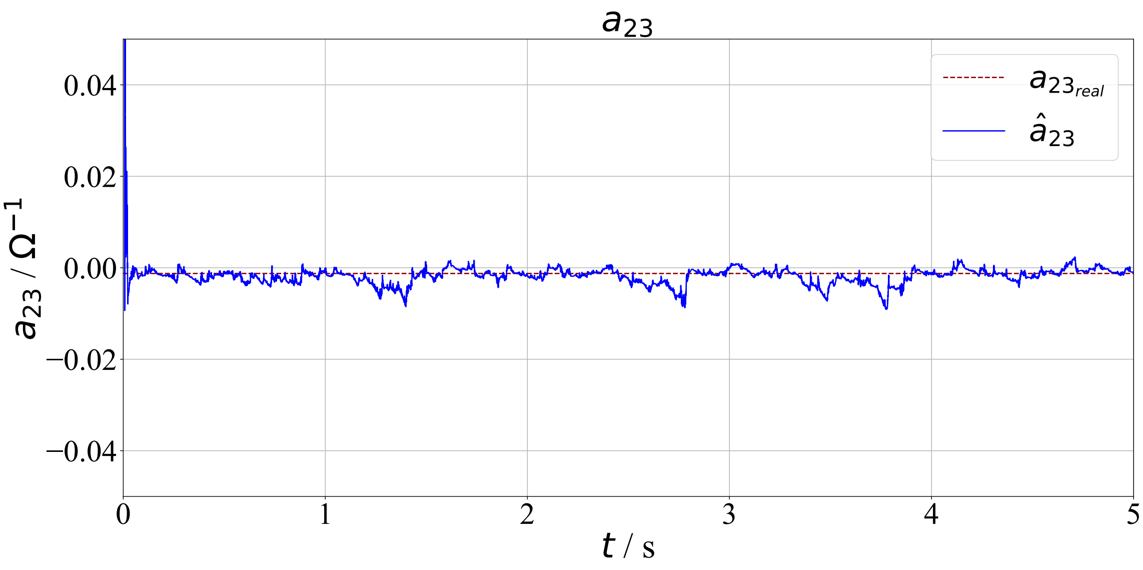


Fig. A.3: Trajectory of the estimated parameter $\hat{a}_{23}$

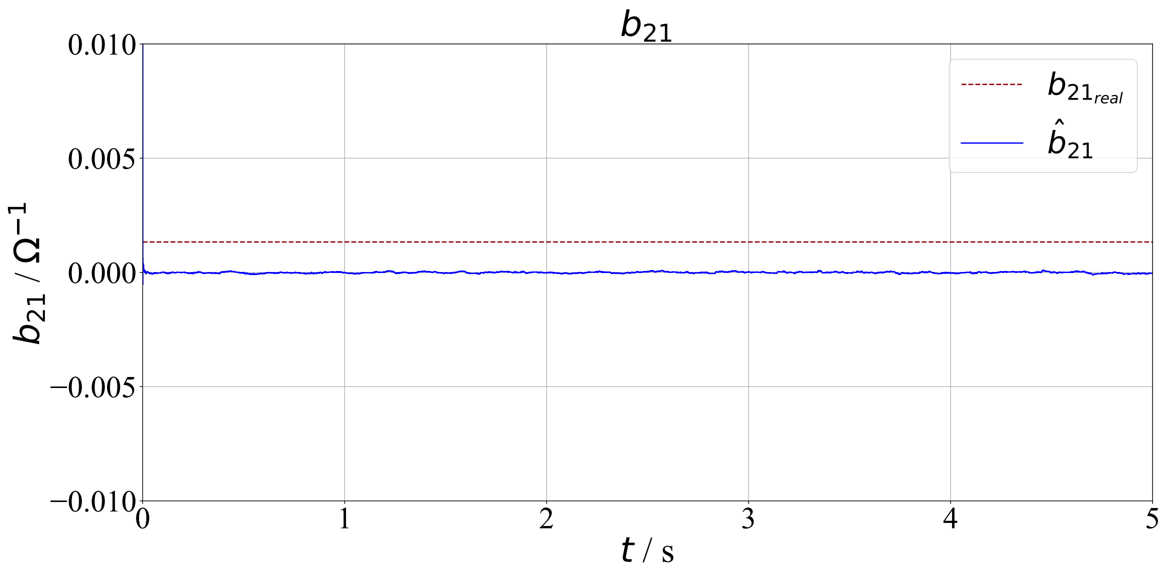


Fig. A.4: Trajectory of the estimated parameter $\hat{b}_{21}$

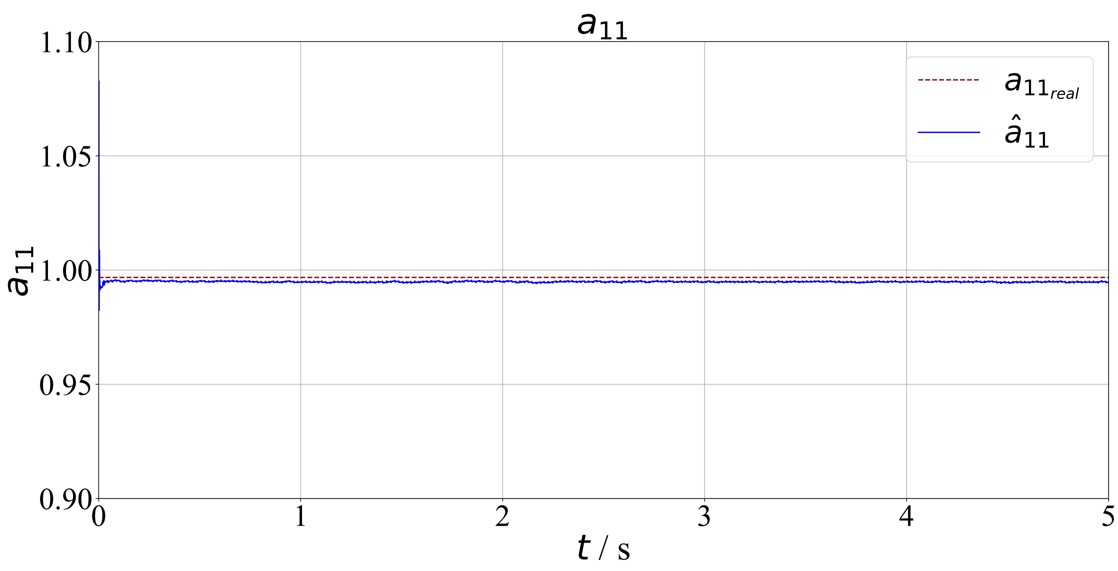


Fig. A.5: Estimated parameter $\hat{a}_{11}$ selecting a voltage vector for 50 time steps

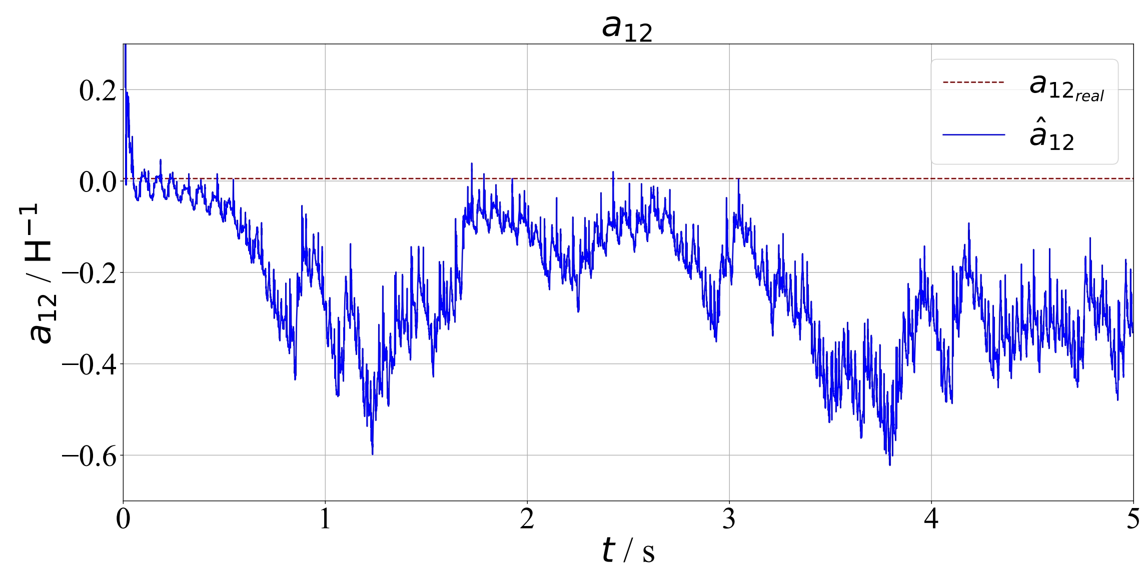


Fig. A.6: Estimated parameter $\hat{a}_{12}$ selecting a voltage vector for 50 time steps

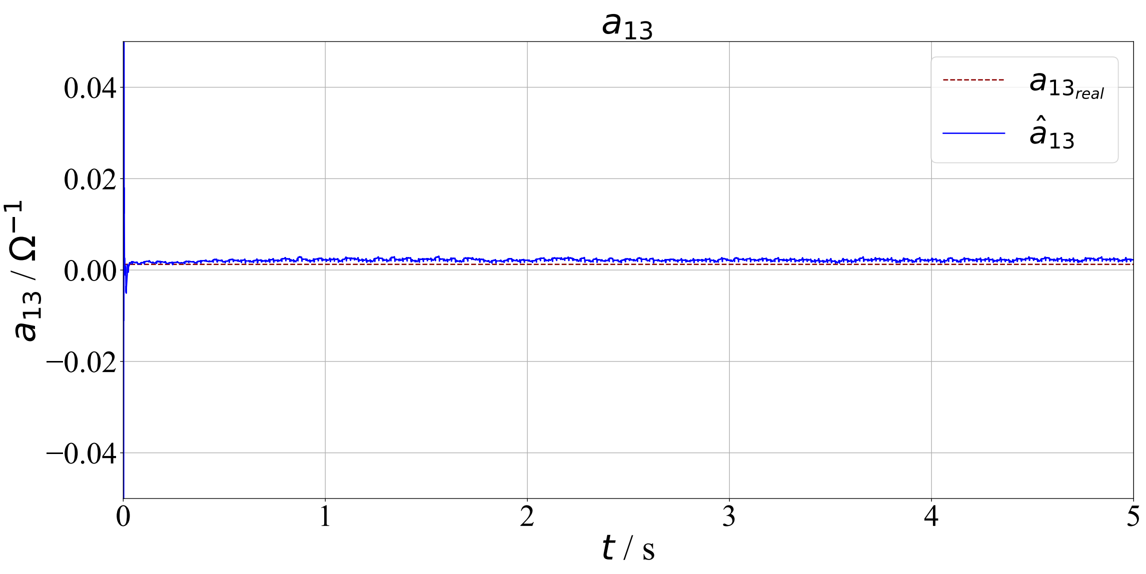


Fig. A.7: Estimated parameter $\hat{a}_{13}$ selecting a voltage vector for 50 time steps

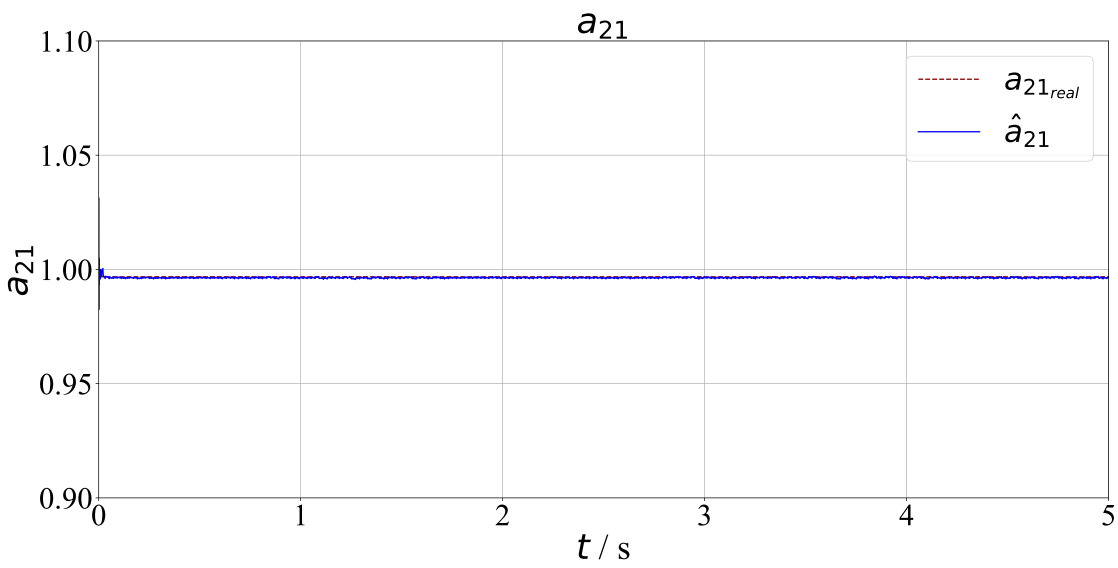


Fig. A.8: Estimated parameter $\hat{a}_{21}$ selecting a voltage vector for 50 time steps

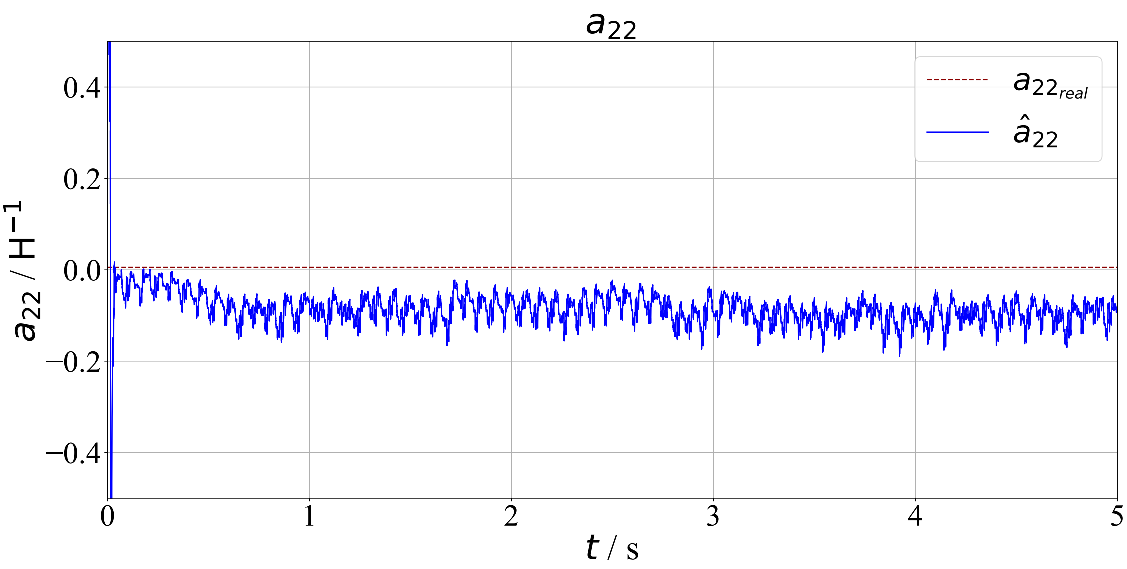


Fig. A.9: Estimated parameter $\hat{a}_{22}$ selecting a voltage vector for 50 time steps

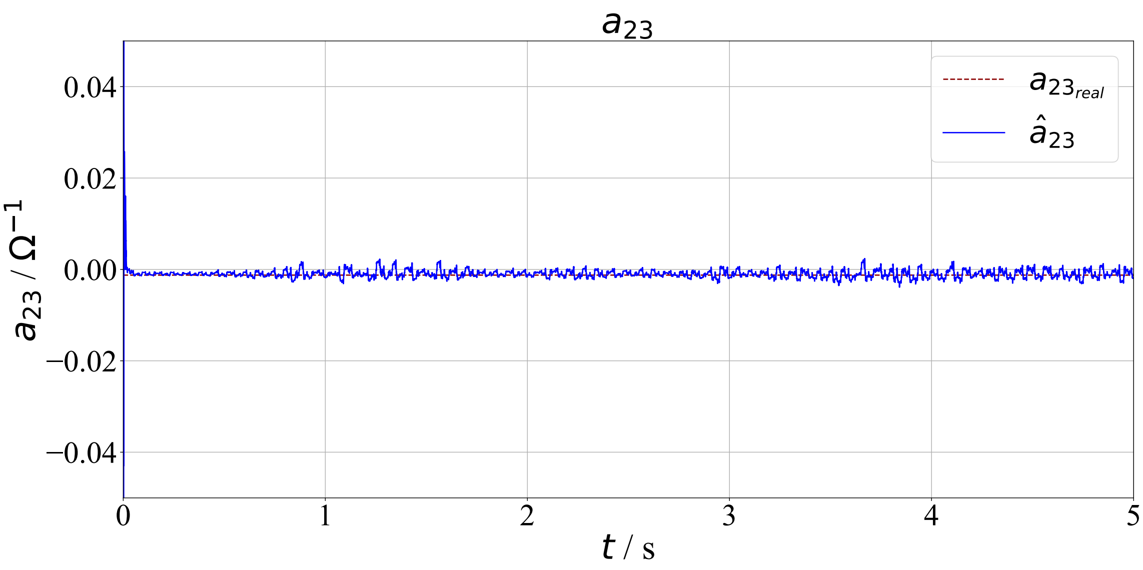


Fig. A.10: Estimated parameter $\hat{a}_{23}$ selecting a voltage vector for 50 time steps

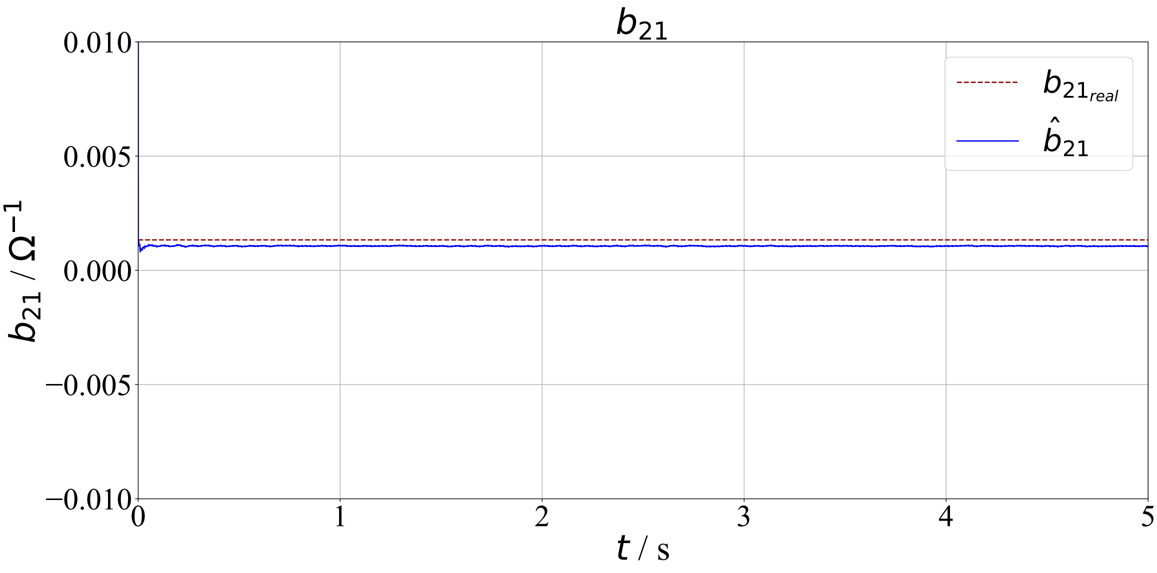


Fig. A.11: Estimated parameter $\hat{b}_{21}$ selecting a voltage vector for 50 time steps

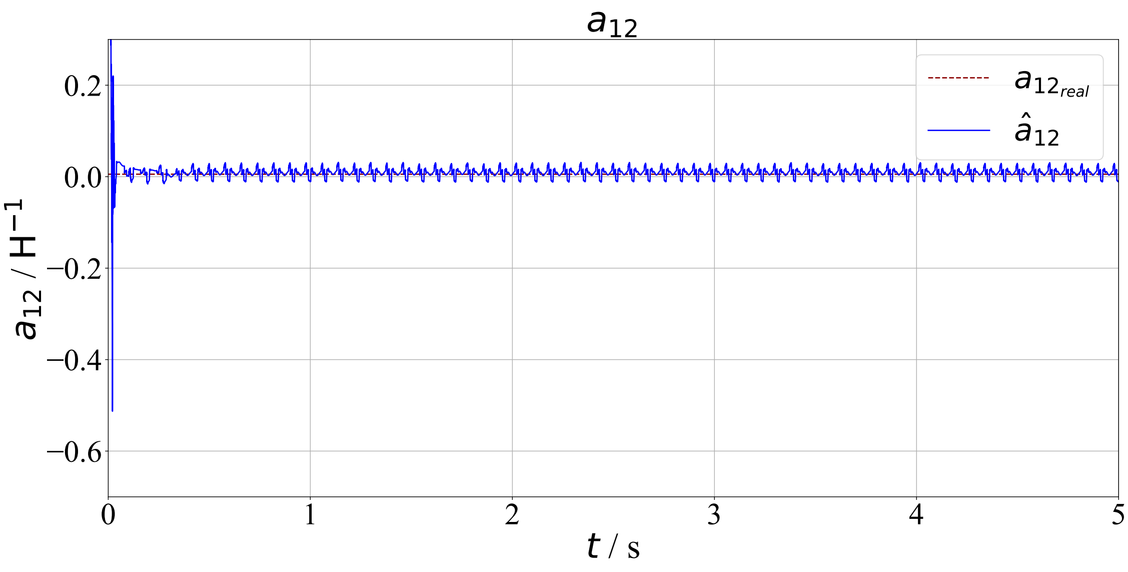


Fig. A.12: Estimated parameter $\hat{a}_{12}$ selecting a voltage vector for 200 time steps

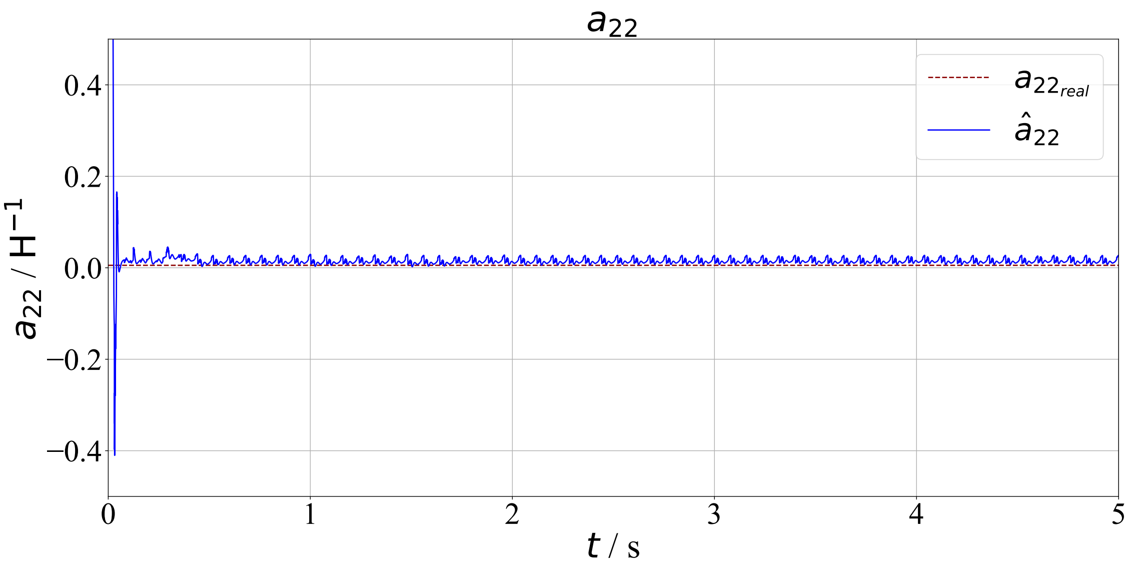


Fig. A.13: Estimated parameter $\hat{a}_{22}$ selecting a voltage vector for 200 time steps

A.2 Analysis of the different algorithms in the hyper-parameter optimization

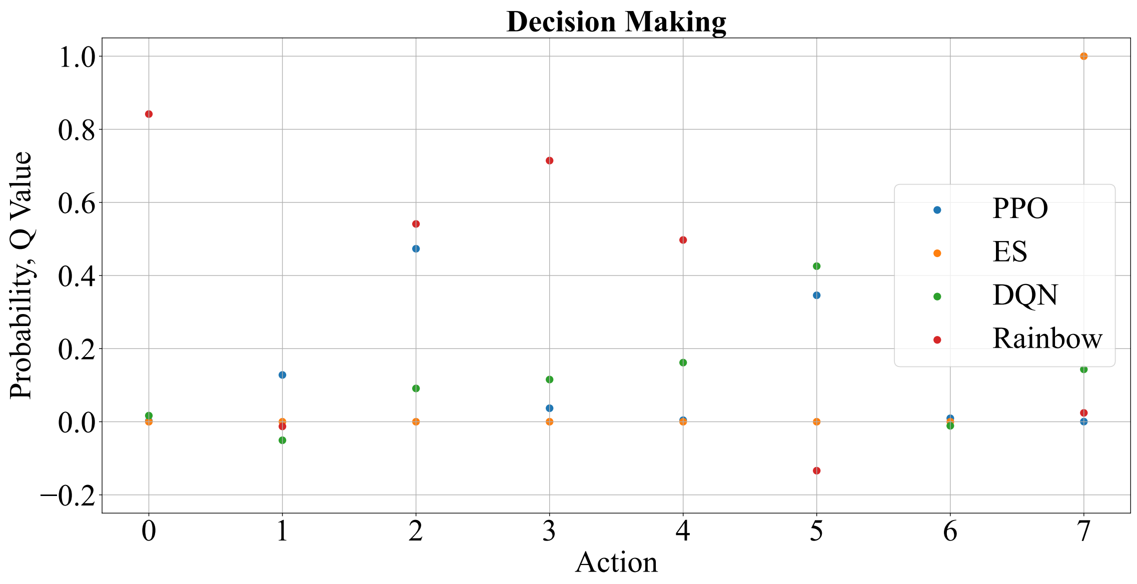


Fig. A.14: Estimated q-values and probabilities for the actions at operation point 1

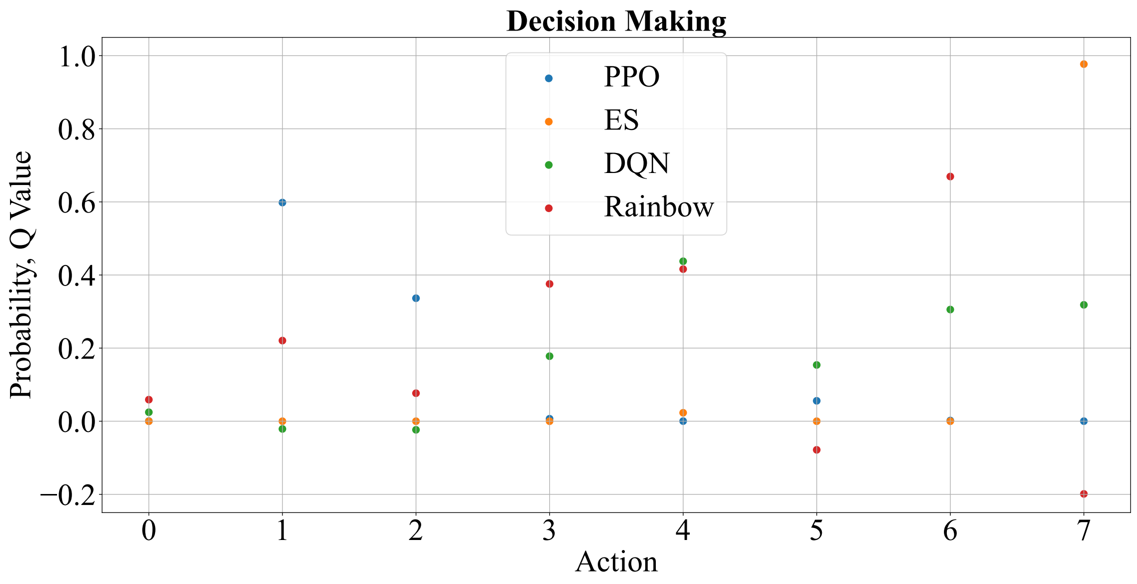


Fig. A.15: Estimated q-values and probabilities for the actions at operation point 2

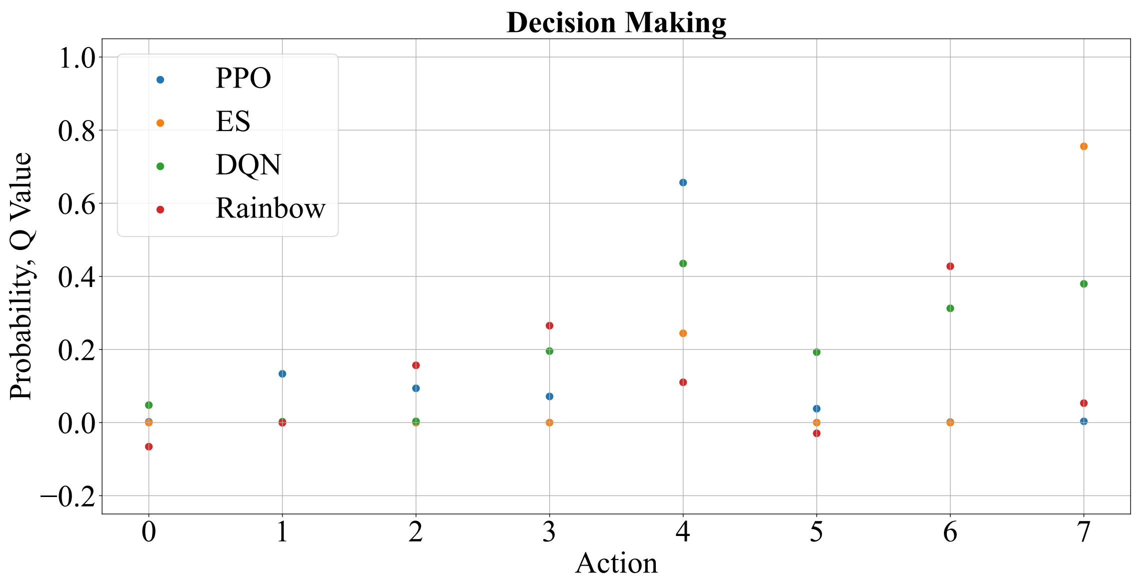


Fig. A.16: Estimated q-values and probabilities for the actions at operation point 3

A.3 Results of the hyperparameter optimization

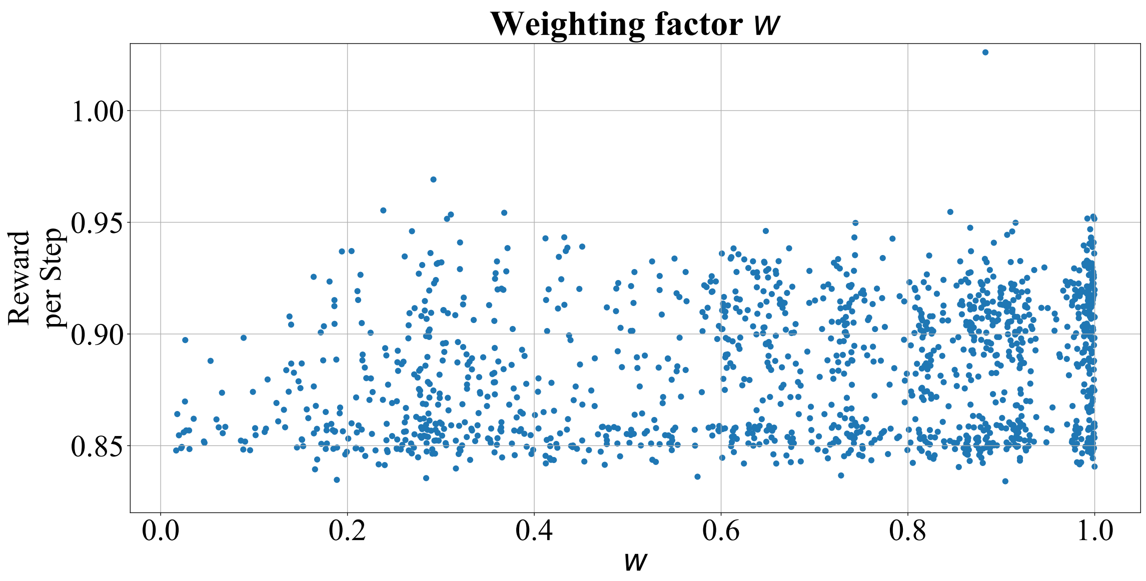


Fig. A.17: Results of the weighting factor w

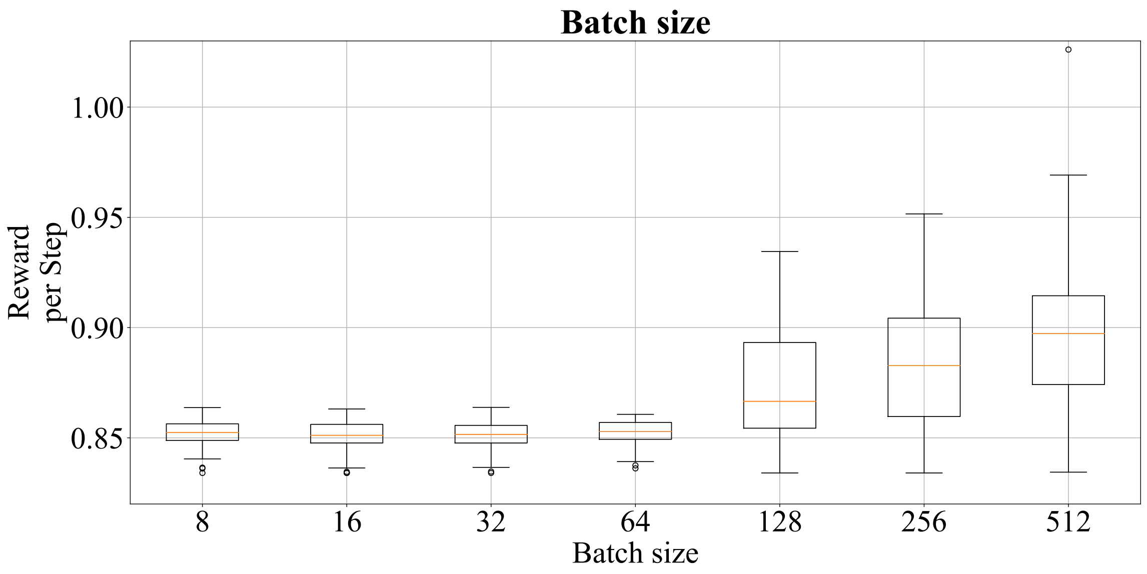


Fig. A.18: Results of the batch size

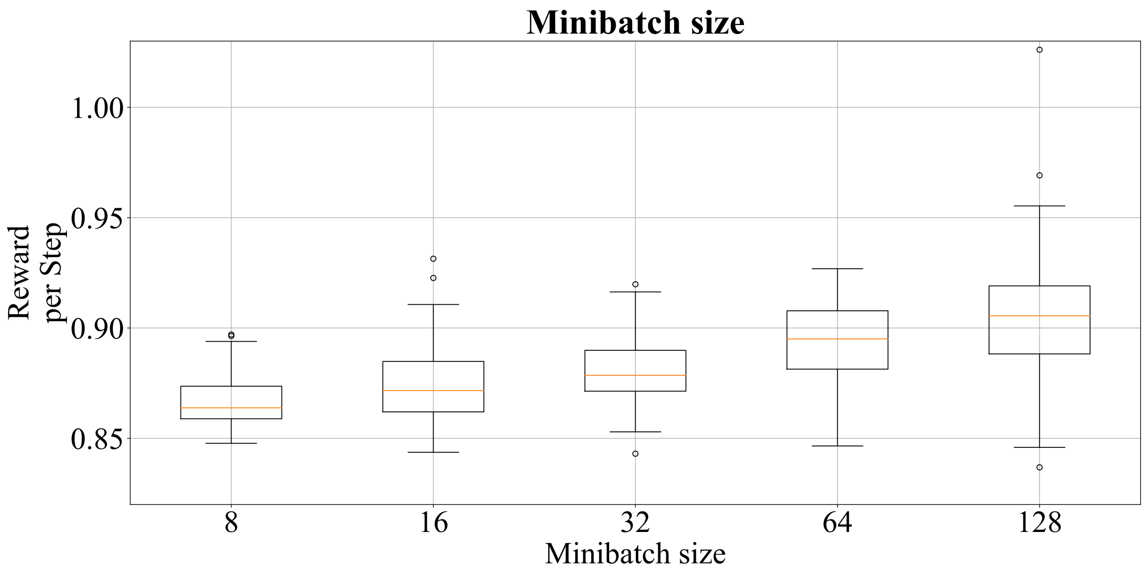


Fig. A.19: Results of the minibatch size

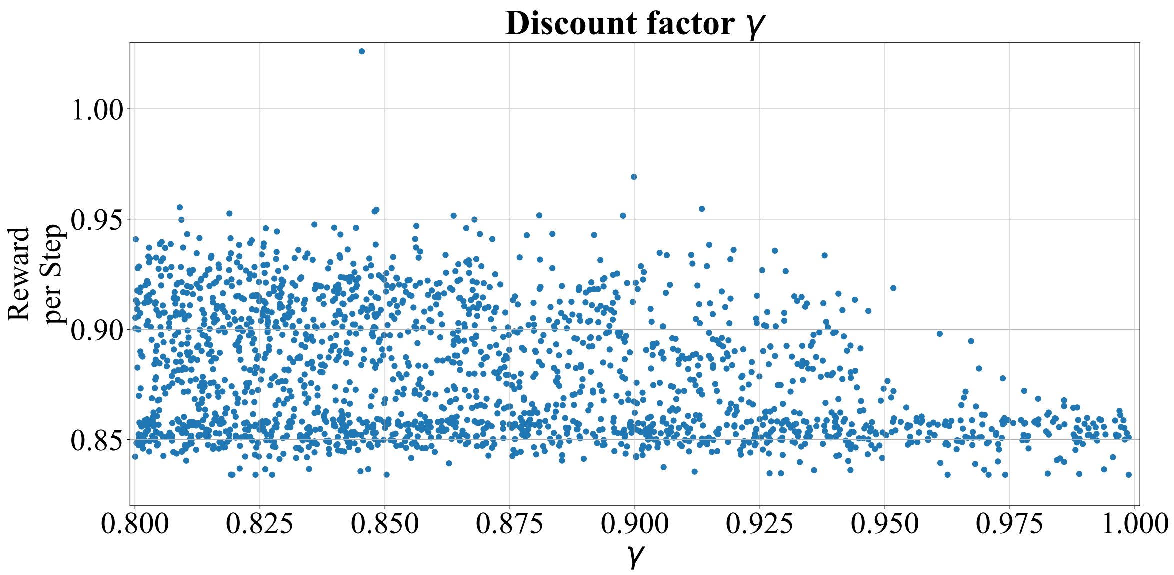


Fig. A.20: Results of the discount factor γ

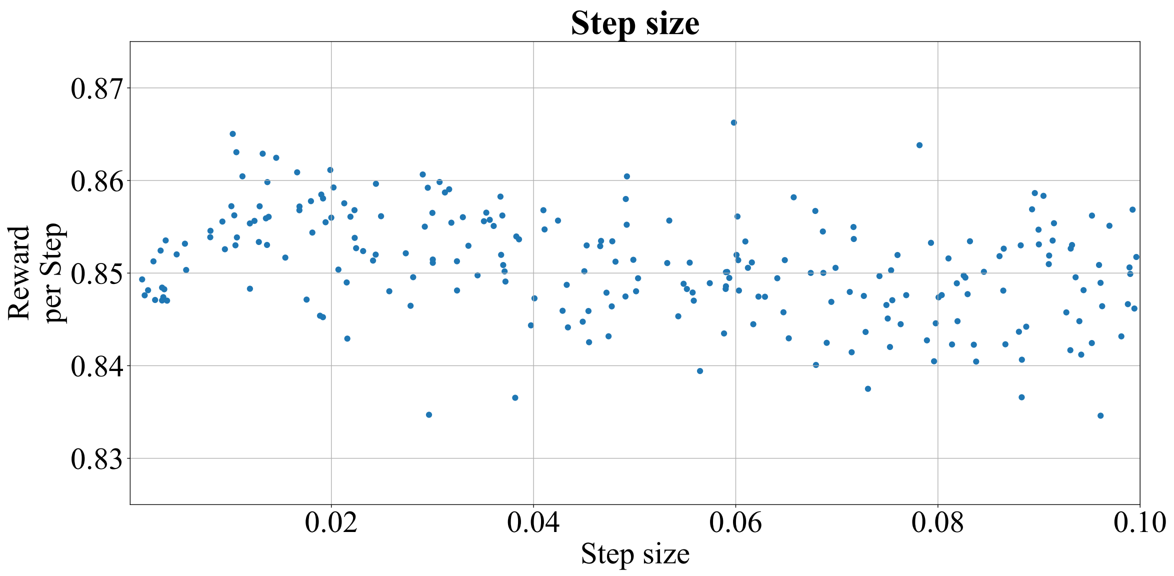


Fig. A.21: Results of the step size

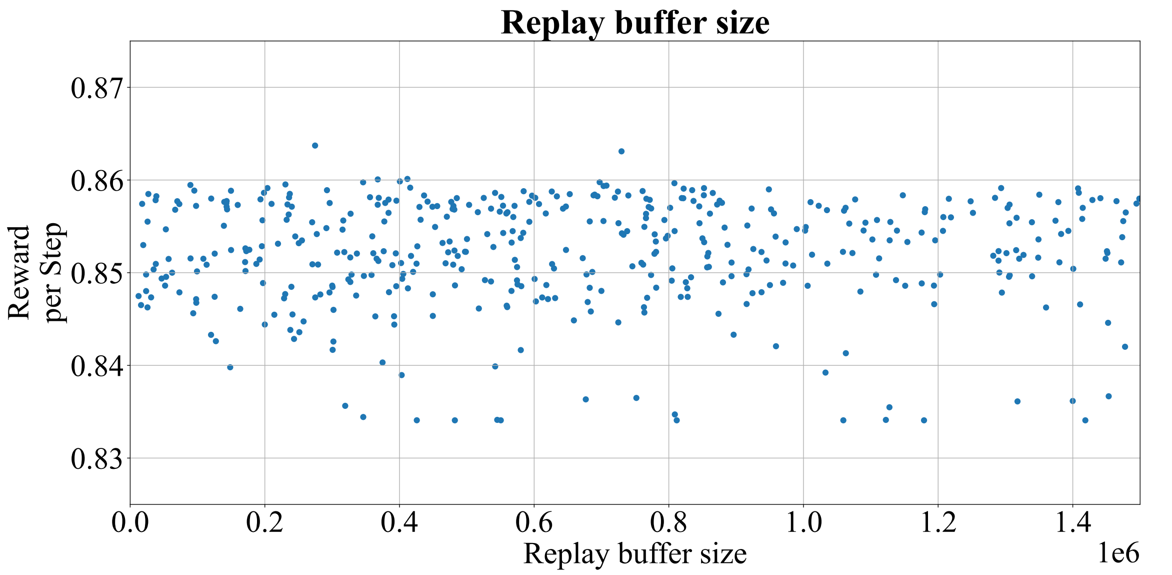


Fig. A.22: Results of the replay buffer size

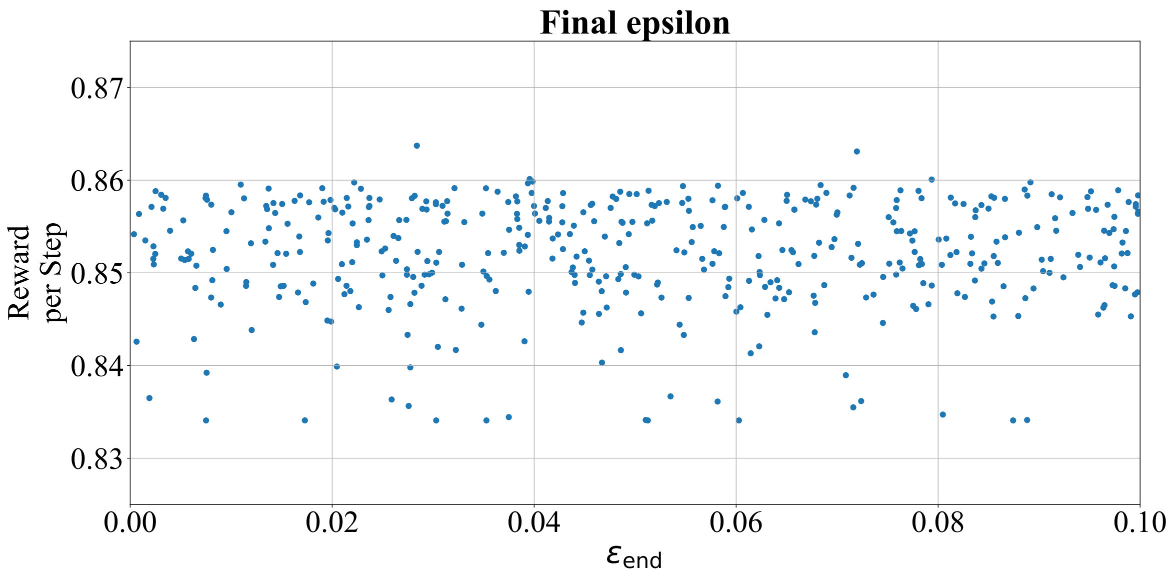


Fig. A.23: Results of the final epsilon ϵ_{end}

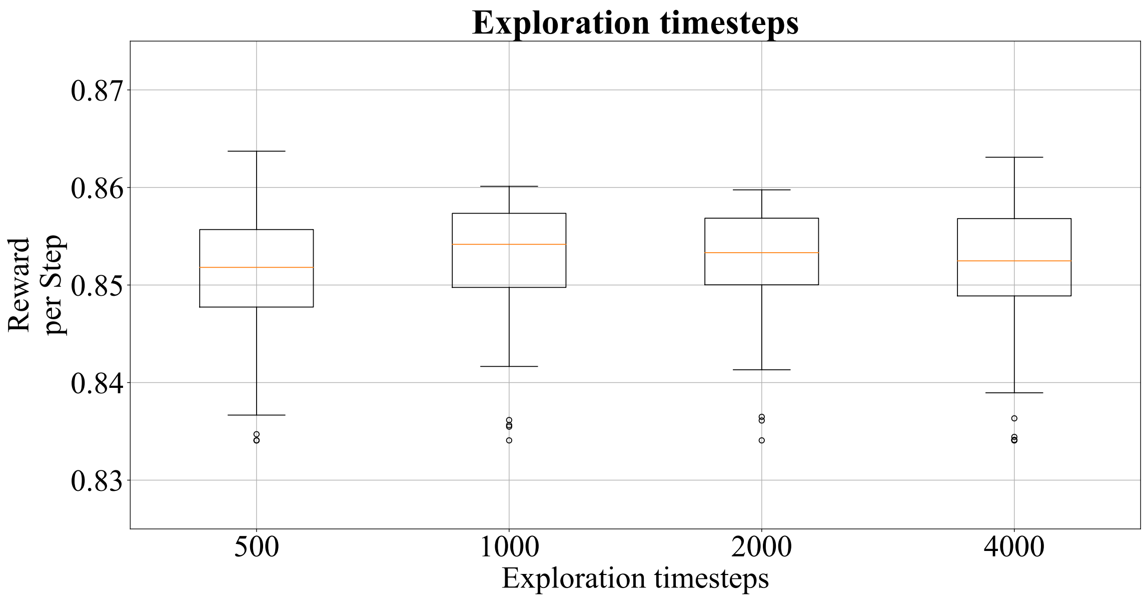


Fig. A.24: Results of the exploration timesteps

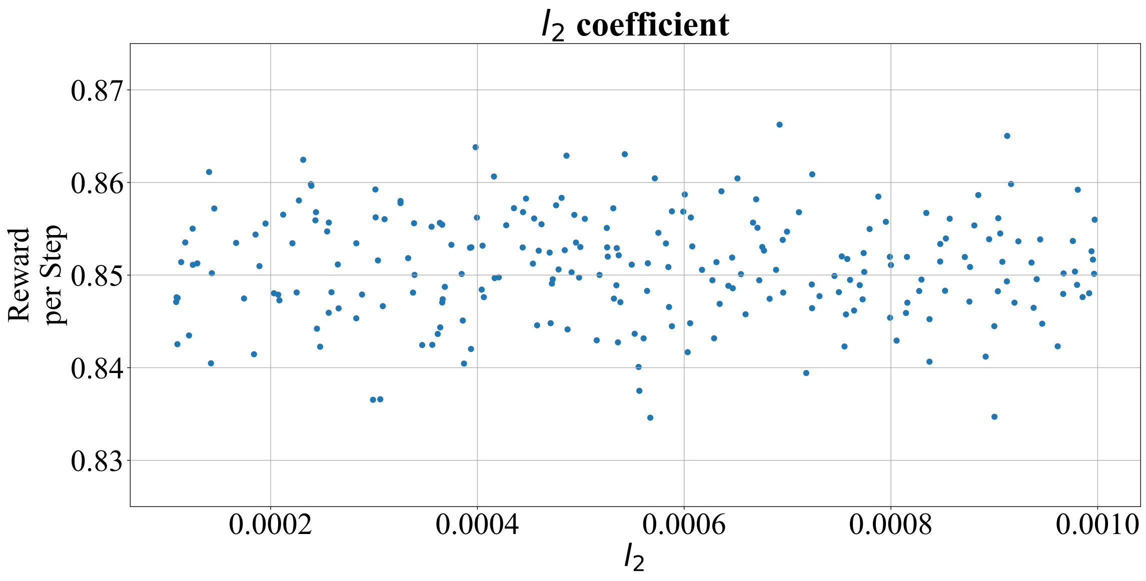


Fig. A.25: Results of the l_2 coefficient

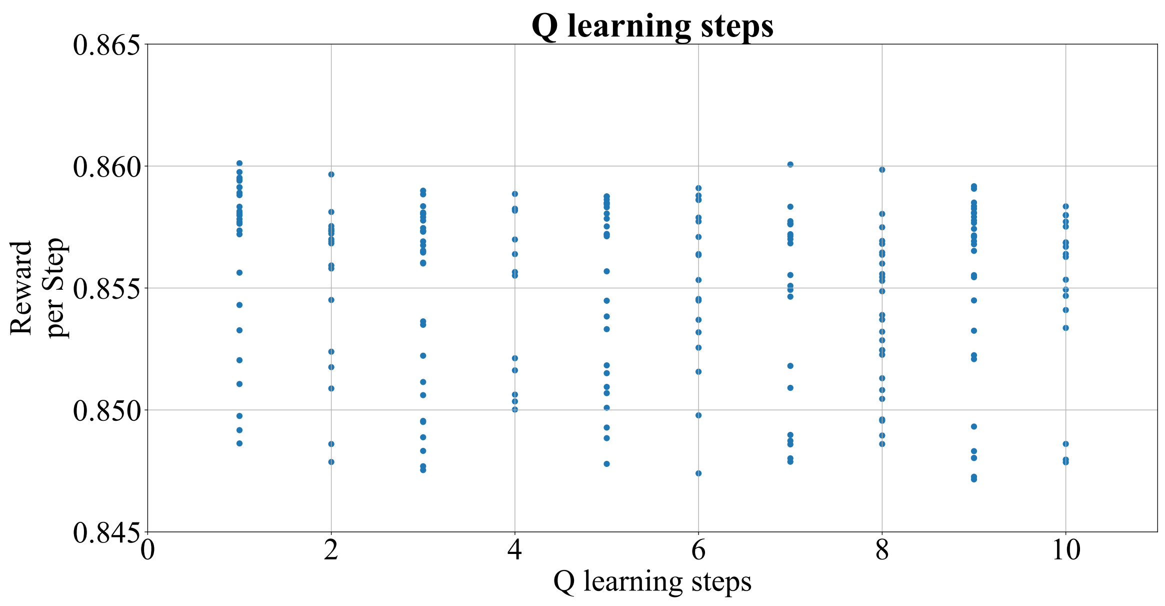


Fig. A.26: Results of the q-learning steps

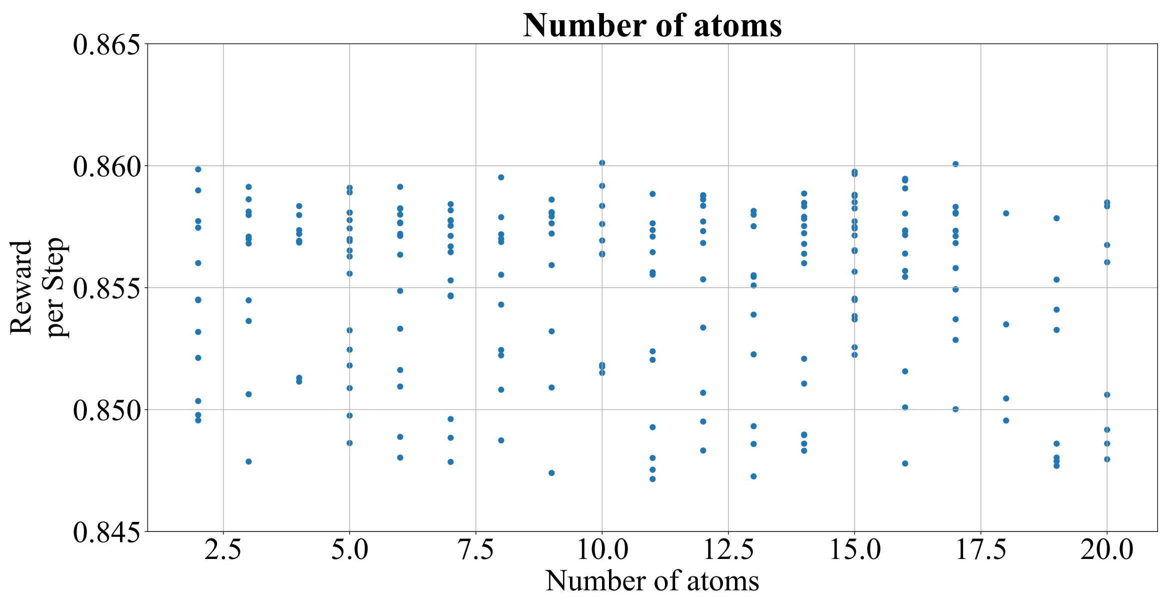


Fig. A.27: Results of the number of atoms

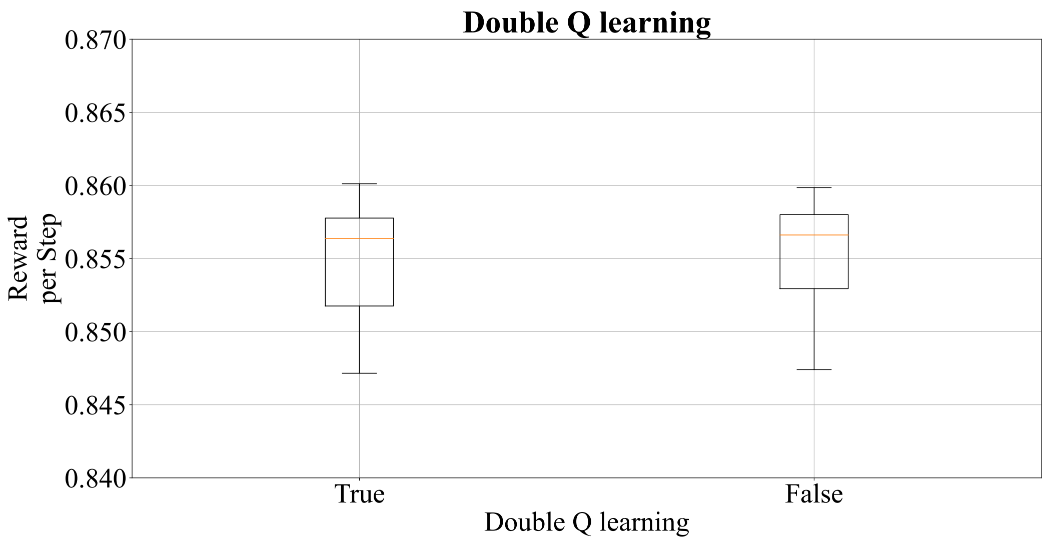


Fig. A.28: Results of the double q-learning

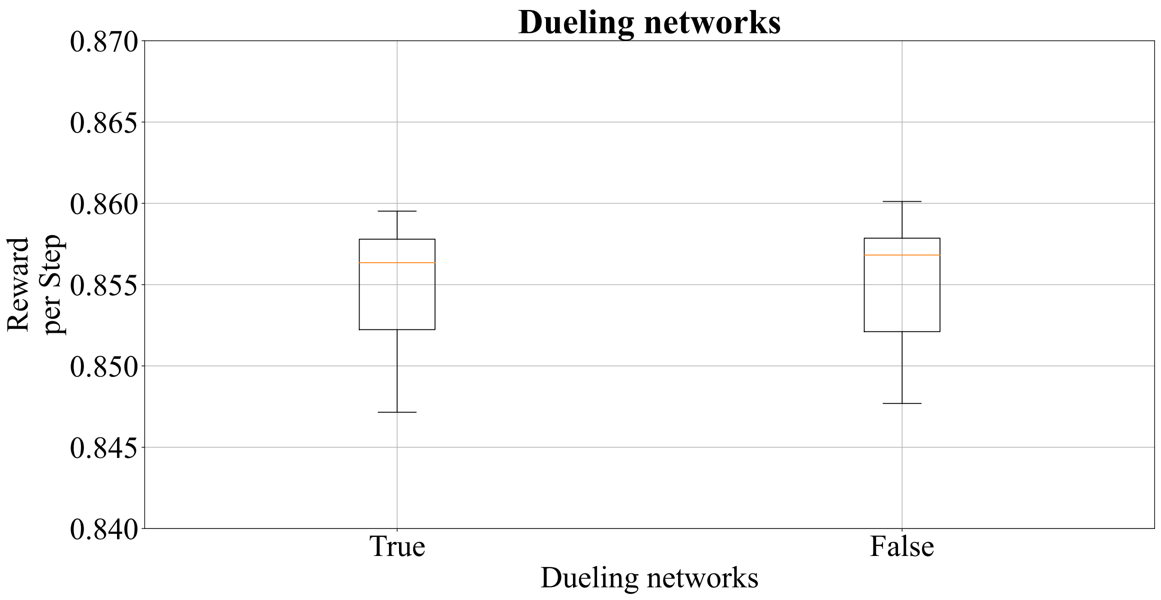


Fig. A.29: Results of the dueling networks

A.4 Statistical analysis of the hyperparameter optimization results

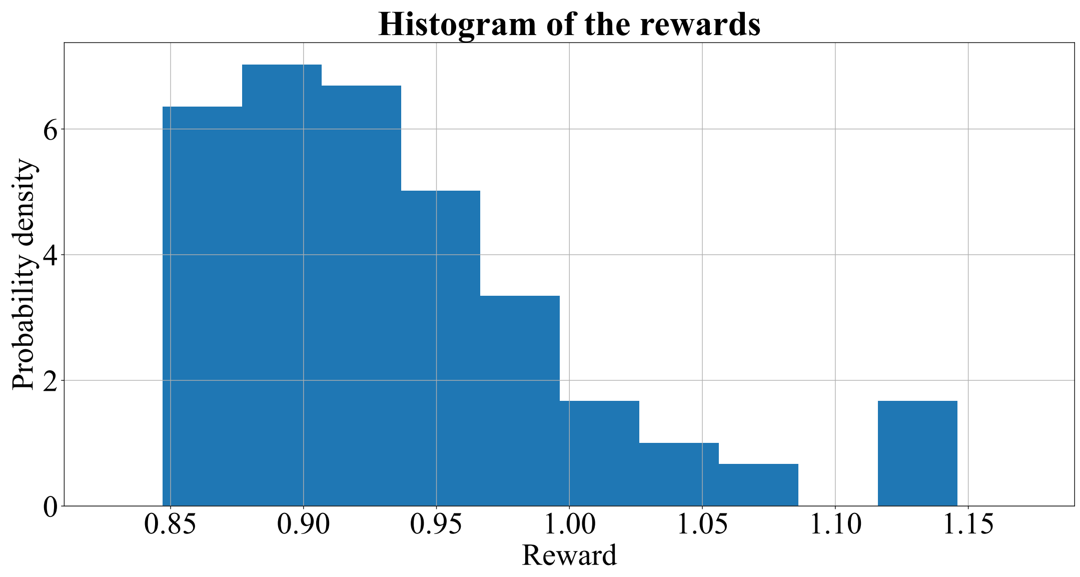


Fig. A.30: Histogram of the test results with equal references

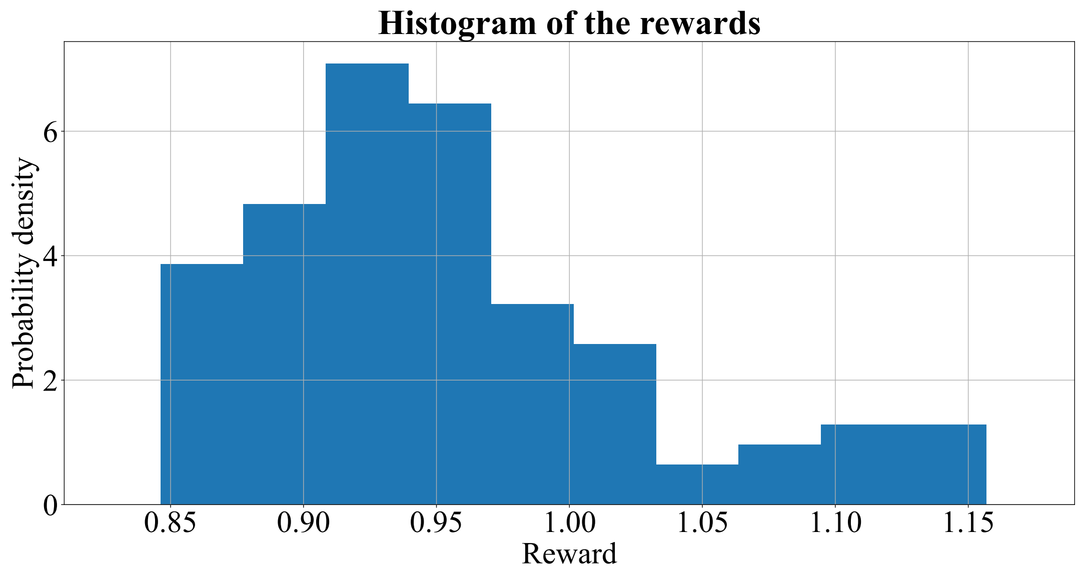


Fig. A.31: Histogram of the test results with identical initial network weights

A.5 Step response of the controllers

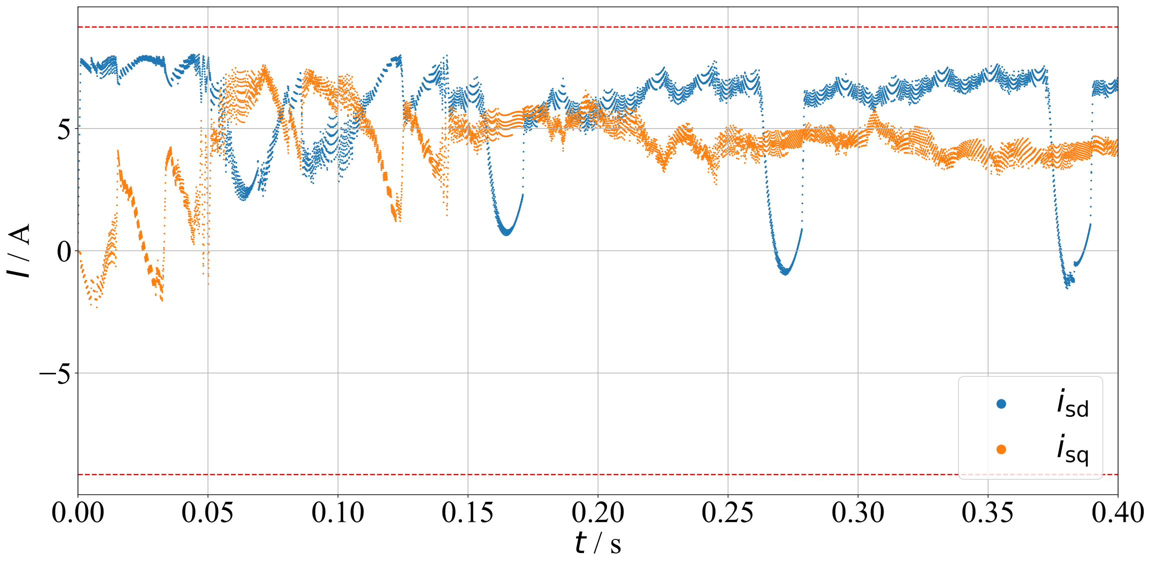


Fig. A.32: Currents of the step response of the RL controller

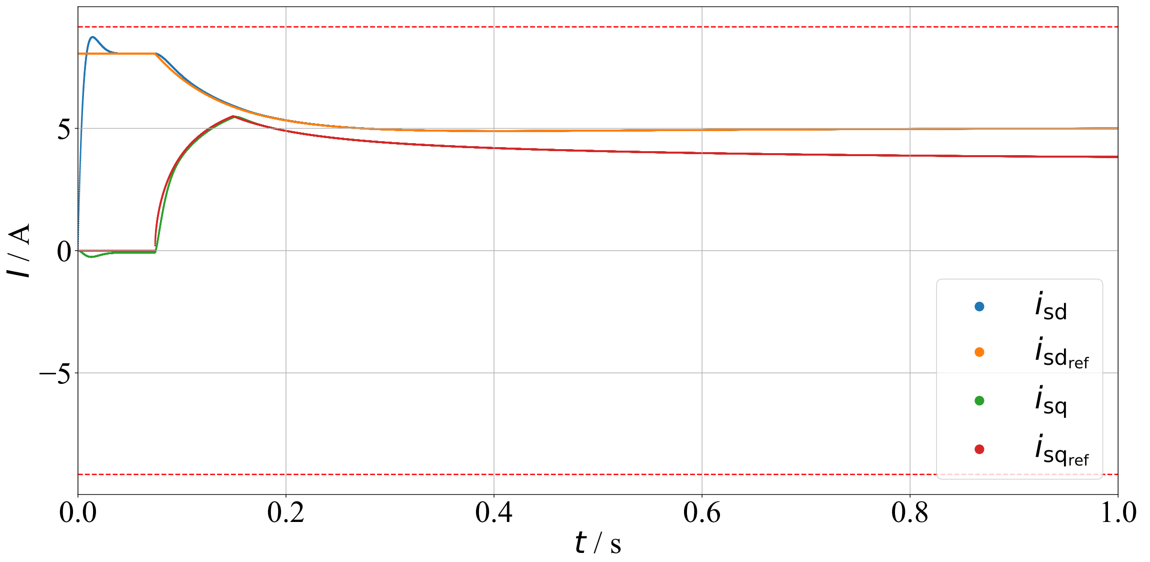


Fig. A.33: Currents of the step response of the FOC

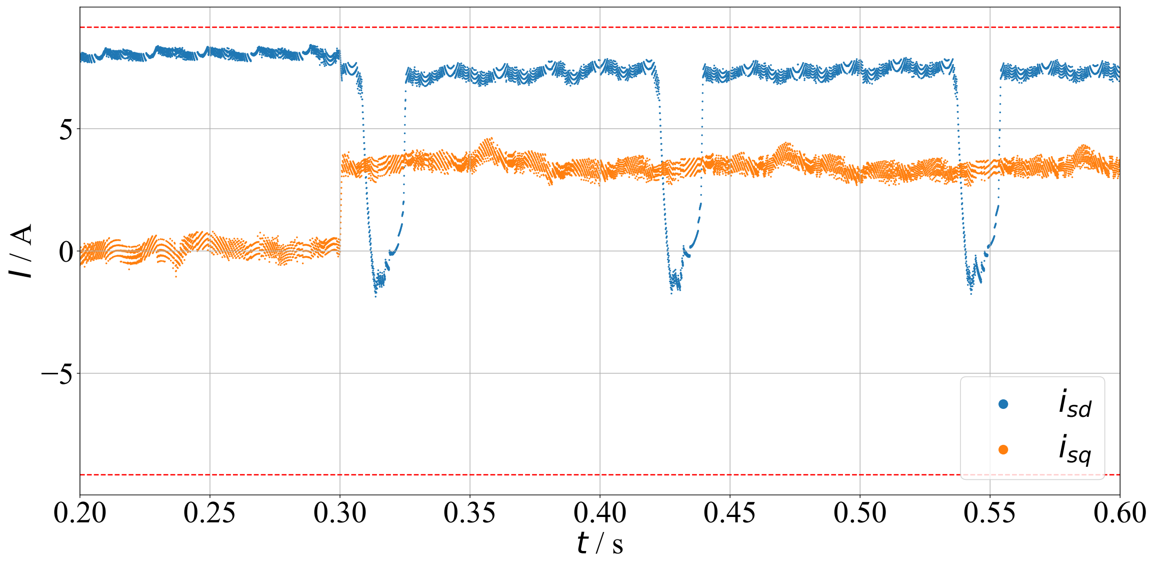


Fig. A.34: Currents of the step response after 0.3 s of the RL controller

Lists

List of Tables

List of Figures

Acronyms

ANN artificial neural network

DDPG Deep Deterministic Policy Gradient

DQN deep Q networks

ES Evolution Strategies

FOC field-oriented controller

GEM gym-electric-motor

IGBT Isolated Gate Bipolar Transistor

IM induction motor

MDP Markov decision process

MOSFET Metal Oxide Field Effect Transistor

MPC model predictive controller

OLS ordinary least squares

PC^2 Paderborn Center for Parallel Computing

PI controller proportional-integral controller

PMSM permanent magnet synchronous motor

POMDP partially observable Markov decision process

PPO Proximal Policy Optimization

ReLU Rectified Linear Unit

RL reinforcement learning

RLS recursive least squares

SCIM squirrel cage induction motor

SGD stochastic gradient descent

tanh tangens hyperbolicus

TPE Tree-structured Parzen Estimator

TRPO Trust Region Policy Optimization

Nomenclature

α	learning rate
η	efficiency
γ	discount factor
λ	Lagrange multiplier
θ	parameter vector
ξ	regressor vector
$\mathbf{Q}$	Park transformation
$\mathit{\Psi}_\mathrm{r}$	rotor flux
$\mathit{\Psi}_\mathrm{s}$	stator flux
ω_me	mechanical speed
ϕ	magnetic flux
π	policy
Ψ_r	rotor flux
σ	leakage coefficient
σ_x	measured standard deviation
τ	sampling time
τ_σ	leakage time constant
τ_r	rotor time constant
Θ	network weights
ε	angle of the rotor
φ	measured data vector
a_k	action at timestep k
c_eff	efficiency weight
c_lp	lowpass coefficient
e	residual vector
G_k	discounted return
i_dc	dc current
i_lim	current limit

i_{r}	rotor current
i_{s}	stator current
i_{total}	total current
L	loss function
$L_{\sigma\mathrm{r}}$	rotor-side stray inductance
$L_{\sigma\mathrm{s}}$	stator-side stray inductance
L_{m}	main inductance
L_{r}	rotor inductance
L_{s}	stator inductance
N	number of turns
n	rotor speed
n_{s}	synchronous speed
P	transition probability
p	pol pair number
P_{d}	dissipation power
P_{el}	electrical power
P_{in}	input power
P_{me}	mechanical power
P_{out}	output power
q	action-value function
r_{eff}	efficiency reward
R_{est}	reward function of the estimator
R_{r}	rotor resistance
R_{s}	stator resistance
R_k	reward
r_k	reward at timestep k
s	slip
s_k	state at timestep k
T	torque

T_{ss}	steady-state torque
u	input variable
u_{dc}	dc voltage
u_{i}	induced voltage
u_{r}	rotor voltage
u_{s}	stator voltage
v_{π}	state-value function
W_{magn}	magnetic energy
y	target value

References

[1] J. Böcker, "Anriebe für umweltfreundliche fahrzeuge lecture notes," 2019.

[2] J. Böcker, "Controlled three-phase drives lecture notes," p. 212, 2021. [Online]. Available: https://ei.uni-paderborn.de/fileadmin/elektrotechnik/fg/lea/Lehre/GDA/Dokumente/Geregelte_Drehstromantriebe_DE_EN1.pdf.

[3] U. Beckert, *Asynchronmotor lecture notes*, Aug. 2005. [Online]. Available: https://tu-freiberg.de/sites/default/files/media/institut-fuer-elektrotechnik-12774/UBeckert_Scr/asm_stat_verhalten.pdf.

[4] W. Peters, "Wirkungsgradoptimale regelung von permanenterregten synchronmotoren in automobilen traktionsanwendungen unter berücksichtigung der magnetischen sättigung," Ph.D. dissertation, University of Paderborn, Aug. 2014.

[5] T. Huber, "Experimentelle identifikation eines thermischenmodells zur überwachung kritischertemperaturen in hochausgenutztenpermanenterregten synchronmotoren fürautomobile traktionsanwendungen," Ph.D. dissertation, University of Paderborn, 2015.

[6] J. Böcker and S. Mathapati, "State of the art of induction motor control," in *2007 IEEE International Electric Machines Drives Conference*, vol. 2, 2007, pp. 1459–1464.

[7] 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.

[8] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, *Openai gym*, 2016. [Online]. Available: https://www.gymlibrary.ml.

[9] D. J. H. David F. Griffiths, *Numerical Methods for Ordinary Differential Equations.* Springer London Ltd, Nov. 2010, 271 pp.

[10] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

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

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” Dec. 2013.

[13] H. van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” Sep. 2015.

[14] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” Nov. 2015.

[15] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, “Dueling network architectures for deep reinforcement learning,” Nov. 2015.

[16] K. D. Asis, J. F. Hernandez-Garcia, G. Z. Holland, and R. S. Sutton, “Multi-step reinforcement learning: A unifying algorithm,” *(2018). In AAAI Conference on Artificial Intelligence.*, Mar. 2017.

[17] M. G. Bellemare, W. Dabney, and R. Munos, “A distributional perspective on reinforcement learning,” Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., 2017.

[18] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, “Rainbow: Combining improvements in deep reinforcement learning,” Oct. 2017.

[19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” Jul. 2017.

[20] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, “Trust region policy optimization,” Feb. 2015.

[21] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” *ICML 2016*, Feb. 2016.

[22] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, “Evolution strategies as a scalable alternative to reinforcement learning,” Mar. 2017.

[23] T. R. Team. “Ray rllib.” (2022), [Online]. Available: https://docs.ray.io/en/latest/rllib/index.html.

[24] 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,” Sep. 2015.

[25] K. Wang, J. Chiasson, M. Bodson, and L. Tolbert, “A nonlinear least-squares approach for identification of the induction motor parameters,” in *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, vol. 4, 2004, pp. 3856–3861.

[26] C. Picardi and N. Rogano, “Parameter identification of induction motor based on particle swarm optimization,” in *International Symposium on Power Electronics, Electrical Drives, Automation and Motion, 2006. SPEEDAM 2006.*, 2006, pp. 968–973.

[27] A. Brosch, S. Hanke, O. Wallscheid, and J. Böcker, “Data-driven recursive least squares estimation for model predictive current control of permanent magnet syn-

chronous motors," *IEEE Transactions on Power Electronics*, vol. 36, pp. 2179–2190, Jul. 2020.

[28] R. Isermann and M. Münchhof, *Identification of Dynamic Systems.* Springer Berlin Heidelberg, 2011.

[29] H. Krasowski, X. Wang, and M. Althoff, "Safe reinforcement learning for autonomous lane changing using set-based prediction," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020, pp. 1–7.

[30] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," 2019.

[31] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Eds., Curran Associates, Inc., 2011.

[32] M. Schenke and O. Wallscheid, "A deep q-learning direct torque controller for permanent magnet synchronous motors," *IEEE Open Journal of the Industrial Electronics Society*, vol. 2, pp. 388–400, 2021.

[33] H. Dong, Z. Ding, and S. Zhang, *Deep Reinforcement Learning.* Springer Nature Singapore, 2020.

[34] S. Jang and H.-I. Kim, "Supervised pre-training for improved stability in deep reinforcement learning," *ICT Express*, Jan. 2022.

[35] A. Traue, F. Book, B. Haucke-Korber, M. Schenke, and O. Wallscheid. "Gem-control," University of Paderborn, Department of power electronics and electrical drives. (2022), [Online]. Available: https://github.com/upb-lea/gem-control.

[36] G. Book, A. Traue, P. Balakrishna, A. Brosch, M. Schenke, S. Hanke, W. Kirchgassner, and O. Wallscheid, "Transferring online reinforcement learning for electric motor control from simulation to real-world experiments," *IEEE Open Journal of Power Electronics*, vol. 2, pp. 187–201, 2021.