IEEE Open Journal of
Power Electronics

Received 11 February 2021; revised 1 March 2021 and 3 March 2021; accepted 7 March 2021. Date of publication 12 March 2021; date of current version 1 April 2021. The review of this article was arranged by Associate Editor Yue Zhao.

Digital Object Identifier 10.1109/OJPEL.2021.3065877

Transferring Online Reinforcement Learning for Electric Motor Control From Simulation to Real-World Experiments

GERRIT BOOK, ARNE TRAUE, PRANEETH BALAKRISHNA, ANIAN BROSCH, MAXIMILIAN SCHENKE, SÖREN HANKE, WILHELM KIRCHGÄSSNER, AND OLIVER WALLSCHEID (Member, IEEE)

Department of Power Electronics and Electrical Drives, Paderborn University, 33098 Paderborn, Germany

CORRESPONDING AUTHOR: OLIVER WALLSCHEID (e-mail: wallscheid@lea.upb.de)

This article has supplementary downloadable material available at https://doi.org/10.1109/OJPEL.2021.3065877

ABSTRACT Reinforcement learning (RL) based methods are an upcoming approach for the control of power systems such as electric drives. These data-driven techniques do not need an explicit plant model like most common state-of-the-art approaches. Instead, the control policy is continuously improved solely based on measurement feedback pursuing optimal control performance through learning. While the general feasibility of RL-based drive control algorithms has already been proven in simulation, this work focuses on transferring the methodology to real-world experiments. In the case of electric motor control, a strict real-time requirement, safety constraints, system delays and the limitations of embedded hardware frameworks are hurdles to overcome. Hence, several modifications to the general RL training setup are introduced in order to enable RL in real-world electric drive control problems. In particular, a rapid control prototyping toolchain is introduced allowing fast and flexible testing of arbitrary RL algorithms. This simulation-to-experiment pipeline is considered an important intermediate step towards introducing RL in embedded control for power electronic systems. To highlight the potential of RL-based drive control, extensive experimental investigations addressing the current control of a permanent magnet synchronous motor utilizing a deep deterministic policy gradient algorithm have been conducted. Despite the early state of research in this domain, promising control performance could be achieved.

INDEX TERMS Edge computing, electric drive, machine learning, optimal control, power electronics, reinforcement learning.

I. INTRODUCTION

Optimal electric motor control is of prime interest for various applications (e.g., automation and automotive engineering) that depend on high-performance drive systems. State-of-the-art motor control methods like linear quadratic regulators (LQR) [1], [2], model predictive control (MPC) [3], [4] or closed-form tuning of proportional-integral control (PI-control) [5], [6] require an accurate drive model for their design. While the latter is relatively more robust than MPC or LQR approaches due to its integral feedback, high-performance PI control still requires an exact model representation [7]. On the contrary, severe deviations between the real drive system and the drive model can occur due to plenty of reasons such as production tolerances or operation-dependent system behavior changes (e.g., temperature, magnetic saturation or wear-and-tear influences). In some cases a model might be even completely unknown, e.g., when a new motor is connected to a power electronic converter (self-commissioning).

In contrast, model-free reinforcement learning (RL) techniques do not require a mathematical motor model at all. RL motor controllers are completely data-driven and learn an optimal control policy directly from the drive's response. Also, secondary and parasitic effects like iron saturation, iron losses, the skin effect, or influences by nonlinear inverter behavior can be learned and directly compensated by RL control without requiring domain expert knowledge in this field.

Moreover, many RL algorithms allow background planning [8], i.e., the control inference (evaluating a control policy function) is decoupled from the learning process (a policy update step). Compared to MPC as a planning-at-decision-time approach, this relaxes real-time requirements and allows more implementation flexibility since learning the control policy can be executed asynchronously to the control inference.

A. RELATED WORK

Recent publications on this topic have shown that RL approaches already reach standard control performance in simulation [9], [10]. In particular, [9] provides a basic proof of concept of the methodology in the motor control context while [10] contributes to the development of an open-source drive system simulation toolbox using the OpenAI Gym standards [11] to test and train RL agents[1]. Such a simulation-based training pipeline can be used to derive RL-based control in an offline fashion, i.e., based on (simplified) motor models. However, deploying an offline-learned RL agent on a real-world drive application leads to the same drawback of limited model accuracy as discussed with the state-of-the-art control approaches. Previous contributions have not investigated the online training of RL-based control using real-world motor drive feedback on a fully experimental basis.

The transfer of RL algorithms from simulation to reality causes several new challenges that have to be faced, as summarized in [12]. In the case of electric motor control, mostly real-time requirements, safety constraints, measurement noise and system delays are of interest. Although an offline, simulation-based pre-training can be utilized in order to speed up the online training on the real physical system [13], the initial control performance after the transfer is non-optimal if the simulation model is not accurately matching the real-world system behavior. As will be discussed in Sec. II, this model mismatch is a prominent problem in drive applications.

Popular RL examples as AlphaGo [14] or other game-related approaches (e.g., [15]) do not face any real-time requirement. In drive control, however, the typical turnaround time ranges from 10200 μs. Due to this real-time constraint, training carried out directly on the real-time hardware becomes infeasible. Hence, the control policy inference and the learning have to be decoupled and implemented on different time scales - a batched RL training [16] is necessary.

Safety constraints are another crucial point in motor control. For example, electric currents exceeding the limits of the drive might destroy it due to rapid overheating. RL algorithms do not consider constraints inherently. For instance, [17] and [18] face this issue by adding a safety layer correcting actions that violate constraints. Moreover, [19] forces the agent to learn the constraints during training by shaping the reward function, which penalizes policies exceeding the safety bounds.

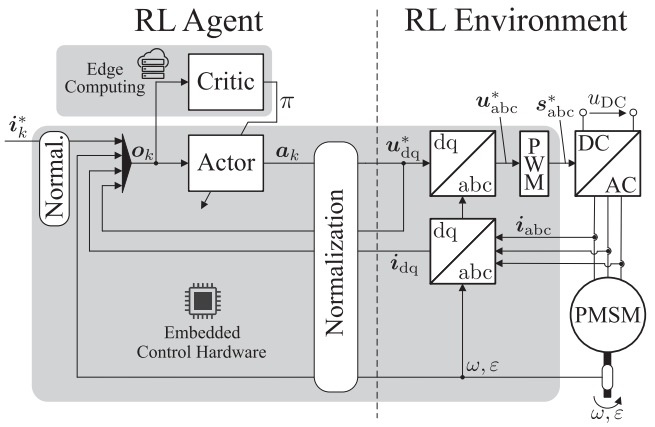

FIGURE 1. Simplified schematic of the overall control and drive system structure; note that all gray shaded parts are control-related while from the RL agent's perspective, both the coordinate transforms and PWM are part of the environment, i.e., they are pre/post-processing steps outside the RL agent's core software.

Furthermore, electric motor control systems contain multiple inherent forms of delays, e.g., calculation time of the controller hardware or the modulation scheme of the power electronic converter [20]. These can be modeled as a one-step delay in the application of the agent's actions, as described in Sec. IV-C. Such delays slow down the learning process of RL agents significantly. To tackle a τ_d-step delay before actions take effect, [21] appends the last τ_d applied actions to the observation of the RL agent. Alternatively, [22] uses recurrent neural network agents and a special reward allocation to properly assign reward to past actions.

In summary, the overwhelming majority of investigations in the field of RL are based on simulations without any interaction to real-world physical systems [12]. Addressing and solving issues when transferring RL-based control approaches to real-world applications, specifically for the field of electric drive systems, is therefore an important object of research in order to be able to transfer data-driven control techniques into industrial processes in the long run.

B. CONTRIBUTION

In this work, the transfer from simplified offline simulation-based training to online training and inference on real motor drive systems is presented. A Python-based rapid control prototyping toolchain[2] is developed that allows online training on a remote platform (edge computing) using measurements obtained from an embedded controller (cf. Fig. 1). Therefore, the training process is executed asynchronously in the background. This toolchain allows to rapidly test and validate various RL algorithms in the context of electric drive control without the necessity to implement the training process within the

[1]While the term *agent* was coined from the field of computer science, the term *controller* originates from the engineering sciences. We use both terms synonymously in the following.

[2]The full rapid control prototyping toolchain with an extended technical documentation is available as an attachment to this publication.

embedded software. Hence, only the control inference (policy evaluation) is required to be executed in real time on the embedded controller while the learning step (policy improvement) is decoupled. For demonstration purpose, a batched version of a deep deterministic policy gradient (DDPG) algorithm [23] is extended to learn the current control policy for a permanent magnet synchronous motor (PMSM) that is fed by a B6-bridge power electronic converter. Further innovations of this contribution handle the safety constraints and system delays in the case of RL motor control including extensions to the baseline DDPG algorithm [23]. In addition, the impact of offline-based pre-training using a drive model is compared to randomly initialized DDPG agents. The functionality of the presented architecture from Fig. 1 is successfully tested on a laboratory electric drive test bench. The data-driven DDPG-based controller is compared against state-of-the-art linear field-oriented control and model predictive control approaches. The presented rapid control prototyping toolchain is an important intermediate step in order to accelerate data-driven control research in power systems. Although an actor-critic-based RL approach is depicted in Fig. 1 the rapid control prototyping toolchain can be directly applied to value-based RL techniques such as (double) deep Q-networks [24], [25], too. In particular, it allows to seamlessly plug-in and test many potentially interesting RL algorithms on a Python basis avoiding cumbersome embedded software implementations of each and every algorithm. This is especially valuable because many contemporary RL algorithms are publicly available as open-source Python code (e.g. Stable-Baselines3 [26], TF-Agents [27] or Keras-RL [28]) and, thus, can be tested comparatively easily on a real, physical system.

C. PAPER STRUCTURE

First, in Sec. II a basic PMSM-based drive model is presented explaining the basic functioning of the system to be controlled for non-motor-experts. Moreover, the same model is utilized within an MPC and a linear feedback approach for comparison reasons in the later part of the paper. In Sec. III the fundamental background of RL in terms of the DDPG algorithm is summarized followed by extensions and modifications for batched online learning in Sec. IV. The utilized laboratory test platform is presented in Sec. V followed by the experimental results in Sec. VI. Finally, a conclusion on the major findings and an outlook to future work in the field are given in Sec. VII.

II. DRIVE SYSTEM MODEL

The drive system (cf. Fig. 1), which is used in this work, consists of a B6-bridge voltage source power electronic converter and a PMSM. In the following, these two main components of the electric drive are briefly modeled.

A. PERMANENT MAGNET SYNCHRONOUS MOTOR

The mathematical model of the three-phase PMSM can be simplified using the rotor-fixed dq-coordinates [29]. Therefore, two transformations are necessary. First, the three phases x_a, x_b and x_c are transformed to stator-fixed x_α, x_β and x_0 components with

$$\begin{bmatrix} x_\alpha \\ x_\beta \\ x_0 \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ 0 & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix} \begin{bmatrix} x_\mathrm{a} \\ x_\mathrm{b} \\ x_\mathrm{c} \end{bmatrix}. \tag{1}$$

Second, a rotational transformation to rotor-fixed variables x_d and x_q is performed with the electrical rotor angle ε

$$\boldsymbol{x}_\mathrm{dq} = \begin{bmatrix} x_\mathrm{d} \\ x_\mathrm{q} \end{bmatrix} = \begin{bmatrix} \cos(\varepsilon) & \sin(\varepsilon) \\ -\sin(\varepsilon) & \cos(\varepsilon) \end{bmatrix} \begin{bmatrix} x_\alpha \\ x_\beta \end{bmatrix}. \tag{2}$$

(1) and (2) apply to current, magnetic flux linkage and voltage [30]. The resulting flux linkage ordinary differential equation (ODE) in dq-coordinates is

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\psi}_\mathrm{dq} = \boldsymbol{u}_\mathrm{dq} - R_\mathrm{s}\boldsymbol{i}_\mathrm{dq} - \omega\boldsymbol{\Phi}\boldsymbol{\psi}_\mathrm{dq} \tag{3}$$

with

$$\boldsymbol{\Phi} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}. \tag{4}$$

Here, $\boldsymbol{\psi}_\mathrm{dq}$ is the stator flux linkage, $\boldsymbol{u}_\mathrm{dq}$ is the stator voltage, R_s is the stator resistance, $\boldsymbol{i}_\mathrm{dq}$ is the stator current and ω is the electrical angular fundamental frequency.[3] The resulting electromagnetic torque T produced by the motor is

$$T = \frac{3}{2}p\boldsymbol{i}_\mathrm{dq}^T\boldsymbol{\Phi}\boldsymbol{\psi}_\mathrm{dq} = \frac{3}{2}p\left(\psi_\mathrm{d}i_\mathrm{q} - \psi_\mathrm{q}i_\mathrm{d}\right) \tag{5}$$

with the pole pair number p. The ODE for the mechanical angular velocity ω_me is defined by

$$\frac{\mathrm{d}\omega_\mathrm{me}}{\mathrm{d}t} = \frac{T - T_\mathrm{l}}{J} \tag{6}$$

with the load torque T_l and the moment of inertia J. The relationship of mechanical (ε_me, ω_me) and electrical (ε, ω) quantities is given by

$$\omega = p\omega_\mathrm{me} \text{ and } \varepsilon = p\varepsilon_\mathrm{me}. \tag{7}$$

To fully describe the motor behavior a relationship between the magnetic flux linkage $\boldsymbol{\psi}_\mathrm{dq}$ and the stator current $\boldsymbol{i}_\mathrm{dq}$ is required. For many drives a simple linear equation

$$\boldsymbol{\psi}_\mathrm{dq} = \boldsymbol{\psi}_\mathrm{PM} + \boldsymbol{L}_\mathrm{dq}\boldsymbol{i}_\mathrm{dq} \tag{8}$$

with

$$\boldsymbol{\psi}_\mathrm{PM} = \begin{bmatrix} \psi_\mathrm{PM} \\ 0 \end{bmatrix}, \quad \boldsymbol{L}_\mathrm{dq} = \begin{bmatrix} L_\mathrm{d} & 0 \\ 0 & L_\mathrm{q} \end{bmatrix} \tag{9}$$

utilizing absolute inductance values ($L_\mathrm{d}, L_\mathrm{q}$) and a constant permanent magnet flux linkage ψ_PM can be often used. Consequently, the electrical ODE simplifies to

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{i}_\mathrm{dq} = \boldsymbol{L}_\mathrm{dq}^{-1}\left[\boldsymbol{u}_\mathrm{dq} - R_\mathrm{s}\boldsymbol{i}_\mathrm{dq} - \omega\boldsymbol{\Phi}\left(\boldsymbol{\psi}_\mathrm{PM} + \boldsymbol{L}_\mathrm{dq}\boldsymbol{i}_\mathrm{dq}\right)\right]. \tag{10}$$

In this case, a classical equivalent circuit diagram can be drawn as depicted in Fig. 2. However, for highly utilized

[3]Bold symbols denote multidimensional quantities, e.g., vectors or matrices.

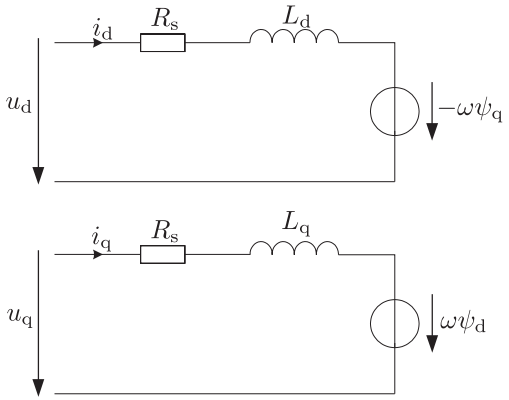

FIGURE 2. Equivalent circuit diagram of a PMSM in dq-coordinates with simplified linear magnetic behavior.

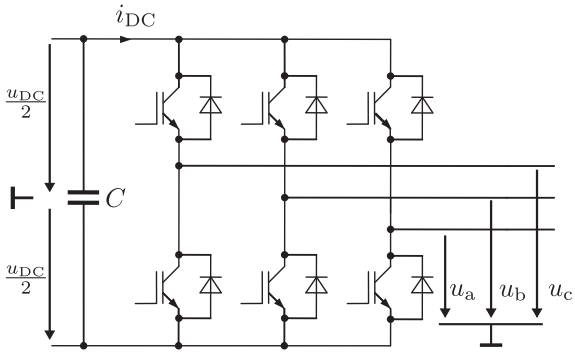

FIGURE 3. Three-phase two-level inverter.

PMSM drives, especially in the automotive or aerospace domain, significant (cross-)saturation within the magnetic circuit is occurring [5]. As a consequence, the linear magnetic model (8) is not valid and has to be replaced by a nonlinear relation

$$\boldsymbol{\psi}_\mathrm{dq} = \boldsymbol{f}_\mathrm{dq}\left(\boldsymbol{i}_\mathrm{dq}\right). \tag{11}$$

Instead of absolute inductances as in (9), operating point-dependent differential inductances have to be taken into account:

$$\boldsymbol{L}_\mathrm{dq} = \begin{bmatrix} \frac{\partial\psi_\mathrm{d}(\boldsymbol{i}_\mathrm{dq})}{\partial i_\mathrm{d}} & \frac{\partial\psi_\mathrm{d}(\boldsymbol{i}_\mathrm{dq})}{\partial i_\mathrm{q}} \\ \frac{\partial\psi_\mathrm{q}(\boldsymbol{i}_\mathrm{dq})}{\partial i_\mathrm{d}} & \frac{\partial\psi_\mathrm{q}(\boldsymbol{i}_\mathrm{dq})}{\partial i_\mathrm{q}} \end{bmatrix} = \begin{bmatrix} L_\mathrm{dd}(\boldsymbol{i}_\mathrm{dq}) & L_\mathrm{dq}(\boldsymbol{i}_\mathrm{dq}) \\ L_\mathrm{qd}(\boldsymbol{i}_\mathrm{dq}) & L_\mathrm{qq}(\boldsymbol{i}_\mathrm{dq}) \end{bmatrix}. \tag{12}$$

The resulting nonlinear current ODE is then

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{i}_\mathrm{dq} = \boldsymbol{L}_\mathrm{dq}^{-1}\left(\boldsymbol{u}_\mathrm{dq} - R_\mathrm{s}\boldsymbol{i}_\mathrm{dq} - \omega\boldsymbol{\Phi}\boldsymbol{\psi}_\mathrm{dq}(\boldsymbol{i}_\mathrm{dq})\right). \tag{13}$$

Besides the magnetic (cross-)saturation, other parasitic effects like temperature influences [31], rotor angle-dependent flux harmonics [32], tolerances in mass production [33], iron losses [34], and the skin- and proximity effect [35] are directly influencing the motor behavior making it a general nonlinear and uncertain control plant. Since those effects are hard to model in an analytical closed-form, we will not go into more details here, but refer to the further technical literature [36].

B. POWER ELECTRONIC CONVERTER

The circuit diagram of the three-phase two-level converter is given in Fig. 3. A pulse width modulation (PWM) scheme is used for the generation of the switching commands of the transistors [20]. Hence, the control actions are the reference stator voltages $\boldsymbol{a} = \boldsymbol{u}_\mathrm{dq}^* = \begin{bmatrix} u_d^* & u_q^* \end{bmatrix}^\mathrm{T}$, which are continuous variables. The action space is limited by the DC-link voltage u_DC and by six box constraints in stator-fixed coordinates (so-called voltage hexagon) [20], which translates to a rotor angle-dependent limitation in dq-coordinates:

$$\underline{\boldsymbol{g}}(u_\mathrm{DC}, \varepsilon) \leq \begin{bmatrix} u_d & u_q \end{bmatrix}^\mathrm{T} \leq \overline{\boldsymbol{g}}(u_\mathrm{DC}, \varepsilon). \tag{14}$$

Due to the discrete-time control implementation and the inverter modulation scheme in combination with regular sampling of the relevant states, there is a delay between the applied control action and the applied voltage at the motor terminals [5]. This digital control delay is discussed more deeply and compensated from an RL point of view in the upcoming Sec. IV. Besides that, the inverter itself is also a nonlinear actuator since the reference voltage $\boldsymbol{u}_\mathrm{dq}^*$ from the control is not perfectly transferred to the applied voltage $\boldsymbol{u}_\mathrm{dq}$ at the motor terminals due to resistive voltage drops, the anti-short-circuit interlocking times as well as non-ideal switching transitions [37], [38]. Hence, the drive system's two main components show nonlinear behavior, which is caused by a variety of parasitic influences of different physical root causes and, therefore, are hard to fully model during the control design phase. This motivates the usage of data-driven control techniques from the field of RL since they do not require a system model but can directly learn from the interaction with the physical system.

III. DEEP DETERMINISTIC POLICY GRADIENT

First, it may be noted that both the state and action space are continuous within the given drive control problem. For this kind of problem space, the DDPG algorithm is a well known solution candidate from the RL domain and will be utilized for demonstrating the online learning rapid control prototyping toolchain under real-world experimental conditions. Therefore, only a short introduction to the RL in general and the DDPG in particular is given, while more details can be found in [8], [23]. The standard pseudocode of the DDPG algorithm is presented in Alg. 1. Additionally, modifications to the standard DDPG approach that boost the learning process for real-time motor control are presented in Sec. IV.

The classic RL setup consists of an agent and an environment. The environment can be modeled as a Markov decision process (MDP) (cf. [8]). At each discrete time step k, the agent receives an observation $\boldsymbol{o}_k = f(\boldsymbol{s}_k)$ from the environment, i.e., measurements, reference signals and further quantities in a feature engineering sense derived from the environment state $\boldsymbol{s}_k$. The agent calculates an action $\boldsymbol{a}_k$ (i.e, the actuating variable) with its internal policy function $\boldsymbol{a}_k = \boldsymbol{\pi}(\boldsymbol{o}_k)$, and receives

a reward r_{k+1} for applying it to the environment. The reward indicates how well the agent is performing, and it is thus comparable to the MPC framework's cost function, albeit the latter is minimized rather than maximized. From a control engineering perspective, the reward signal r_k is an important degree of freedom which requires proper design to ensure that the RL agent is learning a suitable control policy. The agent's goal is to find an optimal policy $\boldsymbol{a}_k = \boldsymbol{\pi}^*(\boldsymbol{o}_k)$ which maps each observation to the action that maximizes the return g_k, which is the expected discounted cumulative reward over time:

$$g_k = \sum_{i=k}^{\infty} \gamma^{i-k} \mathbb{E}_\pi[R_i | \boldsymbol{O}_i = \boldsymbol{o}, \boldsymbol{A}_i = \boldsymbol{a}], \tag{15}$$

with the discount factor $\gamma \in [0, 1[$. Above, $\mathbb{E}[\cdot]$ is the expectation of the reward R_i (capital symbols denote random variables in Sec. III) given the random observations $\boldsymbol{O}_i$ and actions $\boldsymbol{A}_i$ with realizations $\boldsymbol{o}$ and $\boldsymbol{a}$. This relation can be rewritten in the form of the Bellman equation [8]:

$$q(\boldsymbol{o}_k, \boldsymbol{a}_k) = \mathbb{E}_\pi[\underbrace{R_k + \gamma \underbrace{q(\boldsymbol{o}_{k+1}, \boldsymbol{a}_{k+1})}_{g_{k+1}}}_{g_k}], \tag{16}$$

wherein q is the action value function that performs a mapping from the momentary observation $\boldsymbol{o}_k$ and action $\boldsymbol{a}_k$ to the expected return. This equation is of fundamental importance for many RL methods, as it condenses the problem of finding an optimal sequence of actions to the problem of finding the single optimal action at each time step.

The DDPG algorithm is an efficient RL algorithm for continuous observation and action spaces that separates action selection and policy evaluation [23]. Action selection is performed by the actor $\boldsymbol{\pi}_{\boldsymbol{\xi}} : \boldsymbol{o}_k \to \boldsymbol{a}_k$, which is a policy function approximator with parameters $\boldsymbol{\xi}$. Policy evaluation is carried out with use of the critic $q_{\boldsymbol{\zeta}} : (\boldsymbol{o}_k, \boldsymbol{a}_k) \to g_k$, which is an action value function approximator with parameters $\boldsymbol{\zeta}$. Both function approximators are usually artificial neural networks (ANN) whose specific topologies (general type of ANN, number of neurons per layer and number of layers, learning rate, etc.) represent an important subset of the agent's hyperparameters. The critic learns to predict the return for observation-action pairs $(\boldsymbol{o}_k, \boldsymbol{a}_k)$ [23]. It can be trained by minimizing the cost function J_c for given transition experiences $\boldsymbol{e}_k = (\boldsymbol{o}_k, \boldsymbol{a}_k, r_k, \boldsymbol{o}_{k+1}, d_{k+1})$:

$$\begin{aligned} &\min_{\boldsymbol{\zeta}} J_\mathrm{c} \\ &\text{s.t.} \quad J_\mathrm{c} = \frac{1}{|\mathcal{B}|} \sum_{\boldsymbol{e}_k \in \mathcal{B}} \Big(q_{\boldsymbol{\zeta}}(\boldsymbol{o}_k, \boldsymbol{a}_k) \\ &- \underbrace{\Big(r_k + \gamma (1 - d_{k+1}) q_{\boldsymbol{\zeta}_\mathrm{target}}\big(\boldsymbol{o}_{k+1}, \boldsymbol{\pi}_{\boldsymbol{\xi}_\mathrm{target}}(\boldsymbol{o}_{k+1})\big)_{\hat{g}_k} \Big)^2}_{} \end{aligned} \tag{17}$$

where in $\mathcal{B}$ is a batch of collected experiences and d_k is a done signal, which is equal to one ($d_k = 1$) if the observation $\boldsymbol{o}_k$ belongs to a terminal state and zero ($d_k = 0$) elsewise. The structure of the critic's cost function (17) indicates that the critic $q_{\boldsymbol{\zeta}}$ is trained by supervised learning, under the assumption that the estimated return $\hat{g}_k$ is an accurate approximation. Training the critic is based on the concept of bootstrapping, meaning that the critic $q_{\boldsymbol{\zeta}}$ is improved towards $\hat{g}_k$ by using an estimation of the upcoming action value, as proposed by (16). As this method can lead to unwanted parameter oscillations, it has been proven to be good practice to dampen the parameter updates within the critic that is used to calculate $\hat{g}_k$. This introduces the concept of the target critic $q_{\boldsymbol{\zeta}_\mathrm{target}}$, whose parameters $\boldsymbol{\zeta}_\mathrm{target}$ track the actual critic parameters $\boldsymbol{\zeta}$ with low-pass filter characteristic

$$\boldsymbol{\zeta}_\mathrm{target} \leftarrow (1 - \rho)\boldsymbol{\zeta}_\mathrm{target} + \rho \boldsymbol{\zeta}, \tag{18}$$

with damping parameter $\rho \in]0, 1]$ [15] [23].

When satisfactory action value estimation can be assumed, this knowledge can be used to optimize the actor $\boldsymbol{\pi}_{\boldsymbol{\xi}}$. The task of the actor is to simply choose an action such that $q_{\boldsymbol{\zeta}}(\boldsymbol{o}_k, \boldsymbol{\pi}_{\boldsymbol{\xi}}(\boldsymbol{o}_k))$ is maximized. This leads to a very straightforward actor cost function J_a:

$$\begin{aligned} &\min_{\boldsymbol{\xi}} J_\mathrm{a} \\ &\text{s.t.} \quad J_\mathrm{a} = -\frac{1}{|\mathcal{B}|} \sum_{\boldsymbol{o}_k \in \mathcal{B}} q_{\boldsymbol{\zeta}}(\boldsymbol{o}_k, \boldsymbol{\pi}_{\boldsymbol{\xi}}(\boldsymbol{o}_k)). \end{aligned} \tag{19}$$

Also for the actor it is often helpful to make use of target parameters $\boldsymbol{\xi}_\mathrm{target}$. Since the actor optimization (19) is based on the assumption that the critic outputs action value estimations that fit the policy, harsh changes to the actor could lead the critic to get out of touch with the updated policy. Inaccurate action value estimation would then disrupt the training process. Thus, it is usually advised to also dampen the actor parameters when training the critic, as denoted in (17).

It should also be noted that the DDPG agent represents a parametric control algorithm defined by the actor network weights $\boldsymbol{\xi}$. To enable online-suitable control, only the actor inference $\boldsymbol{\pi}_{\boldsymbol{\xi}} : \boldsymbol{o}_k \to \boldsymbol{a}_k$ needs to be executed under real-time requirements while the learning of the actor and critic can be executed asynchronously.

REPLAY MEMORY

During training, the actor applies actions to the environments and receives transition experiences $(\boldsymbol{o}_k, \boldsymbol{a}_k, r_k, \boldsymbol{o}_{k+1}, d_{k+1})$. These are stored in a replay memory $\mathcal{D}$, which buffers the last experiences the agent has gathered. From this memory, multiple experiences are sampled in a training batch $\mathcal{B}$ to train the actor and critic networks. Those samples are independently drawn from the memory and are not necessarily temporally consecutive. Random sampling of multiple experiences from a longer history decorrelates the experiences in one training batch from each other, which results in gradient updates with more general improvement.

EXPLORATION NOISE

To avoid getting stuck in a local optimum, exploratory behavior is required. This is implemented by superimposing the actor's action choice $\boldsymbol{a}_k$ with random noise. Here, the discrete-time zero-mean Ornstein-Uhlenbeck-Process (OU) [39]

$$\tilde{\boldsymbol{a}}_k = \boldsymbol{a}_k + \underbrace{\boldsymbol{n}_{k-1} - \theta \boldsymbol{n}_{k-1}\Delta_t + \sigma\sqrt{\Delta_t}\mathcal{N}(0,1)}_{\boldsymbol{n}_k}, \tag{20}$$

is used as exploration noise with the sampling time Δ_t, the stiffness θ, the diffusion σ and the standard Gaussian noise process $\mathcal{N}(0, 1)$, respectively. The benefit of this process is that it is time-correlated. Having non-correlated random noise, which is the case for, e.g., standard Gaussian noise, will lead to fewer exploration throughout the observation space because consecutive actions are more likely to cancel each other out.

Algorithm 1: DDPG Pseudocode.

Randomly initialize weights of $q_{\boldsymbol{\zeta}}(\boldsymbol{o}, \boldsymbol{a})$ and $\boldsymbol{\pi}_{\boldsymbol{\xi}}(\boldsymbol{o})$
Initialize target weights accordingly: $\boldsymbol{\zeta}_{\text{target}} \leftarrow \boldsymbol{\zeta}$, $\boldsymbol{\xi}_{\text{target}} \leftarrow \boldsymbol{\xi}$
repeat
  observe $\boldsymbol{o}_k$
  select $\boldsymbol{a}_k = \boldsymbol{\pi}_{\boldsymbol{\xi}}(\boldsymbol{o}_k)$, calculate $\tilde{\boldsymbol{a}}_k$ according to (20)
  execute $\tilde{\boldsymbol{a}}_k$ and observe r_k and $\boldsymbol{o}_{k+1}$
  if $\boldsymbol{o}_{k+1}$ is terminal**then**
    set $d_{k+1} = 1$ and reset the environment
  else
    set $d_{k+1} = 0$
  end if
  save state transition experience to replay buffer $\mathcal{D} \leftarrow (\boldsymbol{o}_k, \tilde{\boldsymbol{a}}_k, r_k, \boldsymbol{o}_{k+1}, d_{k+1})$
  sample experience batch $\mathcal{B} \subset \mathcal{D}$
  update $\boldsymbol{\zeta}$ by minimizing J_{c} on $\mathcal{B}$, (17)
  update $\boldsymbol{\xi}$ by minimizing J_{a} on $\mathcal{B}$, (19)
  update weights of target networks:
    $\boldsymbol{\zeta}_{\text{target}} \leftarrow (1-\rho)\boldsymbol{\zeta}_{\text{target}} + \rho\boldsymbol{\zeta}$
    $\boldsymbol{\xi}_{\text{target}} \leftarrow (1-\rho)\boldsymbol{\xi}_{\text{target}} + \rho\boldsymbol{\xi}$
  $k \leftarrow k + 1$
until convergence condition is met

TABLE 1. Test Bench Parameters and Nameplate Test Motor Data

DC Power supply	Gustav Klein	
DC output	galvanically isolated	
Max. DC current	$i_{\text{DC,max}}$	500 A
DC output voltage	u_{DC}	300 V
IPMSM	Brusa HSM16.17.12-C01	
Stator resistance	R_{s}	18 mΩ
Inductance in d-direction	L_{d}	370 µH
Inductance in q-direction	L_{q}	1200 µH
Permanent magnet flux	ψ_{PM}	66 mVs
Pole pair number	p	3
Rated mechanical power	P_{me}	57 kW
Rated torque	T	130 N m
Max. current in dq-system	$\lvert\boldsymbol{i}_{\text{dq}}\rvert_{\max}$	250 A
Inverter	3×SKiiP 1242GB120-4D	
Typology	voltage source inverter, 2-level, IGBT, Integrated current measurement	
Max. phase current	$I_{\max}$	1200 A
Controller hardware	dSPACE	
Processor board	DS1006MC, 4 cores, 2.8 GHz	
IO boards	DS2004, DS5101, DS4302, DS3004	
Control cycle	100 µs	
Remote workstation		
Basic specs	i7-9800X, 128 GB DDR4, 2 TB SSD	

TABLE 2. Test Bench Setups With the Noise Parameters σ and θ

case	σ	θ	pre-training	actor layers
a	0.2	5	None	1
b	0.2	5	Yes, A	1
c	0.2	5	Yes, B	1
d	0.25	50	Yes, B	1
e	0.2	50	Yes, C	2

TABLE 3. DDPG Learning Parameters

Parameter	Value	Parameter	Value
training steps	250 000	discount factor γ	0.9
episode length	100 ms	batch size	128
sampling time	100 µs	memory interval	1
sampling frequency	10 kHz	actor learning rate	$5 \cdot 10^{-6}$
Memory Limit	5000	critic learning rate	$5 \cdot 10^{-4}$
warm up steps actor	2048	noise variance σ	0.2
warm up steps critic	1024	leaky ReLU α	0.1
target model update	1000	weight regularizer L2	0.01

IV. REINFORCEMENT LEARNING MODIFICATIONS

In this section, the modifications to the standard DDPG training are presented which boosted the real-world training process. Although the modifications are discussed based on the DDPG framework, they are equally useful and applicable for other RL algorithms in the motor control context.

A. BATCHED AND REMOTE REINFORCEMENT LEARNING

RL agents are usually trained after each iteration [23]. However, this is not possible with the utilized real-time setup, because waiting for a policy update every step would violate the real-time constraints of the controller. In the subsequent experimental tests, the controller is operating at 10 kHz (cf. Table 1), i.e., all computations on the embedded hardware need to be executed in less than 100 μs. This includes the measurement of observations, computation of actions and rewards, safety routines and the application of actions to the converter.

Thus, the training and the control inference of the DDPG agent shall be split up (cf. Fig. 1), since only the actor network is required to be implemented within the embedded controller. As shown in [9] the actor needs only a rather shallow ANN for function approximation. Hence, evaluating the actor ANN under hard real-time requirements is not a problem. The training of the DDPG agent, i.e., improving the actor and critic, is shifted to an asynchronous background task which is not required to be executed in real-time. In

TABLE 4. DDPG Artificial Neural Network Architectures

Actor		
Layer	**Width**	**Activation**
Input	7	/
Dense	100	LeakyReLU
Dense	2	linear

Critic		
Layer	**Width**	**Activation**
Input	9	/
Dense	75	LeakyReLU
Dense	75	LeakyReLU
Dense	75	LeakyReLU
Dense	1	linear

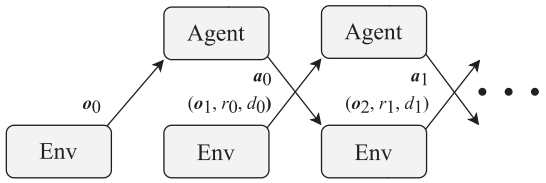

FIGURE 4. Concurrent execution of agent and environment.

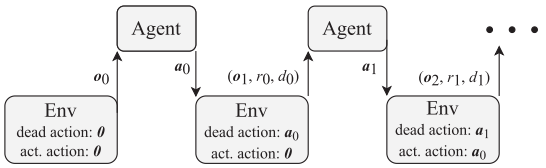

FIGURE 5. Concurrent execution mapped to alternating execution.

order to enable a rapid control prototyping pipeline that will evaluate different RL algorithms in the future, this background task is implemented on a flexible remote workstation with a non-real-time operating system. However, without loss of feasibility, the background task could easily be implemented on the embedded hardware, for example on system-on-a-chip (SoC) solutions with auxiliary machine learning hardware accelerators.

1) CONTROLLER

All the functions that belong to the environment in the RL setting are located on the controller. This includes the reward calculation, the generation of current references as well as observation of limits with a safety mechanism. Moreover, the controller contains DDPG agent functionality. These are the actor network to control the motor and the exploration noise to ensure exploratory behavior.

Up to τ_b consecutive experiences are recorded in a batch as one episode, while the RL controller is active. Next, the experiences are sent to the remote DDPG agent. The RL controller receives updated weights and uses them for a new recording.

Another important functionality of the controller is the safety mechanism to interrupt the RL control by the actor network, if motor constraints are violated (e.g., overcurrent or overheating). This terminates an episode of the RL agent. In this case, fewer than τ_b samples are sent to the remote agent. When the motor is set into a safe state and the weights are updated, the RL controller takes over the control again.

2) REMOTE DDPG AGENT

The remote DDPG agent contains most of the components of the DDPG agent described in Sec. III. A copy of the actor network, the critic network, the target networks and the replay memory are located in the remote DDPG agent. It receives the batches of experiences. Then, each experience is stored in the memory buffer, and a training step is performed like it is done in the regular DDPG algorithm until all experiences are processed. The remote DDPG agent sends the updated actor weights back to the embedded controller.

B. ACTOR AND CRITIC PRE-TRAINING

Training on real motors can be costly and also dangerous in case of limit violations. To speed up the training, pre-trained weights can be used. One option is to initialize weights with imitation learning [40]. The RL agent is trained to imitate a target controller with supervised learning. Another possibility is to make use of a simulative environment [10]. Due to the fact that a simulative motor model is only an approximation, a control error arises when a control algorithm is ported unaltered from simulation to reality. However, this can result in expedient initial weights for the beginning of the training, which might speed up the training significantly. Furthermore, safety critical constraints can be pre-learned in the simulation. This leads to fewer violations of safety limits during real-world training.

C. DIGITAL CONTROL DELAY COMPENSATION

Usually, RL agents interact with their environment in an alternating execution order. The environment responds with an observation and does not change its state while the agent calculates the next action. This is valid for typical RL problems like computer games or control problems with system dynamics that are much slower than the speed of action calculation.

Here, however, an action is always applied one time step delayed due to the digital control delay in the system. When the agent applies the action $\boldsymbol{a}_0$ that was calculated based on the last observation $\boldsymbol{o}_0$, the real environment state has changed already and displays the next observation $\boldsymbol{o}_1$. Consequently, the effects of action $\boldsymbol{a}_k$ can be observed in $\boldsymbol{o}_{k+2}$ and not instantly in $\boldsymbol{o}_{k+1}$ (cf. Fig. 4). Therefore, the reward r_k and terminal flag d_{k+1} are independent of the action $\boldsymbol{a}_k$. Hence, the interaction between agent and environment can be sketched with a concurrent scheme, see Fig. 4. Ignoring this fact can lead to longer and more unstable training. For example, the agent could see a high instant reward r_k for an action $\boldsymbol{a}_k$ that would actually lead to a low reward r_{k+1}.

The concurrent execution can be mapped to the alternating execution if the environment is modeled with a one-step delay as shown in Fig. 5. To overcome this additional hurdle, two changes have been implemented in the DDPG agent.

1) EXPERIENCE MODIFICATION

The experience $\boldsymbol{e}_k$ definition from (17) is extended to $\boldsymbol{e}_k = (\boldsymbol{o}_k, \boldsymbol{a}_k, r_{k+1}, \boldsymbol{o}_{k+2}, d_{k+2})$. Having added r_{k+1} and $\boldsymbol{o}_{k+2}$, an experience contains the reward and the next observation after the action $\boldsymbol{a}_k$ has actually taken effect on the electric motor.

2) ACTION FEEDBACK

Furthermore, the action $\boldsymbol{a}_{k-1}$ which has been played in the last cycle will be active in the next timestep. This action is appended to the observation $\boldsymbol{o}'_k = (\boldsymbol{o}_k, \boldsymbol{a}_{k-1})$ as proposed by [21]. With this information, the agent is able to estimate how the system will behave in the next time step. For example, after applying actions that lead to steep changes in the electrical current the agent might better use smaller actions to reduce overshooting the reference in the next time step. The actions from time step k have not had any effect on the system due to the digital control delay. A simple feed forward network as actor or critic approximation model cannot remember the previously played action. Therefore, it is fed back into the networks inputs as part of the observation. This allows the agent to comprehend causal relationships again [9].

D. REWARD FUNCTION AND SAFETY CONSTRAINTS

It must be ensured that the RL controller learns to comply with the safety constraints of the motors [12]. In electric motor control, especially the current constraint

$$i_{\mathrm{d},k}^2 + i_{\mathrm{q},k}^2 \leq i_{\max}^2 \tag{21}$$

is important to avoid overcurrent that could destroy the motor or the feeding inverter including the power supply (e.g., traction battery). To ensure that a trained RL agent complies with these constraints, a reward shaping approach is used [10]. In case of a limit violation, an additional penalty term r_{lim} is added to the reward of

$$r_k = w_1 \sum_{j \in \{\mathrm{d},\mathrm{q}\}} \sqrt{\frac{|i_{j,k}^* - i_{j,k}|}{i_{\max}}} + w_2 r_{\mathrm{lim}}. \tag{22}$$

Here, $\{w_1, w_2\} \in \mathbb{R} < 0$ are weighting parameters to balance the regular and the penalty component of the reward function.

The regular part of the reward function (22) is representing the motor current control problem following given reference trajectories i_j^* (e.g., from superimposed control loops). The root-function (22) delivers improved early and long-term training performance compared to the standard mean-squared-type rewards which are most common in tracking control problems. In particular, the steady-state control error can be reduced significantly compared to a mean-squared control error reward [10].

V. EXPERIMENTAL TEST SETUP

In this section, the experimental setup is presented. First of all, the workflow from simulation-based investigations via real-time controlled software-in-the-loop (SIL) models to the final test bench training session is described. Afterwards, the specific hardware architecture including the motor, controller and workstation is presented. Finally, important implementation details for the tests are described.

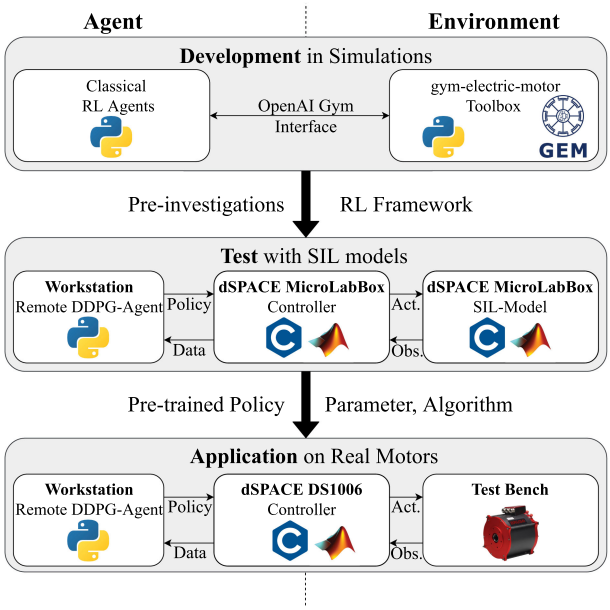

FIGURE 6. Setup of the development process including the online RL remote rapid control prototyping toolchain.

A. WORKFLOW FROM SIMULATION TO THE TEST BENCH

The development of RL motor controllers can be split into three steps as shown in Fig. 6. First, the gym-electric-motor toolbox [10] can be used with the standardized interface from OpenAI Gym [11]. Therewith, many different general-purpose RL agents from several Python libraries can be adapted and tested easily for this use case. Also, different investigations (e.g., on training parameters and network architectures) can be executed in a simple and quick manner. Afterwards, selected RL algorithms and parameter specifications are tested with the presented remote training setup on a real-time controlled SIL model utilizing an embedded rapid control prototyping hardware system. The batched learning under the real-time control and the proper transfer from a pure simulation framework to an embedded hardware framework is tested with this setup. Furthermore, the RL agent's weights are pre-trained in the SIL simulation. Finally, the chosen algorithm is trained and tested on the test bench. The training on the workstation as well as the controller can stay the same when exchanging the SIL model with the real motor. Solely the actions of the actor are applied to the laboratory inverter and the observations are received from measurement sensors.

B. HARDWARE SETUP

The nominal parameters of the test bench equipment and the motor used for experimental investigations are given in Table 1, while an illustrating picture of the test bench setup with the utilized (interior) PMSM motor in the background is

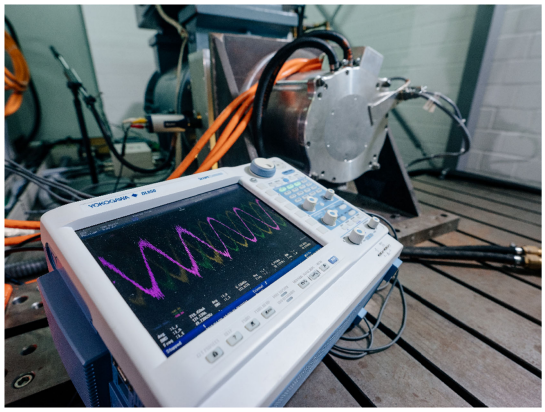

FIGURE 7. Test bench with the used PMSM in the background.

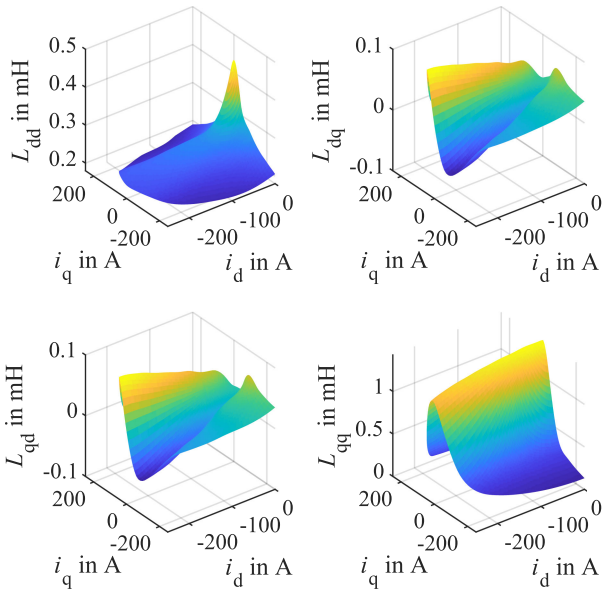

FIGURE 8. Differential inductances of the highly utilized test motor obtained from offline characterization.

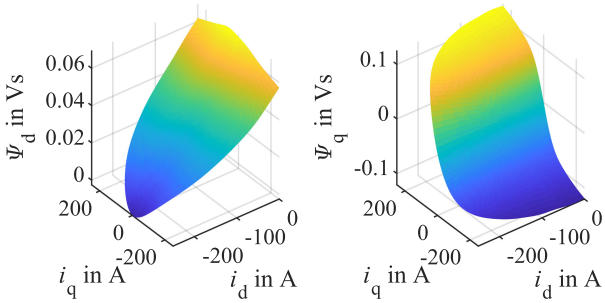

FIGURE 9. Flux maps of the highly utilized test motor obtained from offline characterization.

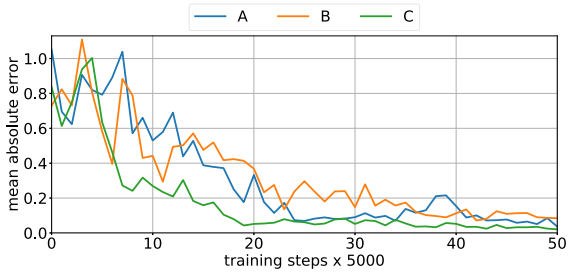

FIGURE 10. MAE of the pre-training in the SIL model.

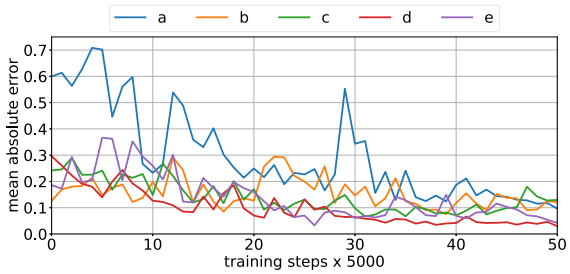

FIGURE 11. MAE between normalized actual and reference currents during training on the laboratory test bench.

shown in Fig. 7. To highlight that the motor under test shows significant (cross-)saturation effects within the normal operation range, the offline-identified flux and differential inductance maps are shown in Fig. 8 and Fig. 9. More information regarding the motor under test including open-source characterization measurement data highlighting its highly nonlinear behavior can be obtained from [41]. However, it should be noted that the offline-obtained data from Fig. 8 and Fig. 9 have not been made available to the RL training.

The controller and the SIL simulation models for the pre-training of the agents are built in Simulink and run as automated and exported C-code on a dSPACE MicroLabBox with a real-time kernel in the pre-training sessions. During the test bench investigations a similar dSPACE DS1006MC system is used, since it is already fully integrated into the laboratory test bench. Both, the MicroLabBox and the DS1006MC, are commercial off-the-shelf embedded hardware products for rapid control prototyping. Hence, they speed up the development process since the embedded C-code must not be written manually. As shown in Fig. 1, only the DDPG actor networks need to be implemented on the embedded hardware, while the actual RL training pipeline is executed remotely. Besides the DDPG actor, the embedded control framework incorporates the usual general measurement processing and safety protocols for motor control applications. A Python-based script controls and automates both the measurement recordings (gathering RL experience samples) and the actor weight updating using the ControlDesk interface, which is part of the dSPACE rapid control prototyping environment.

It should be noted that the specific implementation is adapted to dSPACE hardware and software, however, the general concept of edge computing-based RL with an asynchronous training pipeline decoupled from the

embedded actor is generalizable to any hardware setup. The main advantage is that the learning process can be executed on standard computer hardware with a full operating system (e.g., Linux) such that testing and tuning of different RL algorithms is easily possible without the need to transform the actual learning algorithms into the embedded world.

C. SOFTWARE SETUP

The remote DDPG agent on the workstation is a modified version of the DDPG implementation in keras-rl [28]. The DDPG agent offers several parameters which are summarized in Table 3. The actor and critic are structured as multi-layer perceptron ANNs with their specific designs summarized in Table 4. Hence, the RL agent is learning a direct, state-free (no internal integral action or similar) mapping between observations and actions to maximize its return. The configuration from Table 4 is only an example and is not to be understood as the best possible actor and critic configuration. Further ANN types (e.g., recurrent or convolutional ANNs) as well as the size of the networks (number of neurons and layers) can be investigated in terms of a superimposed hyperparameter optimization, as outlined in Sec. VII.

The applied action $\boldsymbol{a} = \begin{bmatrix} u_\mathrm{d}^* & u_\mathrm{q}^* \end{bmatrix}^\mathrm{T}$ is the reference voltage generated for the inverter. The observations shown to the agent consist of the following quantities:

$$\boldsymbol{o} = (i_\mathrm{d}, i_\mathrm{q}, \omega, a_{\mathrm{d},k-1}, a_{\mathrm{q},k-1}, i_\mathrm{d}^*, i_\mathrm{q}^*). \tag{23}$$

While the currents and the motor speed are standard measurements, the last applied voltage vector (action $\boldsymbol{a}_{k-1}$) is also utilized as an observation. This enables the RL agent to learn and compensate for the digital control delay (cf. Sec. IV-C). Finally, the reference current values are also part of the observation space to allow proper current control.

All quantities in the observation space are normalized to a range of $[-1, 1]$ with their limits. In the training, step-like reference changes $\boldsymbol{i}_\mathrm{dq}^* \in [-250\mathrm{A}, 250\mathrm{A}]$ are updated every 100ms from a uniform distribution (cf. [9], [10]). The reward function (22) is chosen. As exploratory action noise an OU process (20) is used with variable parameters θ and σ in the different experimental sessions and without a mean $\mu = 0$.

D. PRE-TRAINING OF THE MOTOR CONTROLLER

Three different RL agents are pre-trained with a SIL motor model, in order to evaluate whether the simulation-based pre-trained RL controller is learning faster compared to randomly initialized actor and critic networks during experimental investigations. During the training, the noise is reduced to $\sigma = 0.1$ after 100000 and to $\sigma = 0.0$ after 200000 steps. The pre-trained agents of training sessions *A* and *B* are trained on the networks as shown in Table 2. The pre-trained agent *C* has one additional hidden actor layer. For agent *B* the motor model inductance values were reduced to a third compared to case *A* to investigate parameter sensitivity. After the transfer of the pre-trained agent to the real motor, a better performance in the nonlinear regions is expected, because the effective inductances are reduced at higher current due to magnetic saturation. As shown in Fig. 10, all pre-trained models show a similar mean absolute error (MAE) averaged over 5000 consecutive samples after 250000 training steps.

VI. EXPERIMENTAL INVESTIGATION

The goal of these experimental investigations is to give a proof of concept for the presented simulation-to-experiment toolchain and to show that RL motor controllers are feasible, not only in simulation, but also in real experiments. Furthermore, the influence of different training parameters is investigated. The effects of pre-trained networks as well as different OU-noise parameters and two different actor network topologies are examined in the experiments. All subsequent experimental training and cross-validation results are obtained at a fixed motor speed of $n = 1000$ 1/min (maintained by a directly coupled speed-controlled load machine).

A. DDPG TRAINING ON A REAL MOTOR

A number of different training scenarios have been evaluated which are listed in Table 2. In case *a*, the controller is trained only on the real motor for 250000 steps. In the other cases, pre-trained networks from Sec. V-D are used for the initialization of the real motor training. The different noise parameters are also given in the table and were kept constant during the training. In case *e*, an actor network with one additional hidden layer is used.

The learning process of the RL controller is analyzed with the MAE during training. In Fig. 11, a large difference between the pre-trained cases and case *a* can be seen in the beginning. This is expected and shows that a simulative pre-training can be helpful in the beginning to speed up the training process on the real system. However, at the end of the training, the MAE s are all in the same range. This shows that complete training on a real motor is possible. The lowest MAE is achieved in case *d*. It seems to be nearly constant during the last 100000 training steps. The used exploration noise has faster dynamics compared to the cases *a*, *b* and *c* which seems to lead to better exploration. Also, the variations in the MAE during the training are smaller. It can be inferred from this that the exploration noise has a large impact on the training. Also case *e* with similar noise to case *d* performed well during the training with the larger actor network. Prior to these experiments, several tests with the larger network failed, which was not the case with the smaller network. Hence, the smaller network seems to be more robust and sufficient for this RL motor controller. Moreover, it should be noted that the DDPG algorithm requires active exploration to learn something, i.e., if the OU-noise is deactivated while the DDPG algorithm still actively performs policy updates there is a high likelihood that the agent diverges. Hence, when the learning is converging into steady state, both the exploration noise and the policy updates should be discontinued. In this case, the actor becomes a static policy (mapping observations to actions by the feed-forward ANN from Table 4) until active learning

and exploration are re-activated again (e.g., if the rewards are getting worse due to control plant changes).

Based on the various experimental training runs, a DDPG setup was selected for the subsequent transient and steady-state tests that delivered the comparatively best performance. The parameter setup used in the following is summarized in Table 3 and Table 4. Since the conceptual presentation of a rapid control prototyping pipeline for data-driven control approaches is the main focus of this contribution, it should be mentioned that no systematic hyperparameter optimization or feature engineering was performed for the DDPG controller, i.e., it can be assumed that the performance of this model-free controller can be further increased within future investigations.

B. COMPARISON TO STATE-OF-THE-ART CONTROLLERS

Finally, the DDPG-based controller is set against a linear field-oriented approach using PI controllers [42] and a continuous-control set (CCS) MPC [4]. For these two model-based controllers, the discrete-time representation of (13) is utilized

$$\begin{aligned}\boldsymbol{i}_{\mathrm{dq},k+1} = &\left(\boldsymbol{I} - \boldsymbol{L}_{\mathrm{dq}}^{-1}(\boldsymbol{i}_{\mathrm{dq},k}) R_{\mathrm{s}} \Delta_t\right) \boldsymbol{i}_{\mathrm{dq},k} \\ &+ \boldsymbol{L}_{\mathrm{dq}}^{-1}(\boldsymbol{i}_{\mathrm{dq},k}) \boldsymbol{T}(-\Delta_t \omega_k) \Delta_t \boldsymbol{u}_{\mathrm{dq},k} \\ &+ \boldsymbol{L}_{\mathrm{dq}}^{-1}(\boldsymbol{i}_{\mathrm{dq},k}) \left(\boldsymbol{T}(-\Delta_t \omega_k) - \boldsymbol{I}\right) \boldsymbol{\psi}_{\mathrm{dq},k}(\boldsymbol{i}_{\mathrm{dq},k})\end{aligned} \tag{24}$$

where x_k is the discrete-time sample of a given quantity, Δ_t is the time difference between two control cycles (assuming regular sampling) and $\boldsymbol{I}$ is the identity matrix [43]. To allow a fair comparison, the model (24) has been accurately parameterized using offline motor characterization (cf. Fig. 8 and Fig. 9) to cover magnetic (cross-)saturation effects and the inverter nonlinearity [31]. The results of the system identification have been made available by look-up tables to the two model-based controllers.

Using (24) for decoupling the dq-axes by a feed-forward compensation of the induced voltage [5], two independent PI controllers for the linear field-oriented approach have been designed using the symmetrical optimum [42]

$$K_{\mathrm{p,d/q}} = \frac{2}{3} \frac{L_{\mathrm{dd/qq}}(\boldsymbol{i}_{\mathrm{dq}})}{\kappa \Delta_t}, \quad K_{\mathrm{i,d/q}} = \frac{4}{9} \frac{L_{\mathrm{dd/qq}}(\boldsymbol{i}_{\mathrm{dq}})}{\kappa^3 \Delta_t^2} \tag{25}$$

with K_{p} and K_{i} being the proportional and integral gains as well as κ being the bandwidth design parameter, respectively. It should be noted that the PI controller is adaptively designed reacting to parameter changes as denoted in (25). The best PI controller performance was achieved for $\kappa = 3$ by experimental pre-testing at the test bench. Suitable anti reset-windup measures have been included to prevent an integrator runaway if the set voltage is clipped by the inverter limits (14).

For the MPC, the discussed control delay is compensated by an additional prediction step of the system states using (24) at the beginning of each control cycle. In order to retrieve a convex optimization problem, the MPC problem definition is

$$\begin{aligned}\min_{\boldsymbol{a}} \sum_{k}^{N} c(\boldsymbol{o}_k, \boldsymbol{a}_k) = \min_{\boldsymbol{a}} \sum_{k}^{N} \left\| \boldsymbol{i}_{\mathrm{dq},k}^{*} - \boldsymbol{i}_{\mathrm{dq},k} \right\|_2 \\ \text{s.t.} \quad \boldsymbol{i}_{\mathrm{dq},k+1} = \boldsymbol{f}(\boldsymbol{i}_{\mathrm{dq},k}, \boldsymbol{a}_k, \ldots), \ \boldsymbol{h}(\boldsymbol{o}_k, \boldsymbol{a}_k) \leq 0\end{aligned} \tag{26}$$

where c is a quadratic cost function with respect to the current reference tracking, $\boldsymbol{f}$ is the system model (24) and $\boldsymbol{h}$ is a linear inequality action space constraint due to the inverter limitation as in (14). To solve the linearly constrained quadratic program (LCQP) (26), the embedded QP solver from the Matlab MPC toolbox [44], [45] was utilized with $N = 1$ prediction steps.

TABLE 5. MRE, MAE and MSE values According to (27) Given a Transient Test Benchmark (Cf. Fig. 12, Fig. 14 and Fig. 16)

	MRE	MAE	MSE
DDPG	17.08 %	1.97 %	0.18 %
PI	18.36 %	2.43 %	0.27 %
MPC	19.01 %	2.41 %	0.19 %

C. TRANSIENT TESTS

The trained DDPG agent and the model-based controllers are tested based on a sequence of random current reference jumps within the entire left i_{d}-i_{q} half-plane. To compare the following results, different performance metrics are used. The basic metric is

$$\rho_m = \frac{1}{100000} \sum_{k=1}^{100000} \sum_{j \in \{\mathrm{d,q}\}} \left| \frac{i_{j,k}^{*} - i_{j,k}}{i_{\max}} \right|^m \tag{27}$$

with different power values m applied to the test recordings. Those are $m = \{0.5, 1, 2\}$ for mean square root error (MRE), mean absolute error (MAE) and mean squared error (MSE), respectively. Note the difference between root mean square error (RMSE) and mean square root error (MRE). The motivation for the MRE metric arises from its beneficial scaling effect in the reward function.

The transient benchmark results are summarized in Table 5 and the test time series profiles are shown in Fig. 12 to Fig. 17. First of all, it should be noted that the DDPG controller can be operated stably and safely over the entire operating range. The transient performance is satisfactory and in comparison the DDPG controller is on par with the MPC but clearly better than the adaptive PI controller.

It may also be noted that all current measurements show a rather high level of noise independently of the specific control approach. This is due to the integrated current measurement sensors within the utilized voltage source inverter (cf. Table 1) whose maximum measurement range substantially exceeds the exemplary test motor's operation range. Nevertheless, unfavorable measurement noise is a typical problem in many industrial applications and a given motor control scheme must be able to handle it. All three control methods compared here are able to do so, whereby the RL controller performs

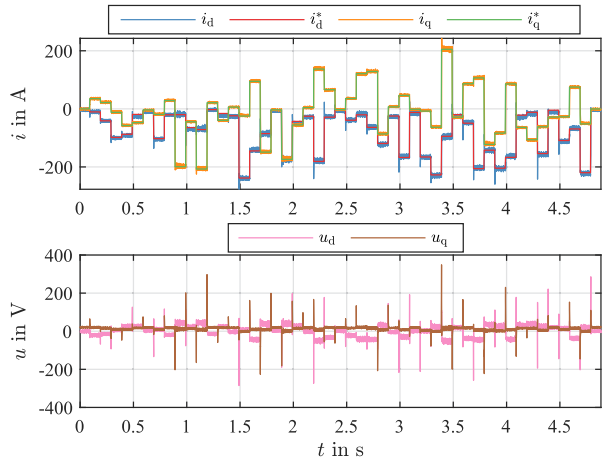

FIGURE 12. Test episode of the DDPG RL controller with random current reference changes (fixed speed $n = 1000$ 1/min).

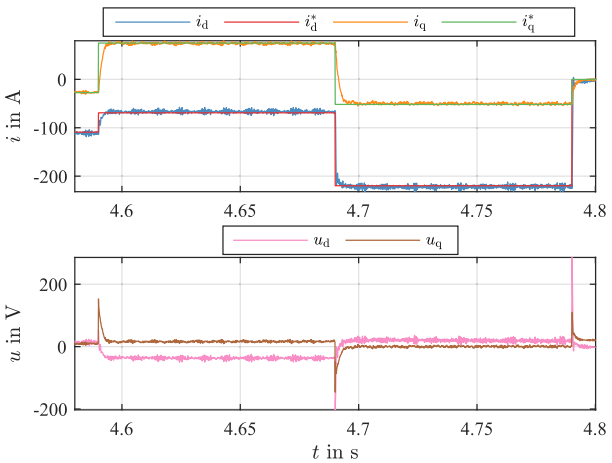

FIGURE 13. Zoom into previous Fig. 12 to highlight the transient performance of the DDPG RL controller.

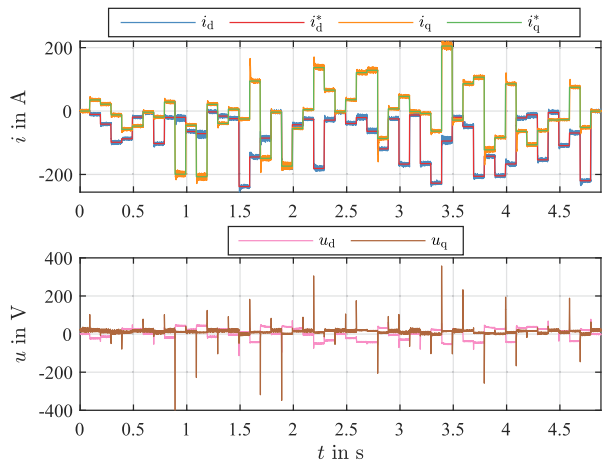

FIGURE 14. Test episode of the PI controller with random current reference changes (fixed speed $n = 1000$ 1/min).

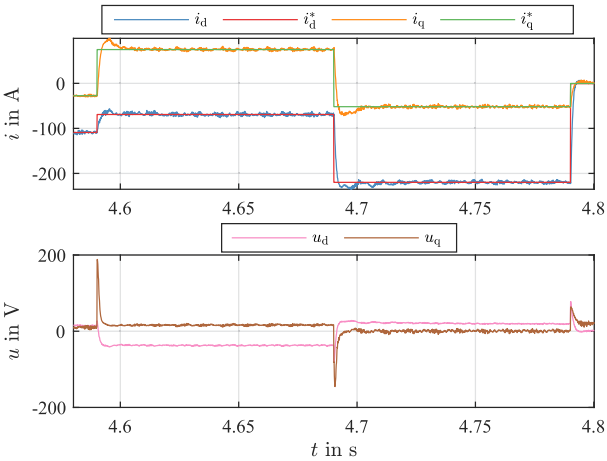

FIGURE 15. Zoom into previous Fig. 14 to highlight the transient performance of the PI controller.

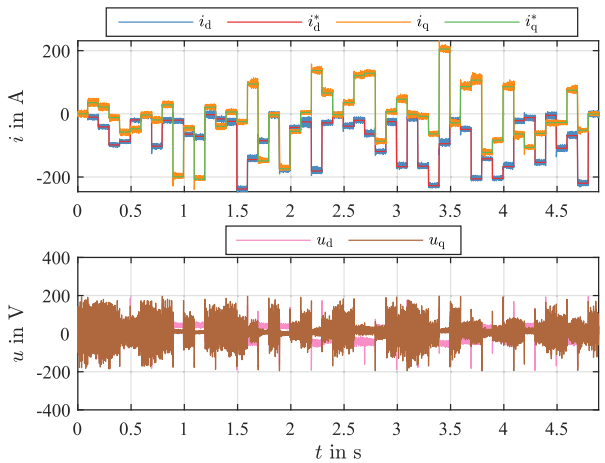

FIGURE 16. Test episode of the CCS-MPC with random current reference changes (fixed speed $n = 1000$ 1/min).

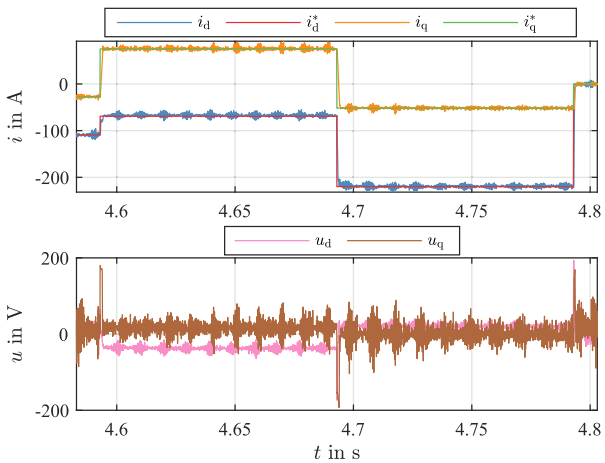

FIGURE 17. Zoom into previous Fig. 16 to highlight the transient performance of the CCS-MPC.

TABLE 6. Turnaround Times on the Embedded dSPACE System (Using Only One Core of the DS1006 CPU)

	DDPG	PI	MPC
Mean	29.99 µs	25.78 µs	27.53 µs
Std.	0.405 µs	0.387 µs	0.407 µs

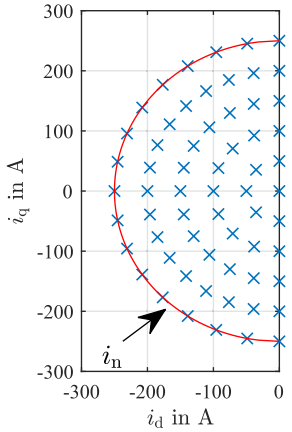

FIGURE 18. Grid of reference points for steady-state tests in the left i_d-i_q half-plane.

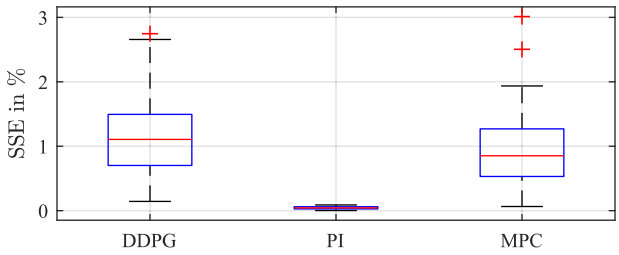

FIGURE 19. Box plots of the SSE distribution of each control technique given the steady-state references from Fig. 18.

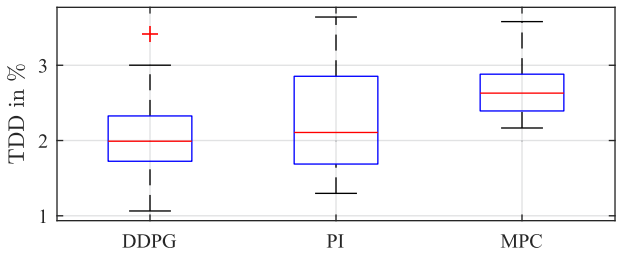

FIGURE 20. Box plots of the TDD distribution of each control technique given the steady-state references from Fig. 18.

minimally better during the stationary phases than the state-of-the-art approaches.

Additionally, the controller execution time on the embedded hardware was measured during the transient tests. The results are summarized inTable 6. As expected, the PI controller requires the least computational burden whereby the majority of the execution time is not caused by the actual controller, but by auxiliary functions (measurement acquisition and processing, safety monitoring, etc.). Nevertheless, both the MPC and DDPG only require marginally more computation time. It should also be considered that when using parallel computing devices (FPGA, neural cores, etc.), the calculation time of the DDPG actor can be reduced compared to the implementation shown here on a sequentially working CPU.

D. STEADY-STATE TESTS

To evaluate the steady-state performance of the different control approaches an evenly distributed grid of reference operating points has been designed as shown in Fig. 18. Each controller has been operated in each reference point for 2s given a constant speed of $n = 1000$ 1/min enforced by a load machine. For quantitative evaluation of the steady-state controller behavior, the following benchmark quantities are introduced:

$$\mathrm{SSE} = \frac{1}{N}\sum_{k=0}^{N}\left\|\frac{\boldsymbol{i}^{*}_{\mathrm{dq},k} - \boldsymbol{i}_{\mathrm{dq},k}}{i_{\mathrm{n}}}\right\|_2, \qquad \mathrm{TDD} = \frac{\sqrt{\sum_{h\neq 1} I_h}}{I_{\mathrm{n}}}. \tag{28}$$

Here, the SSE is the normalized steady-state error between the reference and the momentary current and the TDD is the averaged total demand distortion with I_h being the h-th current harmonic and I_{n} being the nominal RMS motor current. The TDD is chosen as a benchmark quantity instead of the total harmonic distortion (THD), since the fundamental current component may be different depending on the given steady-state control error which would distort the THD.

The results of the steady-state control accuracy are shown in Fig. 19. As expected, the PI controller is performing best due to its integral control action being able to compensate for any model deviation. The MPC and DDPG approach show similar steady-state accuracy, whereby the observed stationary deviations are still tolerable. It should be taken into account that the MPC has a very precise, adaptive time-discrete motor model at its disposal, which was hand-picked and pre-parametrized on the test bench during an offline characterization. In contrast, the RL-based controller is able to achieve comparable steady-state control accuracy within a short period of training time without any model knowledge, solely on the basis of the data-driven learning process. Moreover, it should be mentioned that the stationary accuracy of both methods can be further increased: For the MPC the coupling with a disturbance observer [46] is possible, while for the DDPG controller a comprehensive hyperparameter optimization and feature engineering as well as the consideration of recurrent neural networks in the actor [47] (in analogy to standard integral controllers) are possible.

In addition, the results regarding the current harmonics by means of the TDD in steady-state operation are depicted in Fig. 20. Here, the DDPG controller is outperforming the

PI and MPC approach, which was already indicated by the evaluation of the transient test profiles.

E. RESULT DISCUSSION

In summary, the presented exemplary RL agent's control performance is on par with the established model-based control procedures. While the MPC approach used here is the state-of-the-art solution after more than 20 years of active, extensive research in the field of power electronics, the data-driven RL controller is at a very early research stage which to the best of the authors' knowledge has now been demonstrated for the first time in a real-world power electronics application. The fact that a RL-based, model-free controller performs at eye level compared to MPC without the use of expert knowledge is therefore an important intermediate step in the research of data-driven control methods from the machine learning domain.

The exemplary investigated DDPG agent is only one possible control approach from the field of reinforcement learning, which itself has a lot of potential for improvement, because for this work no hyperparameter optimization or feature engineering regarding the observation space or the reward signal has been performed yet. It can therefore be assumed that further improvements can be realized in the future by systematic optimization. On the other hand, it should be emphasized once again that the central scientific contribution of this work is the rapid control prototyping simulation-to-experiment toolchain and the associated experimental proof-of-concept as a methodical piece in the puzzle to introduce data-driven approaches in real-world power system control. It is not claimed that the presented RL performance is outperforming the MPC approach as the state-of-the-art control solution at the current point of time.

VII. CONCLUSION AND OUTLOOK

In this work, the transfer of RL electric drive control from offline simulation to online real-world learning was successfully presented. An RL-based PMSM current controller was experimentally trained solely using a measurement data stream without any expert or model knowledge. Several modifications to the classical DDPG training algorithm are presented that enabled the successful online training on a laboratory drive test bench. The presented rapid control prototyping pipeline allows fast and flexible testing of RL algorithms since the learning is shifted to an asynchronous background task. Here, the used remote workstation is only optional in order to speed-up the learning process, but the presented training scheme can be implemented on typical SoC embedded hardware, too. The embedded part of the RL agent (actor) could be implemented on a typical laboratory hardware system without any real-time problems and due to the continuous development of control electronics with specialized parallel processing units for the matrix algebra of machine learning (FPGA, neural cores, etc.), an implementation in typical industrial applications will also be possible for low-cost applications in the future. Furthermore, it is conceivable to outsource the training process to an edge or cloud computing framework, which allows completely new possibilities in the control and monitoring of power electronic systems.

Moreover, there is much space for future research in this field. The training process needs to be optimized for sample efficiency and, therefore, to accelerate the learning process. An adaptive parameter setting of learning rates and noise parameters could be part of an upgraded training process. Hyperparameter optimization and extended feature engineering for the RL agent is also very likely to improve the overall learning and control performance. This includes particularly the possibility to use recurrent ANNs (with corresponding internal states / memory cells) as the actor, enabling the RL agent to learn some kind of integral feedback to minimize the steady-state control error. Also, investigations regarding the safety constraints are important for real-world control engineering applications. Additionally, transferring the RL-based controller framework to other tasks such as torque or speed control for PMSM and other motor types including finite-control-set approaches is highly interesting. And finally, using the rapid control prototyping toolchain for investigating the performance of entirely different RL algorithms on an experimental basis is of prime interest.

REFERENCES

[1] M. Brasel, "A gain-scheduled multivariable LQR controller for permanent magnet synchronous motor," in *Proc. Int. Conf. Methods Models Automat. Robot.*, 2014, pp. 722–725.

[2] C. Xia, N. Liu, Z. Zhou, Y. Yan, and T. Shi, "Steady-state performance improvement for LQR-based PMSM drives," *IEEE Trans. Power Electron.*, vol. 33, no. 12, pp. 10622–10632, Dec. 2018.

[3] A. Linder, R. Kanchan, P. Stolze, and R. Kennel, *Model-Based Predictive Control of Electric Drives*, *1st ed.* Göttingen: Cuvillier Verlag, 2010.

[4] T. Geyer, *Model Predictive Control of High Power Converters and Industrial Drives*, *1st ed.* Hoboken, NJ, USA: Wiley, 2017.

[5] W. Peters and J. Böcker, "Discrete-time design of adaptive current controller for interior permanent magnet synchronous motors (IPMSM) with high magnetic saturation," in *Proc. Annu. Conf. IEEE Ind. Electron. Soc.*, 2013, pp. 6608–6613.

[6] S. Lee, "Closed-loop estimation of permanent magnet synchronous motor parameters by PI controller gain tuning," *IEEE Trans. Energy Convers.*, vol. 21, no. 4, pp. 863–870, Dec. 2006.

[7] S. J. Underwood and I. Husain, "Online parameter estimation and adaptive control of permanent-magnet synchronous machines," *IEEE Trans. Ind. Electron.*, vol. 57, no. 7, pp. 2435–2443, Jul. 2010.

[8] R. S. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, *2nd ed*. Cambridge, MA, USA: MIT Press, 2018.

[9] M. Schenke, W. Kirchgässner, and O. Wallscheid, "Controller design for electrical drives by deep reinforcement learning: A proof of concept," *IEEE Trans. Ind. Informat.*, vol. 16, no. 7, pp. 4650–4658, Jul. 2020.

[10] A. Traue, G. Book, W. Kirchgässner, and O. Wallscheid, "Towards a reinforcement learning environment toolbox for intelligent electric motor control," *IEEE Trans. Neural Netw. Learn. Syst.*, to be published, doi: 10.1109/TNNLS.2020.3029573.

[11] G. Brockman *et al.*, "OpenAI Gym," 2016. [Online]. Available: *arXiv:1606.01540*.

[12] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," [Online]. Available: http://arxiv.org/pdf/1904.12901v1

[13] A. Marco *et al.*, "Virtual vs. real: Trading off simulations and physical experiments in reinforcement learning with Bayesian optimization," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 1557–1563.

[14] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[15] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.

[16] B. Scherrer, V. Gabillon, M. Ghavamzadeh, and M. Geist, "Approximate modified policy iteration," 2016. [Online]. Available: http://arxiv.org/pdf/1205.3054v2

[17] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa, "Safe exploration in continuous action spaces," 2018, *arXiv:1801.08757*.

[18] T.-H. Pham, G. de Magistris, and R. Tachibana, "OptLayer - practical constrained optimization for deep reinforcement learning in the real world," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 6236–6243.

[19] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 22–31.

[20] R. de Doncker, D. W. Pulle, and A. Veltman, *Advanced Electrical Drives: Analysis, Modeling, Control*. Berlin, Germany: Springer Science Business Media, 2011.

[21] T. Hester and P. Stone, "TEXPLORE: Real-time sample-efficient reinforcement learning for robots," *Mach. Learn.*, vol. 90, no. 3, pp. 385–429, 2013.

[22] C.-C. Hung *et al.*, "Optimizing agent behavior over long time scales by transporting value," *Nature Commun.*, vol. 10, no. 1, pp. 1–12, 2019.

[23] T. Lillicrap *et al.* "Continuous control with deep reinforcement learning," 2016, *arXiv:1509.02971*.

[24] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[25] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016.

[26] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, "Stable baselines3," 2020. [Online]. Available: https://github.com/DLR-RM/stable-baselines3

[27] S. Guadarrama *et al.*, "TF-Agents: A library for reinforcement learning in tensorflow," 2018. [Online]. Available: https://github.com/tensorflow/agents

[28] M. Plappert, "Keras-Rl," 2016. [Online]. Available: https://github.com/keras-rl/keras-rl

[29] J. Chiasson, *Modeling and High-Performance Control of Electric Machines*. Hoboken, NJ, USA: Wiley, 2005.

[30] M. Hinkkanen, H. Awan, Z. Qu, T. Tuovinen, and F. Briz, "Current control for synchronous motor drives: Direct discrete-time pole-placement design," *IEEE Trans. Ind. Appl.*, vol. 52, no. 2, pp. 1530–1541, Mar./Apr. 2016.

[31] O. Wallscheid, A. Specht, and J. Böcker, "Observing the permanent-magnet temperature of synchronous motors based on electrical fundamental wave model quantities," *IEEE Trans. Ind. Electron.*, vol. 64, no. 5, pp. 3921–3929, May 2017.

[32] M. Fasil, C. Antaloae, N. Mijatovic, B. B. Jensen, and J. Holboll, "Improved dq-Axes model of PMSM considering airgap flux harmonics and saturation," *IEEE Trans. Appl. Supercond.*, vol. 26, no. 4, pp. 1–5, Jun. 2016.

[33] M. Ott and J. Böcker, "Sensitivity analysis on production tolerances for electric drive systems in automotive application," in *Proc. Eur. Conf. Power Electron. Appl.*, 2016, pp. 1–10.

[34] F. Chai, P. Liang, Y. Pei, and S. Cheng, "Analytical method for iron losses reduction in interior permanent magnet synchronous motor," *IEEE Trans. Magn.*, vol. 51, no. 11, pp. 1–4, Nov. 2015.

[35] A. Al-Timimy, P. Giangrande, M. Degano, M. Galea, and C. Gerada, "Investigation of AC copper and iron losses in high-speed high-power density PMSM," in *Proc. Int. Conf. Elect. Mach.*, 2018, pp. 263–269.

[36] D. C. Hanselman, *Brushless Permanent Magnet Motor Design. The Writers' Collective*, 2003.

[37] M. Stender, O. Wallscheid, and J. Boecker, "Comparison of gray-box and black-box two-level three-phase inverter models for electrical drives," *IEEE Trans. Ind. Electron.* to be published, doi: 10.1109/TIE.2020.3018060.

[38] M. Seilmeier, C. Wolz, and B. Piepenbreier, "Modelling and model based compensation of non-ideal characteristics of two-level voltage source inverters for drive control application," in *Proc. Int. Electric Drives Prod. Conf.*, 2011, pp. 17–22.

[39] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the Brownian motion," *Phys. Rev.*, vol. 36, no. 5, pp. 823–841, 1930.

[40] M. Novak and T. Dragicevic, "Supervised imitation learning of finite set model predictive control systems for power electronics," *IEEE Trans. Ind. Electron.*, vol. 68, no. 2, pp. 1717–1723, Feb. 2021.

[41] S. Hanke, O. Wallscheid, and J. Böcker, "Data set description: Identifying the physics behind an electric motor - data-driven learning of the electrical behavior (Part I)," 2020. [Online]. Available: https://arxiv.org/abs/2003.07273.

[42] D. Schröder, *Elektrische Antriebe - Regelung Von Antriebssystemen*. Berlin, Heidelberg, Germany: Springer, 2009.

[43] A. Specht, O. Wallscheid, C. Romaus, J. Böcker, and S. Ober-Blöbaum, "Discrete-time model of an IPMSM based on variational integrators," in *Proc. IEEE Int. Elect. Mach. Drives Conf.*, 2013, pp. 1411–1417.

[44] C. Schmid and L. Biegler, "Quadratic programming methods for reduced hessian SQP," *Comput. Chem. Eng.*, vol. 18, no. 9, pp. 817–832, 1994.

[45] N. Khaled and B. Pattel, *Practical Design and Application of Model Predictive Control: MPC for MATLAB and Simulink Users*. London, U.K.: Butterworth-Heinemann, 2018.

[46] O. Wallscheid and F. Ngoumtsa, "Investigation of disturbance observers for model predictive current control in electric drives," *IEEE Trans. Power Electron.*, vol. 35, no. 12, pp. 13 563–13572, Dec. 2020.

[47] Z. Zou, X. Yu, and S. Ergan, "Towards optimal control of air handling units using deep reinforcement learning and recurrent neural network," *Building Environ.*, vol. 168, 2020, Art. no. 106535.