22rd International Conference on Renewable Energies and Power Quality (ICREPQ'24)
Bilbao (Spain), 26th to 28th June 2024
ISBN: 978-84-09-60656-6

Useability Evaluation of Reinforcement Learning Toolboxes for Electrical Drives

N. Szécsényi[1] and P. Stumpf[1]

[1] Department of Automation and Applied Informatics, Budapest University of Technology and Economics
Műegyetem rkp. 3., H-1111 Budapest, Hungary

Abstract. The current direction of development predicts that Reinforcement Learning based data driven control methods can become a next generation technology to control electrical drives instead of the classical model-based techniques. The paper aims to evaluate toolboxes that can be used to train agents for control approaches. The paper helps lay the theoretical bases and provides guidelines for using these toolboxes via a case study. This is done to highlight each toolbox's key aspects and workflow patterns, shifting the comparison to useability and peak-performance.

Key words. Artificial Intelligence, Reinforcement Learning, Electrical Machines & Drives, Permanent Magnet Synchronous Machine, Power Electronics

1. Interest of the work

The last years have witnessed an enormous interest in the use of artificial intelligence (AI) techniques in different engineering fields. AI has the capability to facilitate systems with intelligence that is capable of human-like learning and reasoning. Among the different categories of AI, machine learning is used primarily in the research of energy management in renewable-based power distribution applications, power electronic systems and control or monitoring of electrical drives [1], [2]. Reinforcement Learning (RL), which is an area of machine learning, can be considered as a viable solution to many decision and control problems across different time scales. For this reason, current manuscript also focuses on this technique.
Multiphase electric machines – either asynchronous or synchronous - are widely used in renewable energy applications such as wind power generation, tidal energy utilization or electric vehicles. A proper control approach is essential for high performance electric drives, as it directly affects performance of the overall system. Field Oriented Control (FOC), Direct Torque Control (DTC) and Model Predictive Control (MPC) are the most used standard methods to control electrical machines. However, recently increasing attention has been given to data-driven approaches, that do not require an explicit plant model, like these classic control schemes [3]. In the case of RL based model-free approaches the control policy is continuously improved through learning to obtain optimal performance. The main drawback of this method is that the training phase can be time-consuming and the agent settings and parameter selection are not straightforward and require an in-depth knowledge. Furthermore, the training phase should be repeated for each drive system.
Closed- as well as open-source toolboxes are available for ease of training of an RL agent for electric motor control. The main goal of the manuscript is to present and evaluate these RL toolboxes by presenting their features via a case study. Furthermore, it also aims to provide some guidelines that may be used in solving similar problems. As a case study, the feasibility of RL toolboxes is demonstrated on the FOC of a Permanent Magnet Synchronous Machine (PMSM), where the inner linear current controllers are replaced by an RL agent.

2. Basics of Reinforcement Learning

When it comes to machine learning, three different settings can be achieved based on the input and output provided to the neural network. The basic setting is when the model has access to inputs paired with the correct output values, called supervised learning, used mainly for classification tasks. When the desired outputs are not fully available or cannot be acquired, the process is called semi-supervised or unsupervised learning, frequently used in clusterization and dimension reduction. The third option, called RL, is when the model does not learn based on a correct output but rather tries to maximize a long-term reward function by choosing the suitable set of actions to take. This approach is used for tasks where the most optimal solution is not known in advance or very hard to compute so the model can learn it by defining a reward function based on the requirements. For possible applications first video games, self-driving and other action-based environments come to mind, but this method can also be used efficiently in the domain of power electronics, more precisely motor control. This is already backed up by research where it has been showcased that the RL based methods can achieve the same level of performance as the traditional approaches [3], [4].
Although the focus of this paper is not on the architecture of the RL models, it is still important to introduce the basic layout of these methods so that the role of each part can be cleared in advance. To visualize the basic components, Fig. 1. is provided where the example outlines of a RL model is shown.
This model can be broken down into two main components: the environment and the agent. The environment encapsulates the system that we want to control, in our case, the electric drive itself. It also includes any disturbances or factors, like noise or load. The agent

gets observations from this environment and based on them determines its state. Using the available information, the agent decides on the next action which will be applied to the environment to change its state in the desired manner. After this input, a suitable reward is calculated according to the next state, which the agent tries to maximize by altering its actions in a way which will, after many iterations, reliably keep the environment in a desired state.

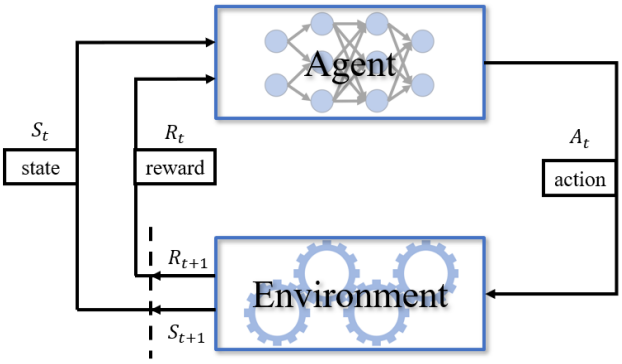

Fig. 1. General Layout of a Reinforcement Learning Setup

3. Available Reinforcement Learning Toolboxes for Electric Drives

As more and more advances are being made in the field of RL based control of electrical drives, the need for an adapt development environment is getting increasingly larger. There are several tools and software already available for this task, but they offer different perks and functions so in this paper a thorough comparison is made between them to establish guidelines when it comes to choosing the development environment to work with. The most used toolboxes can be split into two groups based on the software environment they use: MATLAB or Python (see Fig. 2). The first is a commercial choice while the second one mostly has open-source toolboxes.

In the case of power electronics and motor control, MATLAB offers the Simulink modelling software with several products, one of which is the Motor Control Blockset. This, combined with the Reinforcement Learning Toolbox, makes a complete environment for Artificial Intelligence based motor control solutions. The support for deployment on target hardware (microcontrollers, FPGAs or others) is also a great asset of this environment.

As for the Python based environment, the basis of a RL model is the Gymnasium package, which defines a general interface regarding the specific model environment. Using this, a setup for any task can be modelled quite intuitively, but in the case of electric motors, this can be combined with the so-called GEM package. This is developed specifically for the modelling of electric drives and already has several frequently used predefined setups. Moreover, many of the most used Artificial Intelligence packages (Tensorflow, PyTorch) are also available with additional workflow support modules like RLlib and Stable Baselines. Altogether, these components provide a great deal of flexibility due to their modular structure and open-source code, making it possible to build an environment for every task.

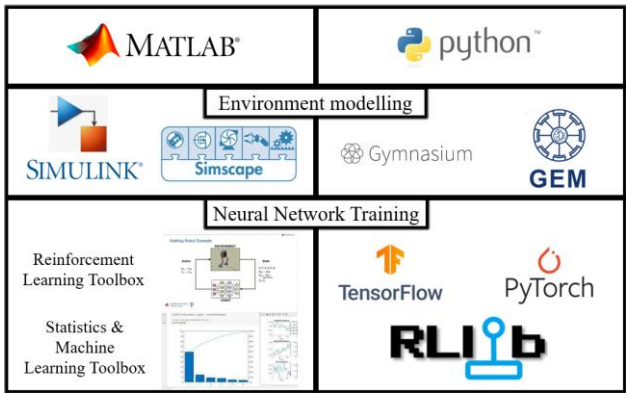

Fig. 2. Reinforcement Learning toolboxes

4. Reinforcement Learning based FOC of a PMSM drive

As it was mentioned previously, the feasibility of RL toolboxes is demonstrated on the FOC of a surface mounted PMSM, where the inner linear current controllers are replaced by an RL agent. This scheme can be considered as a basic workflow for evaluating the performance of RL agents in electric drive systems [6], [7]. Figure 3 presents the simplified schematic of the overall drive system, where the d axis reference current i_d^* is forced to zero and a speed constraint is assumed, where the actual mechanical angular speed Ω is determined by the loading machine.

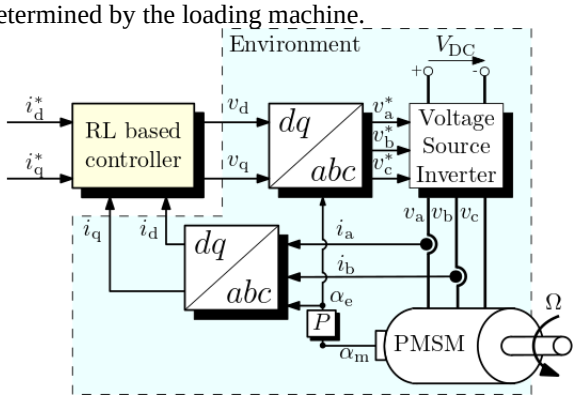

Fig. 3. Simplified schematic of the drive system

Following through the process of training and evaluating the performance of the RL based FOC separately in the two software environments will highlight the key features, making the comparison more balanced. To make it even more unbiased, the same general neural network architecture and the same type of RL algorithm is used in both settings.

A. *Environment modelling*

Typically, PMSMs are modelled in the *dq* rotating coordinate system, where the vector of the pole flux is aligned with the *d* axis. Both the MATLAB/Simulink environment and the GEM toolbox provide, among several other DC and AC machine types, a built-in PMSM model. In the basic GEM package, only a linear model is available currently, where the motor parameters are constant. The toolboxes of MATLAB/Simulink can offer different PMSM models, which give a more realistic machine model. In both cases, it may be necessary to implement a

custom PMSM model, to model nonlinearities, like parameter dependencies, cross saturation, spatial harmonics and iron loss. In this case, Simulink provides a much more intuitive and simpler environment than the Phyton-based approach.
The PMSM is fed from a Voltage Source Inverter (VSI), which is also part of the environment. The RL agent can directly provide the switching signals or the required *d* and *q* axis voltages v_{d} and v_{q}, which can be formed by some PWM method. MATLAB/Simulink offers much more possibilities to model a VSI, than GEM, as many inverter topologies can be tested and the nonlinearities of the inverter can be modeled also quite easily. The basic GEM toolbox currently provides an ideal two-level VSI model for driving the PMSM.
In summary, the main difference in the modeling of the drive system between the two environments is the level of likeness to a real-life setting: the MATLAB model can provide a very detailed machine and inverter model, while the Python based GEM environment models them with only a few parameters. This becomes a key aspect when the trained RL agent is transferred to a real-life motor control system, as there it can experience several effects it has not seen during training, making its performance unstable, so it is advised to introduce them at an early stage in the development.
For the fair comparison, in the paper, the training and the testing phase of the RL agent are performed for an ideal PMSM model assuming constant parameters and the machine is supplied from an ideal VSI.

B. Reinforcement Learning Agent

Typically, Deep Q-network (DQN), Deep Deterministic Policy Gradient (DDPG) and Twin Delayed DDPG (TD3) algorithms are used for RL agents in electric drives and power electronics. Both MATLAB and Phyton libraries support these techniques. In the paper the TD3 algorithm was selected which, as its name suggests, addresses some issues of the classic DDPG method [5]. One of the main aspects of this choice is that it can deal with continuous action spaces, so the output of the agent will be the direct voltages instead of the switching values, as those also have to be optimized which the agent is not primarily encouraged to learn. Another feature is that it trains two separate neural networks during learning: the actor and the critic network. The first one determines the necessary action to take with respect to the observations it receives as input, while the critic network is responsible for producing a predicted reward value for this specific action which updates the policy of the RL agent to reach higher long-term rewards.
One of the most important factors in the success of the RL agent is the definition of the reward function, as it determines the output the agent receives during training and this is the metric which it tries to maximize in the long run. During this experiment, we have elected to use a formula based on the Mean Root Error (MRE) metric, which is frequently used in RL based motor control [6], [7], as this returns a relatively higher reward even for differences very close to zero:

$$r_{\mathrm{k,i}} = \sum_{p\in\{d,q\}} \sqrt{\frac{\left|i^{*}_{\mathrm{p,k}} - i_{\mathrm{p,k}}\right|}{i_{\mathrm{limit}}}}$$

where $i^{*}_{\mathrm{p,k}}, i_{\mathrm{p,k}}$ are the reference and motor current at timestep k respectively and i_{limit} is the current limit of the motor to ensure a safe working condition. Moreover, since every quantity is normalized between -1 and 1, the reward function stays consistent and does not vanish at later stages of training.
The proper generation of reference input signals (like i^{*}_{q} or Ω in our case) is very important for diverse training. It is advised to choose a signal shape that covers a large range of the input space while still not too difficult to learn for the agent and does not violate the previously set limit values. From reference generation point of view, Phyton based GEM can provide much wider possibilities than MATLAB. Not only classic sinusoidal, triangular and step functions can be chosen, but also the Wiener process can be used. Furthermore, these signals can have train-by-train randomly selected time period, amplitude and offset making the RL agent more robust.

5. Results

The TD3 agents were trained and tested for a PMSM drive system in both environments. The parameters of the drive system can be seen in Table I. while the general structure of the neural networks is outlined in Table II.

Table I. – Parameters of the PMSM motor used for training

Parameter Name	Parameter Value
Number of pole pairs (**p**)	**7**
Stator Resistance ($\mathbf{R_s}$)	**293mΩ**
Q- and D-axis Inductance ($\mathbf{L_d}$ and $\mathbf{L_q}$)	**77.724μH**
Magnetic Flux of the Permanent Magnet ($\boldsymbol{\psi_p}$)	**4.6mVs**
Nominal speed in RPM	**3476RPM**
Nominal Current (**I**)	**7.26A**
Nominal Voltage (**V**)	**13.86V**
Sample time ($\boldsymbol{\tau}$)	**100μs**

Table II. – General outline of the neural networks used for the RL agent

Network	Layer and Activation function
Actor	Dense with 32 neurons, ReLU
	Dense with 32 neurons, ReLU
	Dense with 2 neurons, Tanh
Critic	Dense with 32 neurons, ReLU
	Dense with 32 neurons, ReLU
	Dense with 16 neurons, ReLU
	Dense with 2 neurons, Tanh

A. *Results with the Phyton based GEM toolbox*

A conclusion from the training of the RL agent in the Phyton based GEM environment was that the number of episodes has a drastic effect on the performance. As can be seen on Fig. 4., the agent has rapidly improved after a certain total episode number. This behavior can be caused by the limit violations that GEM introduces, which can shut down an episode very early, making the performance of the agent highly fluctuating at the beginning of training. It can also be explained by the algorithm's exploration strategy, as it tends to try out a wider range of actions at the start of learning and then it narrows down its choices as it tries to converge to an optimal solution. However, it still seldomly experiments in later stages as well, to be able to move out from a given local minima, resulting in large drops in the reward value.

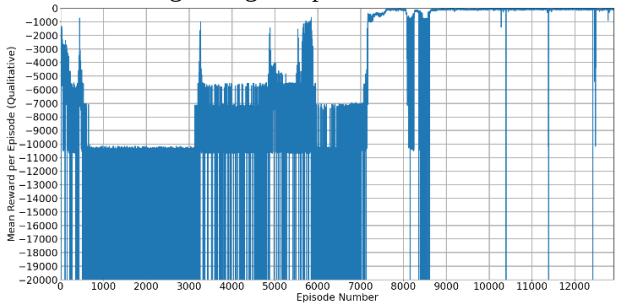

Fig. 4. Python Mean Reward trend over episode number during training

The performance of the Python agent can be seen on Fig. 5. for a load speed that varies in time which makes the agent able to experience a wider range of scenarios and increases its generalization power. The figure presents the time function of the normalized mechanical speed Ω and electric torque τ, as well as the *d* and *q* axis current signal with their respective reference signal in Ampere. The control actions, like the v_d and v_q voltage signals, as well as the time function of one of the phase currents are also given in the figure. An important point to mention here is that based on the figure, we can see that the agent does not violate the voltage limits but rather stays in the zone of safe operating conditions which is highly important in a real-life setting.

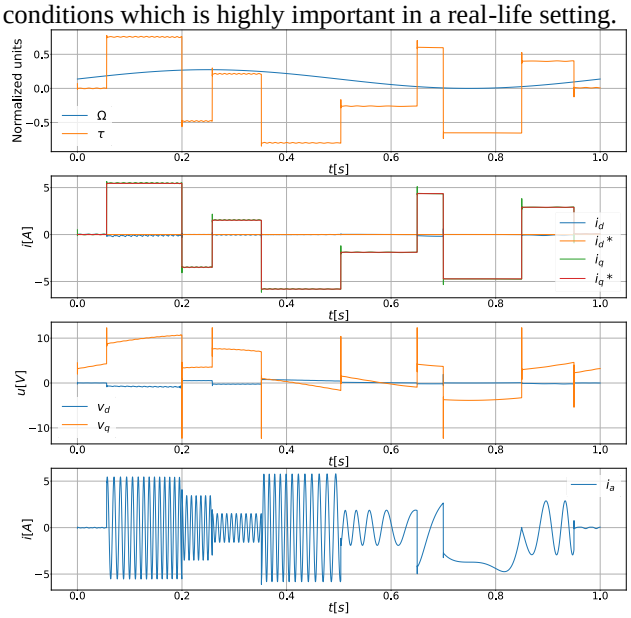

Fig. 5. Trajectories of the Python RL Agent controlled PMSM motor with a varying RPM

B. *Results with MATLAB/Simulink*

As for the MATLAB environment, the obtained results can be seen on Fig. 6. and 7., where the motor is under the same time varying load function and the goal is to follow the reference current just like in the Python experiment. Based on the reward trend, it can be seen that the MATLAB agent converged in much less episodes and achieved a stable reward curve, which can be explained by the different constraint implementation of the environment: the episode is not terminated right away when a limit is broken but rather it gives some time to the agent to correct itself, thus resulting in longer episodes and more stable performance right at the start of training. However, the main conclusion here is also the same: the RL agent managed to learn how to control the motor so that it follows the reference current closely while also staying within the safe operating limits. This behavior can be spotted when the load function is at its peak and the current does not follow the higher reference value but rather stays well within the border of the safe operating conditions. The metric values for the two experiments can be seen in Table III., where it is quite evident that the Python based agent greatly outshined the MATLAB environment. This is possible since the GEM environment generates a random reference function for each episode, making the performance of the RL agent more robust to external factors like the time varying load function.

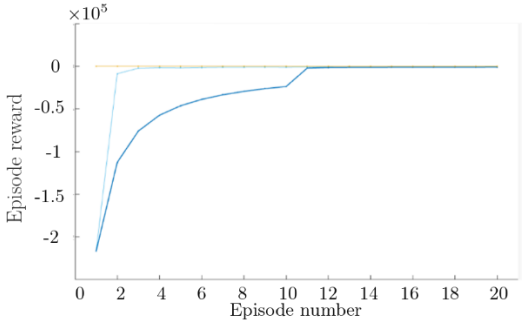

Fig. 6. MATLAB Reward trend over episode number during training

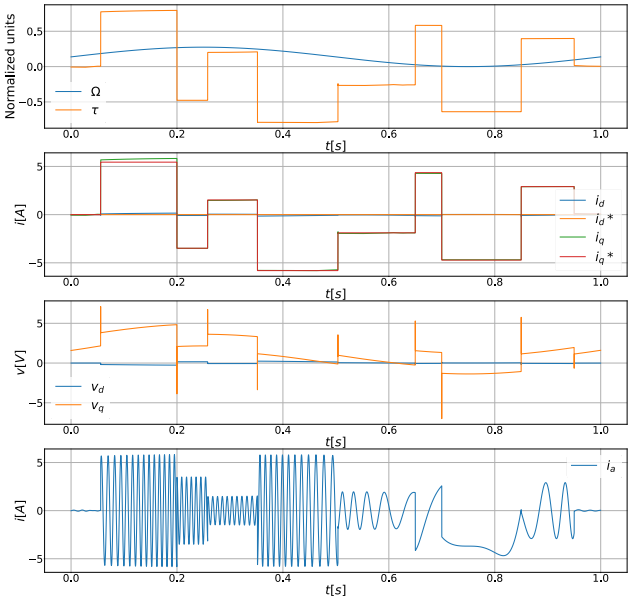

Fig. 7. Trajectories of the MATLAB RL Agent controlled PMSM motor with an outer speed control

Table III. – Performance metrics of the RL Agents for 10,000 steps

Metric	Python	MATLAB
Mean Square Root Error	11%	17%
Mean Absolute Error	0.87%	2.00%
Mean Squared Error	0.014%	0.119%

6. Comparison of the Software Environments

After introducing how the RL agent-controlled motor system was set up, the core part of this paper is addressed: the qualitative comparison of the two software environments. The summary of the comparison can be seen on Fig. 8. where the main takeaways from this experiment are collected.

The main advantage of the Python based environment is that it is open-source and it provides a wide range of software packages in Artificial Intelligence, more precisely, Deep and Reinforcement Learning. Using these resources, we can access most of the cutting-edge research done in this field and use it effectively in our experimentation. These packages also help in speeding up both the construction of the neural network architecture and the training of the agent itself, as they offer a clear workflow and parallelize a lot of operations, greatly exploiting the capabilities of a GPU. As for the modelling of the control system, our choices are a bit more limited, as often if we want to model a specific system with real-life nonlinearities, we have to code it ourselves to achieve the desired outcome. Since no graphical interface is available, we also need to visualize the system layout just by looking at the lines of code, which makes the construction of complex systems more demanding and the debugging process more time-consuming.

In the case of the MATLAB environment, the priorities are switched. The modelling is done using Simulink where there are a lot of available "building blocks" from which we can construct and parametrize a control system as the specifications require it. Here the simulation is closer to reality as the non-ideal nature of every equipment can be modelled and used during the calculation, making the transition from software to hardware much more seamless. The user-friendly graphical interface is also a huge benefit, as it makes the construction of the system quite intuitive and helps understand its functioning, making the process of identifying and correcting issues much faster. Constructing the neural networks, however, is a bit more limited as we have only access to a predefined set of tools which are only updated periodically with the MATLAB versions. If we would like to use some recent research, we would have to implement it ourselves which can be quite demanding as the original coding itself is usually done in Python. The price of MATLAB also has to be mentioned, as it is not openly available to everyone, but this also means that the toolboxes come with a detailed documentation, support background and in-depth examples to ease the development and implementation of ideas.

7. Conclusions

Based on the acquired results it is evident that the Python based environment, due to its open-source nature, has all the cutting-edge RL technology available to use, lifting its performance. In comparison, the MATLAB based environment provides a well-structured software with a user-friendly visual interface and many predefined model blocks from which a given setup can be built relatively easily. To sum up, in earlier stages of development, the Python environment serves well for trying out different algorithms, agent settings and parameter optimization. In comparison, the MATLAB based environment proves better in later stages, where the applicability and deployment are more emphasized. This conclusion suggests the workflow of training the RL agent in the Python environment on a less detailed model and then transferring it to the MATLAB environment where it is further fine-tuned on a more realistic model, which process can be the basis for further research.

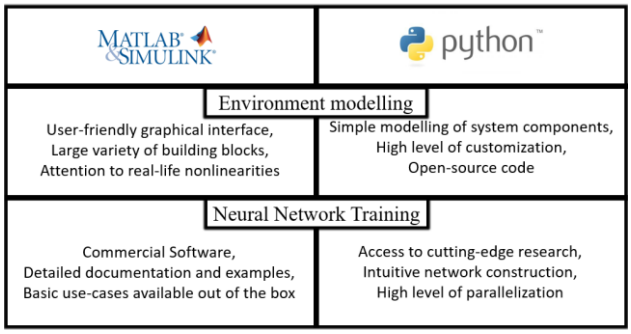

Fig. 8. Summary of the comparison between the two environments

Acknowledgements

This work was supported by the National Research, Development, and Innovation Office under Grant FK 143429.

References

[1] S. Zhao, F. Blaabjerg and H. Wang, "An Overview of Artificial Intelligence Applications for Power Electronics," in *IEEE Transactions on Power Electronics*, vol. 36, no. 4, pp. 4633-4658, April 2021, doi: 10.1109/TPEL.2020.3024914

[2] S. Zhang, O. Wallscheid and M. Porrmann, "Machine Learning for the Control and Monitoring of Electric Machine Drives: Advances and Trends," in *IEEE Open Journal of Industry Applications*, vol. 4, pp. 188-214, 2023, doi: 10.1109/OJIA.2023.3284717

[3] D. Jakobeit, M. Schenke and O. Wallscheid, "Meta-Reinforcement-Learning-Based Current Control of Permanent Magnet Synchronous Motor Drives for a Wide Range of Power Classes," in *IEEE Transactions on Power Electronics*, vol. 38, no. 7, pp. 8062-8074, July 2023, doi: 10.1109/TPEL.2023.3256424

[4] A. Traue, G. Book, W. Kirchgässner and O. Wallscheid, "Toward a Reinforcement Learning Environment Toolbox for Intelligent Electric Motor Control," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 3, pp. 919-928, March 2022, doi: 10.1109/TNNLS.2020.30295

[5] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods." in *International Conference on Machine Learning*, pp. 1587–1596. PMLR, 2018.

[6] G. Book et al., "Transferring Online Reinforcement Learning for Electric Motor Control From Simulation to Real-World

Experiments," in *IEEE Open Journal of Power Electronics*, vol. 2, pp. 187-201, 2021, doi: 10.1109/OJPEL.2021.3065877
[7] D. Weber, M. Schenke and O. Wallscheid, "Steady-State Error Compensation for Reinforcement Learning-Based Control of Power Electronic Systems," in IEEE Access, vol. 11, pp. 76524-76536, 2023, doi: 10.1109/ACCESS.2023.3297274