StableRLS: Stable Reinforcement Learning for Simulink

Robert Annuth, Finn Nußbaum and Christian Becker
Institute of Electrical Power and Energy Technology
Hamburg University of Technology
Email: robert.annuth@tuhh.de, finn.nussbaum@tuhh.de, c.becker@tuhh.de

Abstract

Today's innovative systems rely heavily on complex electronic circuitry and control. This complexity increases development, testing, and manufacturing efforts. Consequently, building the first hardware prototype is often uneconomical, leading to a preference for simulation-based approaches. MATLAB, a commercial software, facilitates rapid prototyping, mathematical analysis, and simulation through its Simulink toolbox. Additionally, MATLAB offers a reinforcement learning (RL) toolbox for integration with Simulink. However, this RL toolbox is still maturing and lacks many recent RL innovations, partly because MATLAB is not alywas the first choice for machine learning in academia and industry. Despite this, many companies and researchers leverage Simulink's powerful simulation capabilities, often developing custom simulation libraries. To bridge the gap between MATLAB/Simulink and the more widely used Python ecosystem for RL, StableRLS was developed. StableRLS enables the use of existing Simulink models for RL in Python by converting these models and offering a Python user interface compatible with the Gymnasium library.

I. Introduction

Simulation-based approaches play a crucial role in various engineering applications, enabling engineers to modify systems and observe the resulting behavioral changes. StableRLS (Stable Reinforcement Learning for Simulink) is a software framework that integrates Simulink simulation models into the Python library Gymnasium [1] to perform reinforcement learning (RL). While this paper presents the framework from an electrical engineering perspective, Simulink is also widely used in communication engineering, control systems, signal processing, robotics, driver assistance systems, and digital twins. Consequently, the framework's applicability extends beyond electrical grid simulations to these and other disciplines. StableRLS requires no modifications for use in other fields, as long as the Simulink models can be exported as C code.

Simulink is a graphical editor for modeling various system components, offering numerous pre-built blocks, algorithms, and physical systems by default. However, the RL interface in Simulink undergoes frequent modifications with each release and can exhibit inconsistent performance. In contrast, Python is extensively used for RL tasks in conjunction with Gymnasium [1], a framework that provides a standard API for communication between RL algorithms and environments. Creating simulation environments directly in Python can be challenging and error-prone, and various tools aim to simplify this process. Currently, a high-performance interface between Simulink and Python is lacking, a gap that StableRLS aims to fill.

This paper is organized as follows: first, a summary of the state of the art is provided, existing frameworks are introduced, and a brief performance comparison is presented. Next, the main features of the StableRLS framework are described, and its API is introduced. Finally, an application example from electrical engineering is provided. Additional information, examples, and a contributor's guide can be found in the StableRLS package documentation.

A. Reinforcement Learning Overview

RL is a field of machine learning that focuses on training agents to make decisions within an environment to maximize their cumulative reward. The agent interacts with the environment by performing actions that influence its state, receiving feedback as rewards or penalties for each action. The agent's goal is to learn a policy that yields the highest possible cumulative reward.

Research in RL can be divided into two main areas: developing environments for agent interaction and creating algorithms that enable agents to converge quickly to an optimal policy. Formulating algorithms broadly applicable to diverse environments presents a significant challenge. In 2016, OpenAI introduced the Gym framework [2], which defined a standard API for interaction between agents and environments. Recently, the Farama Foundation has taken over its development, maintaining the package as Gymnasium [1].

One reason for RL's growing popularity is its ability to address complex problems that are difficult to solve using traditional programming or supervised learning methods. RL excels in situations where the optimal solution is unknown and must be learned through exploration and exploitation. Applications of RL span numerous domains, including robotics, healthcare, finance, energy management, logistics, and game playing [3].

II. Statement of need

Several authors have attempted to simplify the process of creating RL environments in Python. For instance, [4] introduced a customizable Gym OpenAI environment focused on electrical networks. However, its class-based definition of the power network proves impractical for large-scale networks due to the need to define each component and its structure individually. Similar frameworks have been proposed by [5], [6], [7], but they lack extensibility to domains beyond electrical engineering and do not offer a user-friendly modeling interface.

For example, [8] proposed a framework called OMG, which is similar to StableRLS but tailored for the Modelica modeling environment [9]. Due to Modelica's steep learning curve, there is a need for a framework that utilizes a more accessible modeling environment and can be combined with Gymnasium for RL. Like the OMG framework, StableRLS utilizes Functional Mock-up Units (FMUs). An FMU is a standardized file format for exchanging and integrating simulation models across different simulation tools. It contains a dynamic model, encompassing its mathematical equations, inputs, and outputs. The StableRLS implementation uses the Functional Mock-up Interface (FMI) 2.0 standard [10], which defines the data exchange formats and structure.

III. Proposed Framework - StableRLS

StableRLS leverages MATLAB's internal capabilities to compile a Simulink model into an FMU. However, MATLAB's ability to compile FMUs is somewhat restricted and frequently requires model modifications. The effort to compile existing models can be substantial, often making it more practical to develop models specifically for FMU export. StableRLS addresses this issue by automatically modifying Simulink models and compiling them into FMUs. MATLAB requires a user-defined signal bus structure, which can be particularly error-prone for large simulation models as it demands precise definition of all nested input signals. The input to the simulation model corresponds to the agent's action structure, which often necessitates adjustments to the signal structure as the RL problem evolves. StableRLS automatically creates the bus structure for any Simulink model and works with signals connected to multiple blocks, GoTo blocks, and nested structures. Consequently, users can focus on creating the environment model in Simulink rather than on the intricacies of FMU conversion.

In developing StableRLS, different methods for combining Simulink simulation models with Python and Gymnasium were compared. Specifically, the TCP/IP interface and the MATLAB engine Python interface were evaluated. Generally, Simulink does not allow changing simulation parameters during runtime, which necessitates pausing the simulation to apply agent actions at each time step. This means the Simulink simulation must be paused for each agent interaction. This requirement applies to both interface options mentioned above, with the primary difference being the data exchange method between Simulink and Python. It was found that this start-stop interaction, involving pausing and restarting the simulation, is the primary performance bottleneck, and the data transmission time is negligible. Therefore, the performance of the MATLAB engine and the TCP/IP method is nearly identical. The table below compares the performance of StableRLS (using FMUs) with these MATLAB engine/TCP-IP based start-stop methods, demonstrating that StableRLS is over 900 times faster.

TABLE I
Performance Comparison: StableRLS (FMU-based) vs. Start-Stop Methods for RL with Simulink Models.

Method	**Computation Time**
StableRLS	0.049s
Start-stop-method	45.13s

Fig. 1 illustrates the general structure of the framework. The Simulink simulation model is converted to an FMU and integrated into StableRLS. Before the conversion can begin, the StableRLS package prepares the Simulink model, as several input and output signal definition requirements must be satisfied. The resulting FMU model is ready to run in any FMU simulator. In our case, the FMU interacts with StableRLS following the definitions of the Gymnasium API [1], thereby providing a Gymnasium-compliant environment for an RL agent. Furthermore, the structure and behavior of the StableRLS package are designed to be as flexible as possible to accommodate different RL learning strategies and use cases. Please refer to our documentation for a detailed explanation of all features.

To start the agent's training process, a configuration file is required. This file contains basic information about the environment, such as simulation duration, step size, and the FMU path. This file is used to create a child of the StableRLS base class. Additionally, any other parameters specified in this file are available within the resulting class. Functionalities have also been implemented to address specific requirements. For example, when working with external environmental data, such as weather data, it is necessary to read this data at each simulation step. Therefore, it is important to note that the simulation step size does not have to be identical to the agent's step size. As a result, the simulation could run with a step size of 0.1 seconds, while the agent chooses an action every 1.5 seconds. To integrate external data into the model, the user can simply override the 'FMU_external_input(self)' method. Other examples

of such general functionalities include data export and the definition of action and observation spaces. Internally, StableRLS is a custom environment for the Gymnasium Python package and utilizes PyFMI [11] to simulate the FMU.

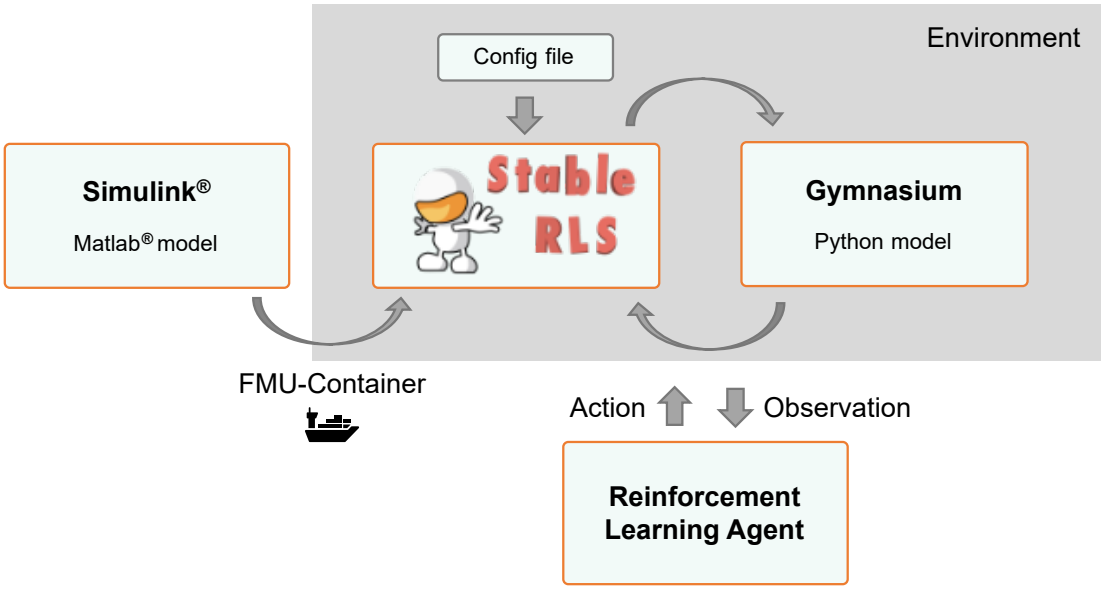

Fig. 1. General overview of the software package and how it interacts with MATLAB Simulink and the Python Gymnasium environment.

IV. Example Use Case: Electrical Microgrid

To demonstrate the usage of the StableRLS package, a small electrical microgrid with a nominal voltage of 48V was selected as the environment, controlled by an RL agent. This paper provides a brief overview of the required steps to work with the StableRLS package and run the simulation. The detailed example, including additional information, is available in the software repository.

In this example, the RL agent coordinates different electrical energy sources to supply a constant power load using the droop control concept. Specifically, a linear droop is employed, which results in a decrease in the output voltage of both the voltage source and the battery under load.

The RL agent can modify the voltage reference of the PV array and the battery relative to the nominal value. The agent's action space is two-dimensional and comprises 11 discrete values. For instance, an action value of 5 corresponds to no change in the voltage reference, while any increase or decrease in the action value will modify this reference. Not all environmental observations are visible to the agent, and the example demonstrates how to scale and process them. At each time step, the agent interacts with the simulation, observing the voltages and currents of all four components, in addition to the battery's state-of-charge. Fig. 2 shows the structure of the Simulink simulation model. A PV array generates renewable energy, which can be consumed directly by the constant power load or stored in the battery. The backup voltage source supplies any load not met by the PV array and battery. However, the energy drawn from this voltage source should be minimized.

The example in the software repository contains further details on how to configure the agent, as the FMU simulation has a step size of 1 millisecond, and the RL agent can interact with the environment every 10 seconds. Additionally, irradiance data is used for the PV array to calculate energy production. This data has a sampling time of 1 minute and must be updated in the simulation.

For simplicity, and because the training process itself is beyond the scope of the StableRLS package and this paper, the agent was run with fixed actions rather than implementing an algorithm to find optimal actions. However, it is straightforward to train an RL agent, such as a Proximal Policy Optimization (PPO) agent
citeschulman2017proximal, using the provided actions, observations, and rewards. The simulation is run for one episode, which lasts 15 minutes
citegymnasium.

For this demonstration, both action values were set to 5. As a result, the reference voltage is always equal to the nominal voltage. Therefore, the output voltages of the battery and PV array are expected to be nearly identical. However, their output voltages are not precisely identical because the battery voltage slightly deviates from the nominal 48V depending on its state of charge.

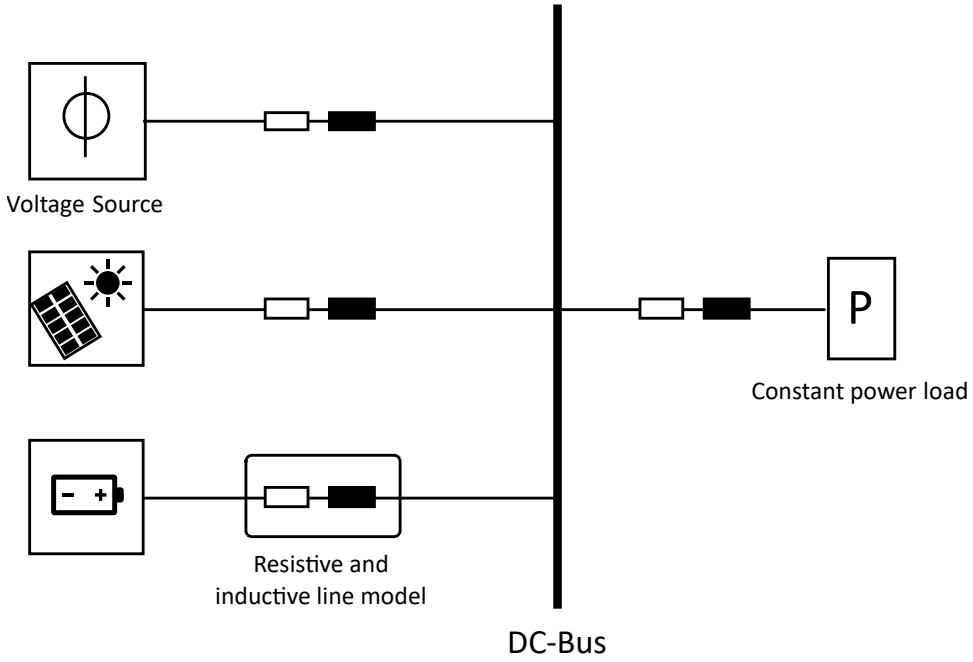

Fig. 2. Simulation model of the example use case, containing one load and three sources connected by electrical lines.

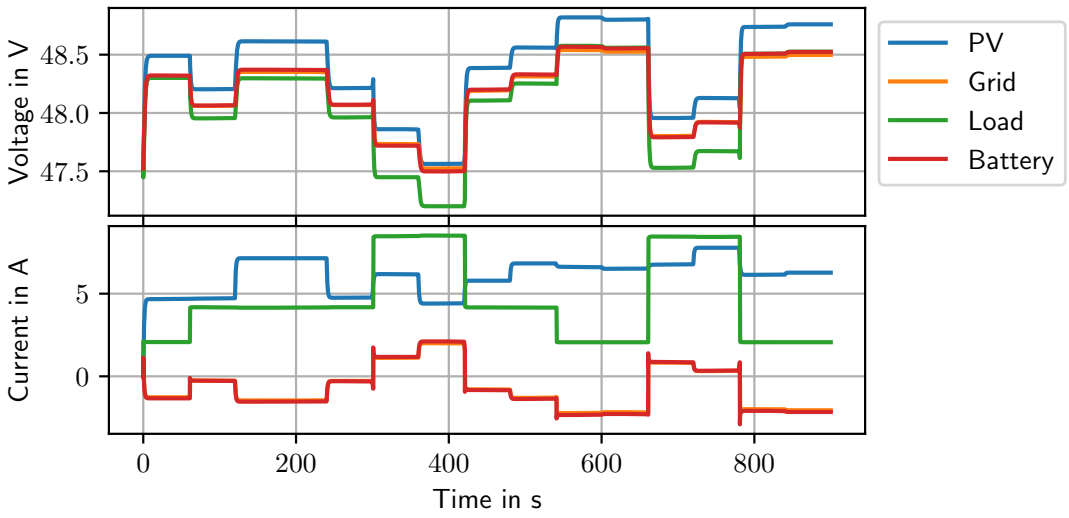

Fig. 3. Simulated voltages and currents of the four components within the simulated electrical grid.

Fig. 3 shows the simulation results. Since the agent maintains a constant action without changing the voltage reference, the voltages of the battery and the PV array (and consequently the grid voltage tied by droop) are nearly identical, as expected. The power from the PV array fluctuates relative to the load power. When the PV power is lower than the load, additional power from the grid (backup voltage source) and the battery is used. This power is shared almost equally because the droop reference voltage is not modified in this example, and the droop curves are identical. If the PV power exceeds the load demand, the surplus power is fed back into the grid and the battery. During battery charging and discharging, the voltage deviates slightly from the nominal voltage, leading to small differences between the currents and voltages of the grid and the battery. Lastly, the droop behavior is also visible in the voltage plot. At high loads, the output voltage of all components decreases according to their droop coefficients.

V. Conclusion

In academia and industry, AI methods are increasingly being tested and validated for their suitability in specific use cases. To verify the applicability of RL, software packages such as StableRLS are particularly relevant as they bridge existing simulation models and newly developed RL algorithms. The easy integration of Simulink models,

without the significant overhead typically associated with model export and Python coupling, allows users to focus on improving the environment model and adapting or extending existing RL algorithms. Additionally, this framework has implications beyond electrical engineering, its initial domain of application. The framework can also be used for other research disciplines without any adaptations, provided a suitable Simulink model exists for agent interaction. Further development will focus on providing a more detailed gallery of use cases beyond electrical engineering to broaden the framework's visibility and adoption.

Index Terms

Reinforcement Learning, FMU, MATLAB/Simulink, Python

References

[1] *Gymnasium documentation*, 2023. [Online]. Available: https://gymnasium.farama.org/.
[2] G. Brockman et al., *Openai gym*, 2016. eprint: arXiv:1606.01540.
[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
[4] R. Henry and D. Ernst, "Gym-anm: Reinforcement learning environments for active network management tasks in electricity distribution systems," *Energy and AI*, vol. 5, p. 100 092, 2021. DOI: 10.1016/j.egyai.2021.100092.
[5] A. Marot et al., "Learning to run a power network challenge: A retrospective analysis," in *NeurIPS 2020 Competition and Demonstration Track*, PMLR, 2021, pp. 112–132.
[6] T.-H. Fan, X. Y. Lee, and Y. Wang, "Powergym: A reinforcement learning environment for volt-var control in power distribution systems," in *Learning for Dynamics and Control Conference*, PMLR, 2022, pp. 21–33.
[7] G. Henri, T. Levent, A. Halev, R. Alami, and P. Cordier, "Pymgrid: An open-source python microgrid simulator for applied artificial intelligence research," *arXiv preprint arXiv:2011.08004*, 2020.
[8] S. Heid, D. Weber, H. Bode, E. Hüllermeier, and O. Wallscheid, "Omg: A scalable and flexible simulation and testing environment toolbox for intelligent microgrid control," *Journal of Open Source Software*, vol. 5, no. 54, p. 2435, 2020. DOI: 10.21105/joss.02435.
[9] P. Fritzson et al., "The openmodelica integrated modeling, simulation and optimization environment," in *Proceedings of the 1st American Modelica Conference*, Modelica Association Linköping, Sweden, 2018, pp. 8–10.
[10] Modelica Association, *Functional Mock-up Interface*, 2020. [Online]. Available: https://fmi-standard.org/.
[11] C. Andersson, J. Åkesson, and C. Führer, *Pyfmi: A python package for simulation of coupled dynamic models with the functional mock-up interface*. Centre for Mathematical Sciences, Lund University Lund, Sweden, 2016.