


StableRLS: Stable Reinforcement Learning for Simulink

Robert Annuth¹, Finn Nussbaum ¹, and Christian Becker¹

¹ Technical University Hamburg, Germany

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Open Journals](#) 

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

License

Authors of papers retain copyright,
and release the work under a
Creative Commons Attribution 4.0
International License ([CC BY 4.0](#)).

Abstract

Today's innovative systems rely heavily on complex electronic circuitry and control. This complexity is associated with additional development, test and manufacturing efforts. As a result, it is often not economical to build the first prototype of a system in hardware and instead simulation-based approaches are chosen. MATLAB ([Inc., 2022](#)) is a commercial software that enables companies to perform quick prototyping, mathematical analysis and provides the possibility to perform simulations with the Simulink toolbox. Also, a reinforcement learning (RL) toolbox for MATLAB and Simulink is available. However, the toolbox is heavily under development and many recent RL innovations are not available since MATLAB is often not the preferred choice for machine learning in academia and industry. Nevertheless, many companies and researchers use the powerful simulation capabilities of Simulink to build their own simulation libraries. StableRLS allows them to use these existing Simulink models to perform RL in Python. It handles the conversion of the model and provides a Python user interface based on the Gymnasium library for RL.

Introduction

Simulation-based approaches are actively used in various engineering applications. They make it possible to modify systems and immediately observe the resulting changes in behavior. StableRLS (Stable Reinforcement Learning for Simulink) is a software framework that integrates Simulink simulation models into the Python library Gymnasium ([Gymnasium Documentation, 2023](#)) to perform reinforcement learning (RL). In this paper the framework is presented from the perspective of electrical engineering, but Simulink is also used in many other disciplines, including communication engineering, control, signal processing, robotics, driver assistance and digital twins. Therefore, this framework is relevant not only to electrical grids simulation, but also to other disciplines. StableRLS does not require any modifications to be used for any other disciplines, as long as the Simulink models can be exported as C code, which is usually the case.

Simulink is a graphical editor for modeling various components of a system, and it provides many prebuilt blocks, algorithms and physical systems by default. However, the RL interface is modified at each release and sometimes does not behave as expected. Instead, Python is often used for RL tasks in combination with Gymnasium ([Gymnasium Documentation, 2023](#)), which is a framework providing a standard API to communicate between RL algorithms and environments. The environment is usually a simulation and often difficult to create directly in Python. Various tools try to simplify this difficult and error-prone process. Currently, there is no high-performance interface between Simulink and Python is available, highlighting out the need for StableRLS. The paper is organized as follows.

First the state of the art is summarized, existing frameworks are introduced and a short performance comparison is presented. Then, the main features of the framework are presented,

and the API is introduced. Last, we showcase an example from the domain of electrical engineering. Additional information, examples, and a guide for contributions, can be found in the documentation of the StableRLS package.

Reinforcement learning overview

RL is a field of the machine learning domain that focuses on training one or more agents to make decisions in an environment to maximize their cumulative reward. The agent repeatedly interacts with the environment by performing actions, thereby influencing the state of the environment. The agent receives feedback in the form of rewards or penalties for each action. The goal of the agent is to learn a policy that leads to the highest possible cumulative reward. The research interests can be divided into developing environments that the agent can interact with, and developing algorithms for the agent to quickly converge to an optimal solution by means of the policy. In particular, formulating algorithms that are applicable to a vast set of environments is challenging. OpenAI developed a framework in 2016 called gym ([Brockman et al., 2016](#)) that defines the API interface between the agent and the environment. Recently, OpenAI stopped the development and the Farama Foundation took over the development by maintaining a package called Gymnasium ([Gymnasium Documentation, 2023](#)). One of the reasons for RL's growing popularity is its ability to tackle complex problems that are difficult to solve using traditional programming or supervised learning methods. RL excels in situations where the optimal solution is not known in advance, and the agent needs to learn through exploration and exploitation. For example, it has found applications in robotics, healthcare, finance, energy management, logistics, game playing and many other domains. ([Sutton & Barto, 2018](#))

State of the art

In the literature, other authors have already tried to simplify the process of creating environments for RL in Python. In ([Henry & Ernst, 2021](#)) the authors introduced a customizable Gym OpenAI environment, but it is focused on electrical networks and the class-based definition of the power network is not reasonable for large network since each component and structure has to be defined separately. The authors in ([Marot et al., 2021](#)), ([Fan et al., 2022](#)) and ([Henri et al., 2020](#)) have proposed similar frameworks, but all of them lack the extensibility to other domains besides electrical engineering and don't include an easy-to-use modeling interface. For example, in ([Heid et al., 2020](#)) the authors proposed a framework called OMG which is similar to StableRLS for the Modelica modeling environment ([Fritzson et al., 2018](#)). However, due to the steep learning curve of Modelica, there is a need for a framework that relies on another modeling environment that can be combined with Gymnasium to perform RL. Similar to the OMG framework, StableRLS uses Functional Mock-up Units (FMUs). A FMU is a standardized file format to exchange and integrate simulation models across different simulation tools. An FMU contains a dynamic model combined with mathematical equations and has inputs and outputs. The implementation of The StableRLS implementation uses the Functional Mock-up Interface (FMI) 2.0 standard ([Modelica Association, 2020](#)), which defines the data exchange formats and the structure.

Proposed framework - StableRLS

StableRLS uses the internal capabilities of MATLAB to compile a Simulink model to a FMU. However, the capabilities of MATLAB to compile FMUs are highly limited and require model modification. In general, the effort to compile an existing model is so high that it often makes more sense to start from scratch. StableRLS solves this issue by modifying Simulink models automatically and compile those models to a FMU. MATLAB requires a user defined signal bus structure and especially for large simulation models the definition is error-prone because all nested input signals have to be defined correctly and since the input to the simulation model

correlates to the action structure of the agent it is often required to modify the signal structure while working on a RL problem. StableRLS creates the bus structure for any Simulink model and also works with signals connected to multiple blocks, GoTo blocks and nested structures. As a result, the user only has to create the environment model in Simulink and don't need to worry about the conversion of the model to a FMU.

While working on StableRLS we also put a lot of effort into comparing different methods to combine Simulink simulation models and Python Gymnasium. Namely, the TCP/IP interface and the implemented Python interface which is called MATLAB engine. In general, Simulink doesn't allow changing simulation parameters during the simulation, so the simulation has to wait for the actions of the agent at each time step. As a result, it is required to pause the Simulink simulation between each interaction of the agent. This is the case for both interface options mentioned above. The only difference is how the data between the Simulink simulation and Python is exchanged. We found that pausing and restarting (start-stop-method) the simulation mainly determines the resulting performance, and the time for data transmission can be neglected. Therefore, the performance of the MATLAB engine and the TCP/IP method is almost identical and only the performance of StableRLS package and the internal Python interface is compared in the table below. It can be seen that StableRLS is above 900 times faster because it uses the benefits of FMUs.

Table 1: Caption Performance comparison between this package and the common alternative. The simulation time was 500 seconds, with a step size of 0.2 seconds.

	Simulation Time
StableRLS	0.049s
Start-stop-method	45.13s

Figure 1 represents the general structure of the framework. The Simulink simulation model is converted to a FMU and integrated in StableRLS. Before the conversion can start, the StableRLS package prepares the Simulink model since many requirements regarding the input and output signal definition must be met. 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 ([Gymnasium Documentation, 2023](#)) to provide an environment for an RL agent. Nevertheless, the structure and behavior of the StableRLS package is as flexible as possible to account for different RL learning strategies and use cases. Please refer to our documentation for a detailed explanation of all features.

To start the training process of the agent a config file must be specified containing the basic information about the environment, e.g. simulation duration, step time, path of the FMU. This file is used to create a child of the StableRLS base class. In addition, any other parameters specified in this file are available within the resulting class. Finally, we implemented functionalities to account for specific requirements. For example when working with external data for the environment, like weather data, it is necessary to read this between every simulation step. Therefore, it is important to note that the simulation step size doesn't have to be identical with the step size of the agent. As a result, the simulation could run with a step time of 0.1 seconds but the agent only chooses an action every 1.5 seconds. To integrate external data into the model, the user can simply overwrite the existing function "FMU_external_input(self)". Other examples for such general functionalities are the export of data and definition of action and observation spaces. Internally StableRLS is a custom environment for the Gymnasium Python package and exploits the functionalities of PyFMI ([Andersson et al., 2016](#)) to simulate the FMU.

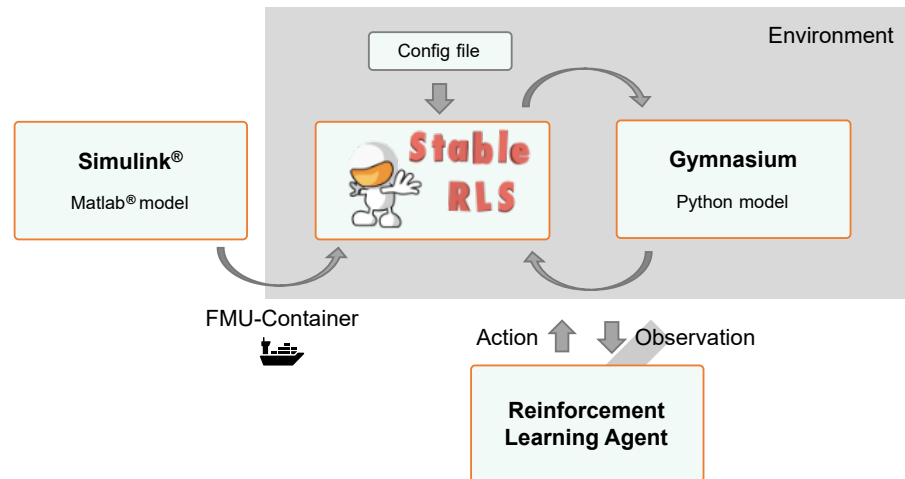


Figure 1: General overview of the software package and how it interacts with MATLAB Simulink and the Python Gymnasium environment.

Example use case: Electrical microgrid

To demonstrate the usage of the package, we chose a small electrical grid with a nominal voltage of 48V as an environment which is controlled by an RL agent. In this paper, we only provide a rough overview about the required steps to work with the StableRLS package and run the simulation. The example is also included in the software repository with additional information. It is demonstrated how the RL agent coordinates different electrical energy sources to feed a load with constant power by using the droop concept. In this case we use a linear droop which results in a decrease in the output voltage of the voltage source and battery.

The RL agent can modify the voltage reference of the PV-array and the battery relative to nominal value. The action space of the agent is two-dimensional and has 11 discrete values to choose from. E.g. an action with the value 5 refers to not changing the voltage reference and any increase or decrease of the action value will modify this reference. Not all observations of the environment are visible to the agent, and the example also shows how to scale and process them. At each time step the agent interacts with the simulation it observes the voltages and currents of all four components and in addition the state-of-charge of the battery.

Figure 2 shows the structure of the Simulink simulation model. A PV-array is used to generate renewable energy which can be consumed directly by the constant power load or can be stored in the battery. The amount of load which is not covered by those two sources is fed by a backup voltage source. However, the energy from this voltage source should be as small as possible.

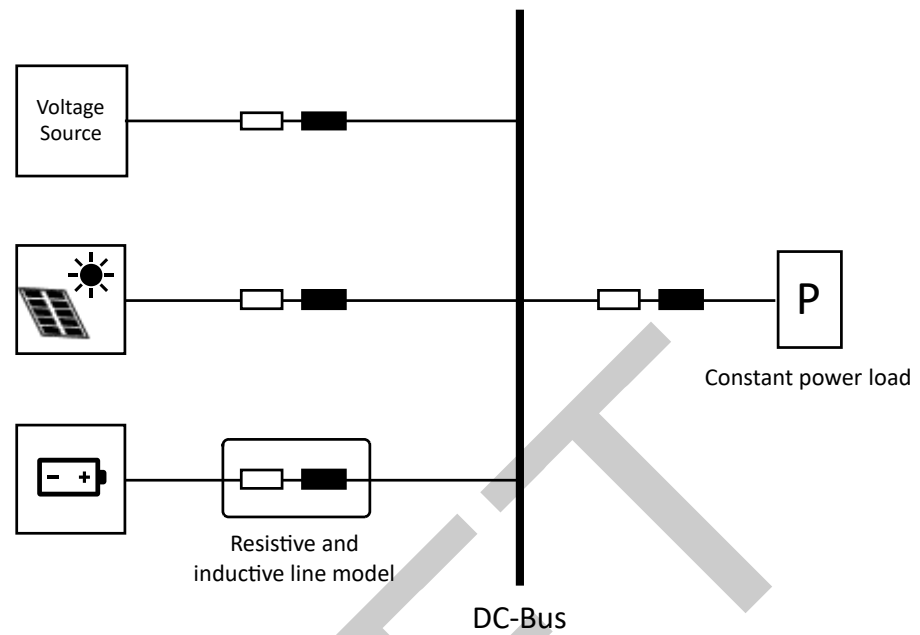


Figure 2: Simulation model of the example use case, containing one load and three sources coupled by electrical lines.

151 The example in the software repository contains further details how to configure the agent
152 because the FMU simulation has a step time of 1 millisecond and the RL agent can interact
153 with the environment every 10 seconds. Also, irradiance data is used for the PV-array to
154 calculate the energy production. This data has a sampling time of 1 min and has to be
155 updated in the simulation. Additionally, to the actions of the agent, the load, irradiance and
156 temperature of the PV-array are set internally every simulation step. However, as already
157 mentioned, those are not observed by the agent. Figure 3 shows the load data and also the
158 irradiance data of the PV-array.

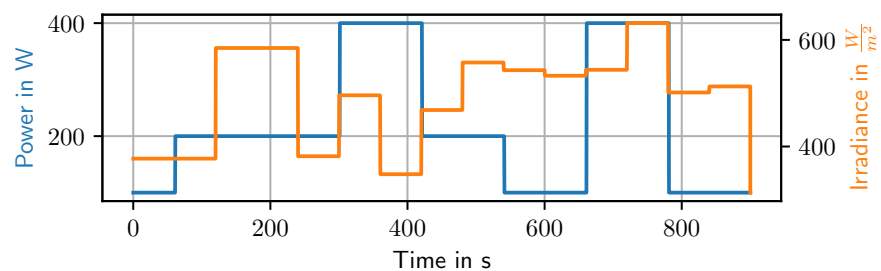


Figure 3: Power of the constant power load and irradiance of the PV-array used for the simulation.

159 For simplicity reasons and because the training itself goes beyond the scope of the StableRLS
160 package and this paper, we decided to run the agent with fixed actions instead of choosing
161 an algorithm to find the optimal actions. However, this is easily possible by training e.g. an
162 PPO agent (Schulman et al., 2017) with each action, observation and reward. We run the
163 simulation for one episode, which is 15 minutes long. (Gymnasium Documentation, 2023)

164 For this demonstration, we chose for both action values 5. As a result, the reference voltage
165 is always equal to the voltage nominal voltage. So, we would expect almost identical output
166 voltages of the battery and PV-array. The output voltage is not identical, since the battery
167 voltage deviates slightly from the nominal voltage of 48V in dependence of the SOC.

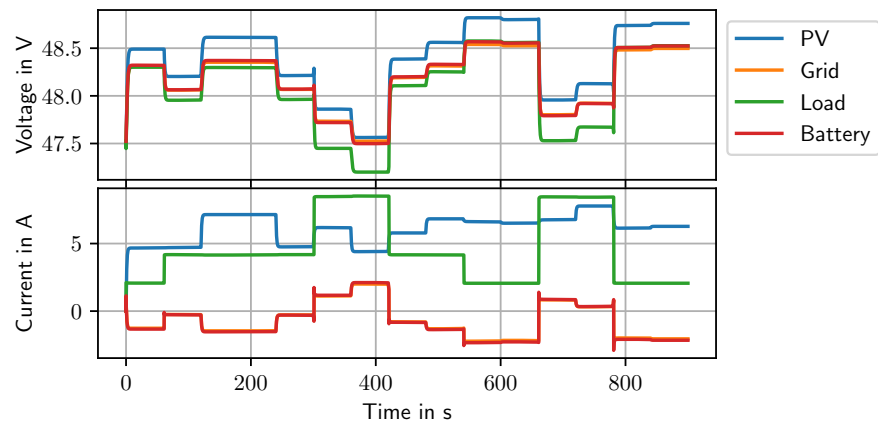


Figure 4: Voltages and currents of the four components within the simulated electrical grid.

Figure 4 shows the results from the simulation. Because the agent doesn't modify the voltage reference by choosing a constant action, the voltage of the battery and voltage source are also almost identical as expected. The power of the PV-array oscillates around the power of the load. If the power from the PV-array is lower compared to the load, additional power from the grid and the battery is used. The power is shared almost identical since, the droop reference voltage is not modified in this example and the droop curve is identical. If the PV-power exceeds the load demand, the additional power is feed back into the grid a the battery. While charging and discharging the battery the voltage slightly deviates from the normal voltage leading to a small difference between the current and voltage of the voltage source and the battery. Last, the droop behavior is also visible within the voltage plot. At high loads, the output voltage of all components decreases in respect to the droop coefficient.

Conclusion

In academia and also in industry, AI methods are increasingly tested and validated for their suitability for suit specific use cases. To verify the applicability of RL, software packages like StableRLS are highly relevant because they provide an interface between existing simulation models and newly developed RL algorithms. The easy integration of Simulink models without additional effort to export the model or couple it to Python, allows users to focus on improving the environment model and also to adapt or extend existing RL algorithms. In addition, this framework implications beyond electrical engineering, for which it was originally developed. The framework can also be used for other research disciplines without any adaptations, as long as a Simulink model is available for the agent to interact with. Further development will focus on providing a more detailed gallery of use cases beyond the domain of electrical engineering to increase the visibility of the framework.

Andersson, C., Åkesson, J., & Führer, C. (2016). *Pyfmi: A python package for simulation of coupled dynamic models with the functional mock-up interface*. Centre for Mathematical Sciences, Lund University Lund, Sweden.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *OpenAI gym*.

Fan, T.-H., Lee, X. Y., & Wang, Y. (2022). Powergym: A reinforcement learning environment for volt-var control in power distribution systems. *Learning for Dynamics and Control Conference*, 21–33.

Fritzson, P., Pop, A., Asghar, A., Bachmann, B., Braun, W., Braun, R., Buffoni, L., Casella, F., Castro, R., Danós, A., & others. (2018). The OpenModelica integrated modeling,

- simulation and optimization environment. *Proceedings of the 1st American Modelica Conference*, 8–10.
- Gymnasium documentation*. (2023). Farama Foundation. <https://gymnasium.farama.org/>
- Heid, S., Weber, D., Bode, H., Hüllermeier, E., & Wallscheid, O. (2020). OMG: A scalable and flexible simulation and testing environment toolbox for intelligent microgrid control. *Journal of Open Source Software*, 5(54), 2435.
- Henri, G., Levent, T., Halev, A., Alami, R., & Cordier, P. (2020). Pymgrid: An open-source python microgrid simulator for applied artificial intelligence research. *arXiv Preprint arXiv:2011.08004*.
- Henry, R., & Ernst, D. (2021). Gym-ANM: Reinforcement learning environments for active network management tasks in electricity distribution systems. *Energy and AI*, 5, 100092.
- Inc., T. M. (2022). *MATLAB version: 9.13.0 (R2022b)*. The MathWorks Inc. <https://www.mathworks.com>
- Marot, A., Donnot, B., Dulac-Arnold, G., Kelly, A., O'Sullivan, A., Viebahn, J., Awad, M., Guyon, I., Panciatici, P., & Romero, C. (2021). Learning to run a power network challenge: A retrospective analysis. *NeurIPS 2020 Competition and Demonstration Track*, 112–132.
- Modelica Association. (2020). *Functional Mock-up Interface*. <https://fmi-standard.org/>
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv Preprint arXiv:1707.06347*.
- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

DRAFT