Dymola-Enabled Reinforcement Learning for Real-time Generator Set-point Optimization

Aisling Pigott, Kyri Baker
Civil, Environmental, and Architectural Engineering
University of Colorado Boulder
Boulder, CO, USA
{aisling.pigott, kyri.baker}@colorado.edu

Sergio A. Dorado-Rojas, Luigi Vanfretti
Electrical, Computer, and Systems Engineering
Rensselaer Polytechnic Institute
Troy, NY, USA
{dorads, vanfrl}@rpi.edu

***Abstract*—This paper introduces a reinforcement learning framework which determines real-time generator set-points in a dynamically changing environment in order to optimize a chosen objective. This is performed within the high-fidelity simulation environment Dymola, which utilizes the Modelica programming language to model complex systems. A case study is created using the OpenIPSL IEEE 9-bus dynamic model, with the objective of minimizing voltage deviations across the network. The reinforcement learning agent shows improvement in minimizing voltage deviations versus the default droop controlled governors without any explicit knowledge of the topology of the system or relative location of the controllers. The results indicate that reinforcement learning may be a useful tool for applications in model-free, real-time power systems dynamics and control. An open-source Python package is provided for the proposed framework with the present case-study as an example.**

***Index Terms*—Modelica, voltage regulation, unsupervised learning**

I. Introduction

With growing uncertainty due to new electrical demands and higher integration of intermittent and variable generation, matching supply and demand in power systems has become increasingly challenging. Computing generator set-points from solving a traditional optimization problem generally requires a known model of the power grid, and cannot be performed in real-time due to computational challenges. Real-time generator control, on the other hand, is typically performed via fast but suboptimal governor control or by using automatic generation control (AGC) heuristics. However, real-time control approaches such as reinforcement learning (RL) can help generators pursue optimal set-points on a second or sub-second level without explicitly solving an expensive optimization problem. In this paper, we show how RL, in conjunction with the dynamic modeling and simulation environment Dymola, can optimize generator set-points in a dynamically changing environment while pursuing an overarching objective.

Reinforcement learning is a model-free decision making framework that is successful in coordinating complex control tasks. One of the greatest advantages of RL is the ability to improve control performance in tasks that are hard to model

This material is based upon work supported by the U.S. Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) under the Advanced Manufacturing Office, Award Number DE-EE0009139.

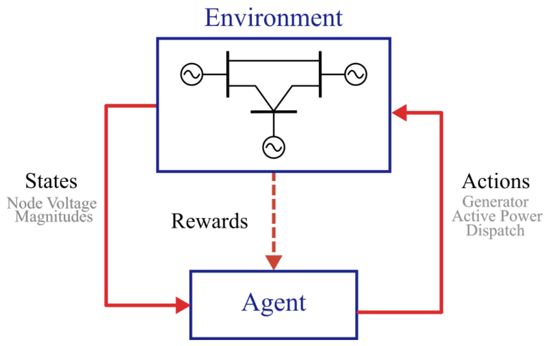

Fig. 1: RL Framework for Set-point Optimization

with conventional optimization approaches. As shown in Fig. 1, RL essentially seeks an optimal set of actions, or *policy*, that maximizes the *reward* received by an *agent* interacting with an *environment*. The *states* represent the feedback from the environment to the agent. The policy corresponds to the mapping from states to actions.

RL techniques have mostly been used in the power systems domain for offline energy optimization and control [1], [2]. RL with applications to power system stability and automatic generation control (AGC) have also shown great promise [3]–[5]. However, most of these real-time control applications do not consider high-fidelity power systems dynamics such as those modeled in the OpenIPSL library within Dymola, and often only consider stability, rather than pursuing optimality. Practical RL techniques that use realistic models are necessary to address both stability and efficiency. In this paper, we consider the application of minimizing voltage deviations across the network by adjusting generation set-points on a second-level timescale in order to minimize a chosen objective. A full nonlinear phasor-domain simulation model is used as an environment to overcome the explicit formulation of dynamical constraints in the short-term dispatch problem. By including a dynamical simulation model, the response of the grid is accounted by measuring the states in the RL setup.

Multiple modeling tools are available to expand system representation beyond the steady-state. In this work, we leverage the Modelica language, and the OpenIPSL library, suitable for power system dynamic modeling [6]. Given its equation-based

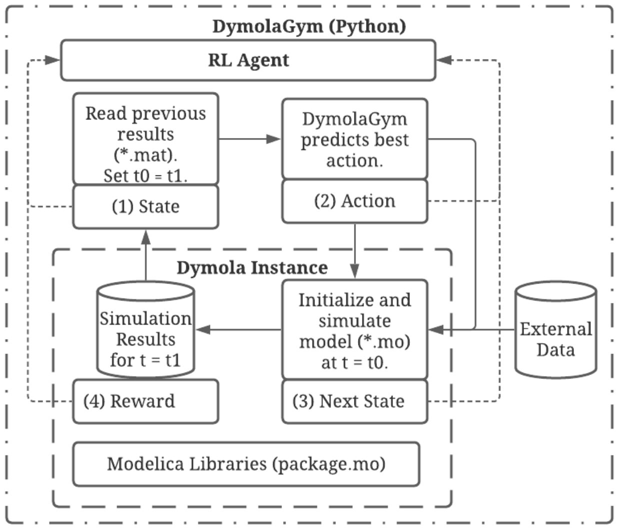
DymolaGym (Python)
RL Agent
Read previous results (*.mat). Set t0 = t1.
DymolaGym predicts best action.
(1) State
(2) Action
Dymola Instance
Simulation Results for t = t1
Initialize and simulate model (*.mo) at t = t0.
External Data
(4) Reward
(3) Next State
Modelica Libraries (package.mo)

Fig. 2: DymolaGym simulation cycle

nature, the use of Modelica language provides a precise model of the dynamical phenomena, such as machine nonlinearities and control discontinuities.

Previous works have used the Modelica language to model environments for RL with great success. These works include [7] and [8] for use with Functional Mockup Units (FMUs). However, in this paper we introduce a method for use with the Dymola software's Python-based Advanced Programming Interface (API) and power system Modelica models built using the OpenIPSL Library. Unlike the FMUs used for modeling environment behavior in [7] and [8], simulating models with a compiler API ensures that the model is as accurate as possible (acausal relationships are maintained) and speeds up performance [9]. Additionally, Dymola has additionally been shown to have improved performance compared to OpenModelica when using the `dassl` solver [10] which is a significant booster for training in any RL problem.

The rest of this paper is organized as follows: in Section II we lay out the proposed framework for Dymola enabled reinforcement learning; in Section III we introduce a sample environment using the IEEE 9-bus network; in Section IV we demonstrate the effectiveness of the RL agent in voltage control and in Section V we conclude the paper.

II. Reinforcement Learning with Dymola

DymolaGym extends the OpenAI gym environment and adds the functionality of the Dymola API to ModelicaGym environments [11]. The OpenAI gym ensures that RL agent methods can be benchmarked in an environment for direct comparison of the RL agents' performance [12]. Analogous to the OpenAI gym environment, DymolaGym requires users to specify *state* values that accurately describe the environment and *action* values that are used to influence the environment. Any value declared as a `parameter` in Modelica (e.g. such as the value of a controller's set-point) may be used in DymolaGym as a control action; any value that is assigned to a `variable` in Modelica may be used in DymolaGym as a "state" value (e.g. typical variables in power system models such as algebraic ones (voltage magnitude) or dynamic states (generator angles or internal control states)). Here we use the measured voltage at all 9 buses, the load at all 3 load buses, and the current output of all 3 generators as "state" values.

Reinforcement learning is a machine learning algorithm that works to model agent-environment interactions and improve the agent's control actions. Most importantly, RL works even when the equations (Eq. (1)) that govern the relationship between state and action (x^t, u^t) and the next state (x^{t+1}) are unknown to the agent. In this case the Dymola model acts as our transition function.

$$x^{t+1} = h(x^t, u^t) \tag{1}$$

At each time-step, the RL agent collects information about the current *state* and attempts to select an action that exploits information that the RL agent already knows about the environment while collecting new information. The trade-off between exploration and exploitation is a key element of reinforcement learning. Too little exploration causes the RL agent to get stuck in a local maxima; too much exploration hampers performance. The selected action should maximize not only the current observed reward, but the potential reward over an infinite time horizon. Let this be known as the q-value:

$$q = r(x^t) + \gamma V(x^{t+1}) \tag{2}$$

$$V(x^t) = \mathbb{E}_{u^* \sim \pi}\left(Q(x^t, u^*) - \log(\pi(u^*|x^t))\right) \tag{3}$$

where the *value*, $V(x^{t+1})$, represents expected future rewards according to the probability π of selecting an action and the predicted q-value resulting from the action selected. Since the q-value cannot be observed for every successive state, it must be predicted based on previous observations and their state-action pairs. The Q-function uses back-propagation of neural networks to minimize the prediction error of q.

The q-value represents the critic portion of the actor-critic network, and informs the RL agent of the current performance. The actor portion of the RL agent consists of a neural network for the mean and standard deviation of a Gaussian action distribution. The actor's purpose is to maximize the probability of selecting an advantageous action from any state. (Actions are classified as advantageous when the observed q-value is greater than the expected $value$ of the state.) The actor does this by minimizing the Kullback-Leiber divergence of probability of selecting an action and its anticipated advantage.

A. Deep Reinforcement Learning

Deep reinforcement learning (DRL) replaces the function approximators of q, v, σ, and μ with deep neural networks. In this paper each network uses the Multilayer Perceptron (MLP) architecture and ReLU activation functions.

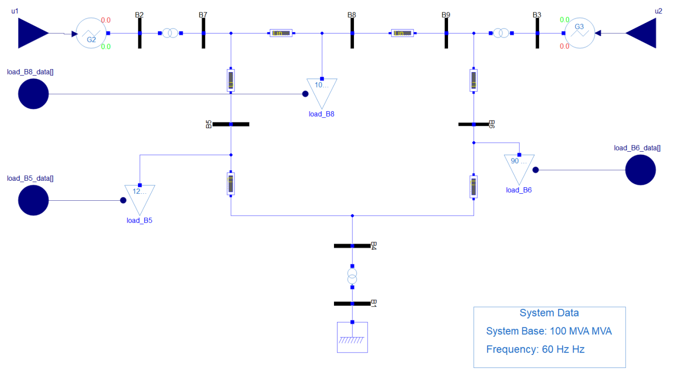

Fig. 3: Modified OpenIPSL IEEE 9-bus network

B. *Soft Actor Critic*

The variation of RL used in this paper, Soft Actor Critic, is a variant of deep RL that additionally introduces the following components to improve performance [13]:

1) Batch Optimization and Experience Replay: Batch optimization selects random data points from a memory buffer and updates the function approximations with an error gradient. Experience replay describes repeated use of data points by replacing them in the memory bank and increases sample efficiency by reusing data points. Experience replay also promotes stability in the RL agent's behavior and convergence of the function approximators by reevaluating old data.

2) Entropy Maximization: promotes exploration by artificially increasing the reward when the Gaussian distribution for action selection has a high variance. The augmented reward function is given as the sum of the observed environment reward and the entropy of the action under the current policy:

$$r'(x^t, u^t) = r(x^t, u^t) + \mathcal{H}(\pi(u^t|x^t)) \tag{4}$$

3) Dual Q-Networks: promote stability by creating two Q-networks. The networks are updated alternately, but they are evaluated simultaneously at each step. Since the Q-function approximation has been shown to be an overestimate of q [14], at each time step the minimum predicted value is used.

4) Actor-Critic Networks: Separate actor and critic networks promote stability by tracking action advantages and state expected values separately.

III. Case Study

Our case study is the IEEE 9-bus, an interconnected transmission network consisting of three generators and 3 loads.

A. *Power Flow Environment*

The OpenIPSL library is an open source Modelica library for power systems components and includes a sample application of the IEEE 9-bus system [6]. For this paper we use a modified version of the OpenIPSL implementation of the IEEE 9-bus network shown in Figure 3. As our implementation uses unconstrained reinforcement learning to perform voltage regulation, we introduce a slack bus at Bus 1 in the network to ensure that the system is balanced with a feasible power flow solution at all times and to establish the reference frequency of the grid. Additionally, time-varying loads are introduced at each of the 3 load buses and faults are removed.

The loads for this network are assumed to be uncontrollable. While the demand is inelastic, since the loads are modeled as ZIP loads, the real power served may vary slightly depending on the voltage of the bus.

B. *Action Space*

The RL agent is trained to perform two actions corresponding to the two conventional generators in network (located at Bus 2 and 3 respectively). These actions influence the droop controlled governor by manipulating the set-point difference at each simulation interval. Effectively, positive actions by the RL agent correspond to reducing the generation set-point, and negative actions correspond to increasing the set-point. When the RL agent ramps down the sum of generation by the controllable generators, the slack bus is ramped up to compensate. In order to reduce reliance on the slack bus an inertia constant ($H = 1$ s) is introduced. When the slack bus has *no* inertia a trivial solution to balancing the network is to rely as much as possible on the infinitely responsive bus.

To be conservative with the RL controller, the RL agent is limited to manipulating the set-point difference to a percentage of the maximum power output. (Two cases are presented in the results with ± 10 MW and ± 50 MW control, respectively.)

C. *Observation and Reward*

At each time-step the RL agent collects data about the environment that correlates the RL agent actions with the achieved reward. For this case study, the RL agent receives the per unit voltage at each bus, the real power output of each generator, and the real power consumption at each load bus.

Let v_i^t be the measured per unit voltage of the i-th bus at time t. The reward is then given as the deviation from 1 pu across all buses:

$$r(x^t) = \sum_{i=1}^{9} (v_i^t - 1)^2. \tag{5}$$

Note that the present framework is generalizable, and more or less information could be given to the RL agent depending on the setting, or an alternative reward function could be considered. In order to take advantage of the default training parameters in the Stable Baselines package, the action, state values, and rewards are normalized to $[-1, 1]$ in the environment. In this study we clip the voltage at each bus to $[0.9, 1.1]$, corresponding to the normalization range of $[-1, 1]$. Although the voltage could reach outside of those bounds, anything outside of those bounds would be non-operational and should be penalized to the fullest extent of the reward function. By clipping the observed voltages we also increase the granularity of observed values around the area of interest at 1 pu. The reward is normalized to the mean, max, and min observed values during a short trial period.

TABLE I: Hardware and software characteristics

RAM	64 GB
Processor	Intel(R) Xeon(R) W-2235 CPU @ 3.80GHz
GPU	AMD Radeon Pro W5500 (8GB)
Dymola Distribution	Dymola 2021x
Dymola Compiler	MinGW CC
Python Version	3.9.2
OpenAI Gym	0.18.0
Stable-Baselines3	1.0

D. Timescale

The RL agent implements a new action once per 60 second interval. While Modelica simulations operate at a much higher fidelity than 60 seconds (typically on the millisecond scale), the RL agent only interacts with the environment at the minute-scale. Updating the RL agent (through back-propagation of several deep neural networks) is a computationally expensive process and we found experimentally that 60 second intervals improved performance without causing the RL agent to progress slower than real-time.

The RL agents presented in Section IV were trained on 10,000 time steps (10,000 minutes, approximately 7 days) in one epoch, each. The environment is automatically reset when the Dymola simulation fails, although this did not occur when a slack bus is used in the system. The training for the agent with the hardware and software described in Table I completed within 2.3 hours, less than 1% of the simulated time.

IV. Results

As power flow formulations are not inherently convex, there are many solutions to improve voltage regulation. Furthermore, as the control domain increases the maximum possible performance improvement also increases. In this section we present two scenarios for voltage control with (1) ± 10 MW control and (2) ± 50 MW control over the generators real power set-point. The results show that the RL agent with the larger control domain performs better by the RL reward metric of voltage regulation. However, the RL agent with the smaller control domain maintains higher voltages specifically at the load buses and operates the generators at a higher power output which could be desirable. As a result the holistic performance of an RL controller should be taken into account when analyzing the performance of such controllers.

A. Voltage

In both scenarios the overall voltage deviations from 1 pu are reduced. In the first case (Fig. 4) the voltage is reduced at each bus with the voltage at the load buses being the lowest (under 1 pu). In the case 1 (“RL-10”) with less control over the generator set-point the voltage is increased locally at Generator 2, but the voltage deviation is reduced overall. In other words, the RL controller ignores the baseline controller goal of local voltage regulation in favor of overall voltage regulation. In case 2 (“RL-50”) the voltage is reduced throughout the network, including at the load buses that are already under 1 pu voltage (Buses 5 and 6).

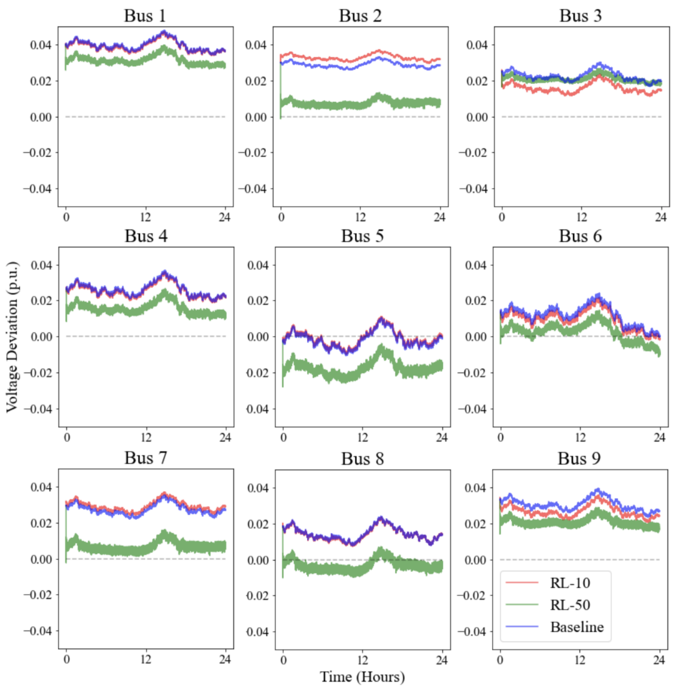

Fig. 4: Voltage deviation from 1 pu

Fig. 5 summarizes the performance of the RL agent with voltage deviation measured by the $\mathcal{L}_2$-Norm (a value of 0 would indicate that all buses achieved perfect regulation at 1 pu, while a larger value would indicate that one or more buses are over/under 1 pu). Additionally, Fig. 5 includes the total voltage spread in the system where the voltage spread is actually increased for the “RL-50” case. In the “RL-50” case voltage is centered around 1 pu giving it a lower deviation at every bus but a larger spread between buses; in the “RL-10” case the voltages are clustered slightly above 1 pu giving it a higher deviation at every bus but lower spread.

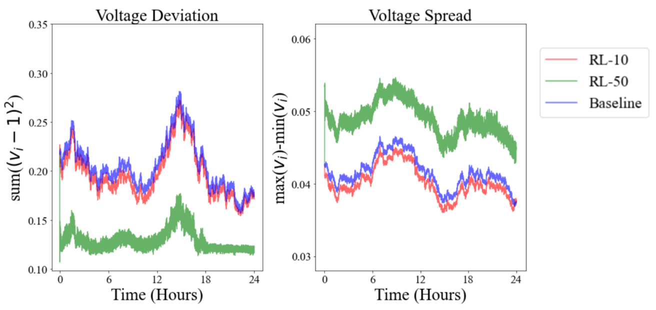

Fig. 5: Summary of network performance

B. Generator Response

The RL generated actions ramp Generator 3 to the extremes of the control domain, while load following with Generator 2 (Fig. 6). The result is that the “RL-50” case operates Generator 3 at a very low fraction of its total capacity, while the “RL-10” case (due to the inherent limitations of the control domain) operates with both generators splitting the load relatively evenly. Note that in response to the low loading of Generator 3 in the “RL-50” case the slack bus must become significantly more responsive than the “RL-10” or baseline case.

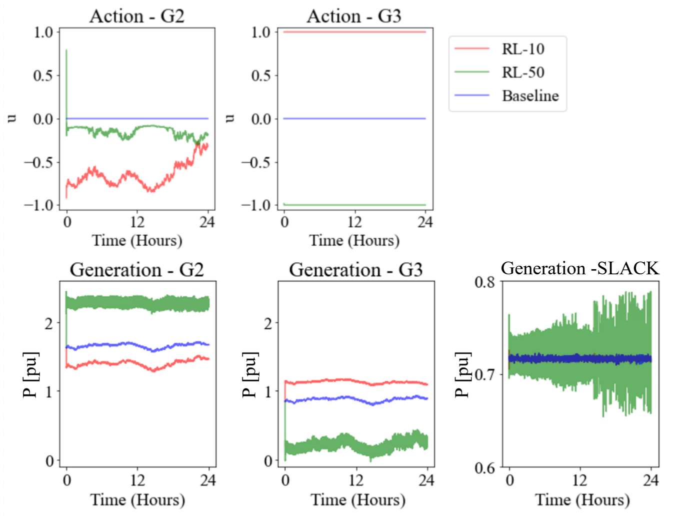

Fig. 6: Actions and output of generators

One possible reason for such a drastic difference in control strategies is that the RL agent with less control ("RL-10") is incapable of reducing the voltages all the way to 1.0 pu and instead selectively reduces the voltage at the highest voltage buses (Bus 1 and 9). Additionally, the reason that the second RL case drastically reduces generation from the controllable generators might be that the slack bus is tuned to be too responsive ($H = 1$ second, compared to $H = 3.33$ and 2.35 s for the controllable generators respectively. We might further reduce reliance on the slack bus by increasing the machine's H or increasing the impedance of the lines connected to it.

C. Impact on Loads

The performance of the RL agent control is assessed in Fig. 7 which shows that the load roughly matches demand for both the RL and baseline droop controlled scenarios (note that the per unit power is plotted with a base unit of 100 MW). A slight difference in load for the baseline and RL controlled versions can be accounted for by the fact that the ZIP loads modeled in OpenIPSL have a constant current component that lowers power consumption of the load under decreased voltages. Fig. 7 compares the transmission losses of the RL agent and the baseline controller.

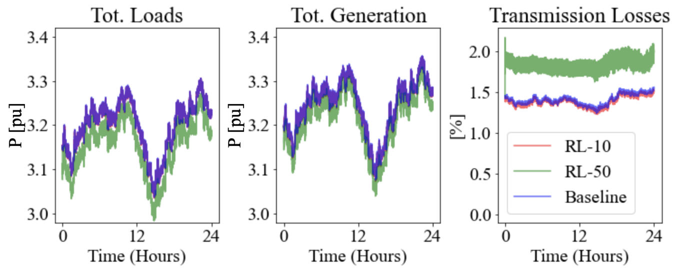

Fig. 7: Load and generation of systems

Although the system losses are higher for the "RL-50" controlled case, the overall generation is reduced by 0.6% throughout the simulation period. Both of these things can be attributed to the overall lower voltage in the network as resistive losses are higher for low voltage networks and constant current loads draw less power at lower voltages. Conversely, the "RL-10" case reduces the transmission losses, but negligibly reduces the total generation.

V. Conclusion

In this paper, we have introduced a reinforcement learning toolkit for Dymola-enabled environments. As a case study, we have provided two trained RL agents that successfully control generator set-points in real time to reduce the voltage deviations throughout the network to various degrees. The results indicate that a learning-based, model-free approach can prove useful for real-time control in dynamical environments. Future work will consider different grid objectives such as frequency regulation or control of inverter-interfaced generation. In addition, more advanced localized controllers will be compared against the RL agent's performance, versus just comparing against the standard droop controllers. Lastly, we plan to extend the presented toolkit and framework to OpenModelica version, an open-source alternative to Dymola.

References

[1] Z. Zhang, D. Zhang, and R. C. Qiu, "Deep reinforcement learning for power system applications: An overview," *CSEE Journal of Power and Energy Systems*, vol. 6, no. 1, pp. 213–225, 2020.

[2] A. Pigott, K. Baker, and C. Mosiman, "Deep Q-learning for aggregator price design," *IEEE Power and Energy Society General Meeting*, 2021.

[3] D. Ernst, M. Glavic, and L. Wehenkel, "Power systems stability control: reinforcement learning framework," *IEEE Transactions on Power Systems*, vol. 19, no. 1, pp. 427–435, 2004.

[4] Q. Huang, R. Huang, W. Hao, J. Tan, R. Fan, and Z. Huang, "Adaptive power system emergency control using deep reinforcement learning," *IEEE Transactions on Smart Grid*, vol. 11, 3 2020.

[5] X. S. Zhang, Q. Li, T. Yu, and B. Yang, "Consensus transfer Q-learning for decentralized generation command dispatch based on virtual generation tribe," *IEEE Transactions on Smart Grid*, vol. 9, no. 3, pp. 2152–2165, 2018.

[6] M. Baudette, M. Castro, T. Rabuzin, J. Lavenius, T. Bogodorova, and L. Vanfretti, "OpenIPSL: Open-Instance Power System Library - Update 1.5 to iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations," *SoftwareX*, vol. 7, pp. 34–36, Jan. 2018.

[7] O. Lukianykhin and T. Bogodorova, "ModelicaGym: Applying Reinforcement Learning to Modelica Models," in *Proceedings of the 9th International Workshop on Equation-based Object-oriented Modeling Languages and Tools*, EOOLT '19, (New York, NY, USA), pp. 27–36, Association for Computing Machinery, Nov. 2019.

[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, p. 2435, Oct. 2020.

[9] W. Chen, M. Huhn, and P. Fritzson, "A Generic FMU Interface for Modelica," in *4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, EOOLT 2011*, pp. 19–24, 2011.

[10] S. A. Dorado-Rojas, M. Navarro Catalán, M. de Castro Fernandes, and L. Vanfretti, "Performance Benchmark of Modelica Time-domain Power System Automated Simulations using Python," in *Proceedings of the American Modelica Conference 2020*.

[11] A. Pigott, "Modelicagym: Dymolagym." https://github.com/apigott/modelicagym, 2021.

[12] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," June 2016.

[13] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018. [Online] Available: https://arxiv.org/abs/1801.01290.

[14] Q. Lan, Y. Pan, A. Fyshe, and M. White, "Maxmin q-learning: Controlling the estimation bias of q-learning," 2021.