The Journal of Open Source Software

gym-electric-motor (GEM): A Python toolbox for the simulation of electric drive systems

Praneeth Balakrishna[1], **Gerrit Book**[1], **Wilhelm Kirchgässner**[1], **Maximilian Schenke**[1], **Arne Traue**[1], **and Oliver Wallscheid**[1]

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

DOI: 10.21105/joss.02498

Software

- Review
- Repository
- Archive

Editor: Kevin M. Moerman

Reviewers:

- @moorepants
- @dineshresearch

Submitted: 29 May 2020
Published: 22 July 2020

Summary

The `gym-electric-motor` (`GEM`) library provides simulation environments for electrical drive systems and, therefore, allows to easily design and analyze drive control solutions in Python. Since `GEM` is strongly inspired by OpenAI's `gym` (Brockman et al., 2016), it is particulary well-equipped for (but not limited to) applications in the field of reinforcement-learning-based control algorithms. In addition, the interface allows to plugin-in any expert-driven control approach, such as model predictive control, to be tested and to perform benchmark comparisons. The `GEM` package includes a wide variety of motors, power electronic converters and mechanical load models that can be flexibly selected and parameterized via the API. A modular software code also allows additional system components to be included in the framework simulation.

Field of Application

Electric drive systems are an important topic both in academic and industrial research due to the worldwide availability and deployment of such plants. Control algorithms for these systems have usually been designed, parameterized and tested within `MATLAB - Simulink`, which is developed and promoted specifically for such engineering tasks. In the more recent past, however, commercial software like `MATLAB` has difficulties to stay on par with the expandability and flexibility offered by open-source libraries that are available for more accessible programming languages like Python. Moreover, the latest efforts concerning industrial application of reinforcement-learning control algorithms heavily depend on Python packages like `Keras` (Chollet & others, 2015) or `Tensorflow` (Abadi et al., 2015). In order to allow easy access to an open-source drive simulation environment, the `GEM` library has been developed.

Package Architecture

The `GEM` library models an electric drive system by it's four main components: voltage supply, power converter, electric motor and mechanical load. The general structure of such a system is depicted in Figure 1.

The Journal of Open Source Software

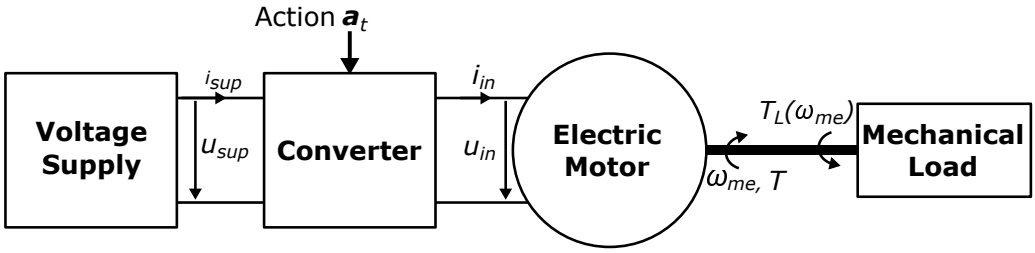

Figure 1: Structure of an electric drive system

The **voltage supply** provides the necessary power that is used by the motor. It is modeled by a fixed supply voltage u_{sup}, which allows to monitor the supply current into the converter. A **converter** is needed to supply the motor with electric power of proper frequency and magnitude, which may also include the conversion of the direct current from the supply to alternating current. Typical drive converters have switching behavior: there is a finite set of different voltages that can be applied to the motor, depending on which switches are open and which are closed. Besides this physically accurate view, a popular modeling approach for switched mode converters is based on dynamic averaging of the applied voltage u_{in}, making the voltage a continuous variable. Both of these modeling approaches are implemented and can be chosen freely, allowing usage of control algorithms that operate on a finite set of switching states or on continuous input voltages. The **electric motor** is the centerpiece of every drive system. It is modeled by a system of ordinary differential equations (ODEs), which represent the electrical behavior of the motor itself. In particular, the domain of three-phase drives makes use of coordinate transformations to view these ODEs in a more interpretable frame. In `GEM`, both, the physical (abc-) and the simplified (dq-)coordinates are available to be used as the frame of input and output variables, allowing for easy and quick controller analysis and diagnose within the most convenient coordinate frame. Finally, the torque T resulting from the motor is applied to the **mechanical load**. The load is characterized by a moment of inertia and by a load torque T_{L} that is directed against the motor torque. Load torque behavior can be parameterized with respect to the angular velocity ω_{me} in the form of constant, linear and quadratic dependency (and arbitrary combinations thereof). Speed changes that result from the difference between motor and load torque are modeled with another ODE which completely covers the mechanical system behavior. Alternatively, the motor speed can be set to a fixed value, which can be useful for the investigation of control algorithms concerning generatoric operation.

Features

A large number of different motor systems is already implemented. These include DC drives as well as synchronous and induction three-phase drives. A complete list can be viewed in the `GEM` documentation (Balakrishna et al., 2020). The corresponding power converters allow to control the motor either directly via applied voltage (continuous-control-set) or by defining the converter's switching state (finite-control-set). Specifically for the use within reinforcement-learning applications, the toolbox comes with a built-in reference generator, which can be used to create trajectories of reference operational points (e.g. for the motor current, velocity or torque). These generated references are furthermore used to calculate a reward. In the domain of reinforcement-learning, reward is the optimization variable that is to be maximized. For the control system scenario, reward is usually defined by the negative distance between the momentary operation point and the desired operation point, such that expedient controller behavior can be monitored easily. The metric by which the distance of two operational points is measured, can be adjusted to the users desire. The reward mechanism also allows to take physical limitations of the drive system into account, e.g. in the way of a notably low reward

if limit values are surpassed. Optionally, the environment can be setup such that a reset of the system is induced in case of a limit violation. In addition, built-in visualization and plotting routines allow to monitor the training process of reinforcement learning agents or the performance of expert-driven control approaches which are being tested.

Examples

A minimal example of `GEM`'s simulation capability is presented in Figure 2. The plot shows the start-up behavior of a squirrel cage induction motor connected to a rigid network concerning angular velocity ω_{me}, torque T, voltage $u_{a,b,c}$ and current $i_{d,q}$. Here, the voltage is depicted within the physical abc-frame while the current is viewed within the simplified dq-frame.

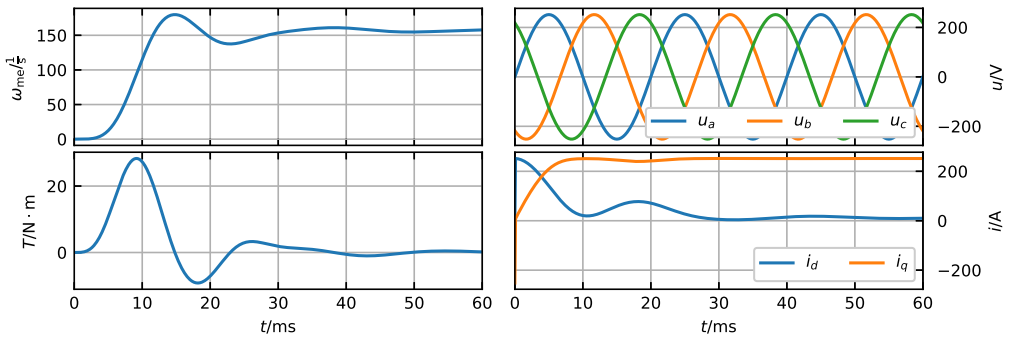

Figure 2: Simulation of a squirrel cage induction motor connected to a rigid network at $50\,\mathrm{Hz}$

Exemplary code snippets that demonstrate the usage of `GEM` are included within the projects repository:

- `ddpg_pmsm_dq_current_control`: a reinforcement-learning control approach applied to the current control of a permanent magnet synchronous motor within the dq-frame
- `ddpg_series_omega_control`: a reinforcement-learning control approach applied to the speed control of a series motor
- `dqn_series_current_control`: a reinforcement-learning control approach for finite-control-set current control of a series motor
- `pi_series_omega_control`: a conventional control algorithm applied to the speed control of a series motor

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Retrieved from https://www.tensorflow.org/

Balakrishna, P., Book, G., Kirchgässner, W., Schenke, M., Traue, A., & Wallscheid, O. (2020). Gym-electric-motor documentation. Retrieved from https://upb-lea.github.io/gym-electric-motor/

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

Chollet, F., & others. (2015). Keras. https://keras.io.