Tiitinen, Lauri; Hartikainen, Hannu; Peretti, Luca; Hinkkanen, Marko

# motulator: Motor Drive Simulator in Python

*Published in:*
2023 IEEE International Electric Machines and Drives Conference, IEMDC 2023

*DOI:*
10.1109/IEMDC55163.2023.10238938

Published: 06/09/2023

*Document Version*
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

# *motulator*: Motor Drive Simulator in Python

Lauri Tiitinen*, Hannu Hartikainen*, Luca Peretti†, and Marko Hinkkanen*
*Aalto University, Espoo, Finland
E-mail: lauri.tiitinen@aalto.fi; hannu.1.hartikainen@aalto.fi; marko.hinkkanen@aalto.fi
†KTH Royal Institute of Technology, Stockholm, Sweden
E-mail: lucap@kth.se

***Abstract*—This paper deals with the time-domain simulation of electric machine drives in Python. Python offers a full-featured numerical computing environment, comparable to leading commercially available alternatives, whilst being fully open-source. Adopting the open-source ecosystem allows electric drives researchers to utilize new methods from other fields, easily share their findings, and improve collaboration with other researchers both in academia and industry. For this purpose, we have launched the Python-based time-domain simulation platform *motulator* for electric drives. To learn more, the reader is encouraged to visit the GitHub page of the project: https://github.com/Aalto-Electric-Drives/motulator.**

***Index Terms*—Control, electric motor drives, open source, power converters, Python, time-domain simulations.**

## I. Introduction

Python is a high-level general-purpose programming language. Its popularity has been increasing both in academia and industry, not only in data science and machine learning but also in several other fields. The numerous openly available high-quality scientific libraries and tools make Python a full-fledged numerical computing environment comparable even to the most leading-edge commercial alternatives.

In the field of electric machines and drives, fast and accurate time-domain simulations of nonlinear systems (including both continuous-time and discrete-time elements) are necessary for research, development, and teaching. While the use of Python for machine design purposes shows a growing trend, the electric drives community has been slow to adopt it. This is unfortunate since the Python ecosystem could help researchers leverage methods and approaches from other fields and boost the integration of different Python-based tools to achieve complete system simulations. Even in education, Python-based exercises could help improve the programming skills of students. Furthermore, the open-source approach could increase research collaboration both within academia and industry.

While there are several notable projects related to electrical machine design and analysis, e.g., [1], [2], and control engineering in Python, e.g., [3], [4], only a few projects deal with the time-domain simulation of electric drives in specific [5]. For this purpose, we have launched the *motulator* simulation platform for electric drives. The library covers the most common continuous-time plant models as well as an extensive selection of the most relevant control methods. High priority has been placed on code readability. With clearly documented and carefully designed interfaces, the user can easily extend the functionality of the library with their self-developed models and control algorithms.

This work was supported in part by ABB Oy and in part by the Academy of Finland Centre of Excellence in *High-Speed Electromechanical Energy Conversion Systems*.

## II. System Framework

Fig. 1 shows a block diagram of a generic motor drive system. Motor drives are sampled-data systems, consisting of both continuous-time and discrete-time systems and the interfaces between them [6], [7]. Digital control systems are typically affected by a computational delay of one sampling period since some computational time is necessary between the sampling instance of the measurements (such as phase currents) and the controller output.

### A. Continuous-Time System

In *motulator*, the continuous-time system is represented by an electric drive model, which consists of the motor, mechanics, and converter submodels. These continuous-time models are collected in the package `motulator.model`. Each submodel specifies a set of differential equations that describe how the system evolves with time. The electric drive model facilitates the interfaces between these submodels and the discrete-time control system, as well as constructs the complete state derivative for the differential equation solver. By default, the system of differential equations is solved using an explicit Runge-Kutta algorithm of order 5(4) from the open-source SciPy library [8], [9].

Peak-valued complex space vectors denoted in boldface are used throughout this paper. The superscript $\mathrm{s}$ marks stator coordinates, while space vectors without the superscript refer to synchronous coordinates. Variables with the subscript *abc* are three-element vectors. Furthermore, monospaced font denotes classes or functions in the library.

*1) Induction Motor:* Fig. 2 shows the $\Gamma$-model equivalent circuit of an induction motor in stator coordinates. In coordinates rotating at $\omega_\mathrm{s}$, the induction motor is governed by

$$\frac{\mathrm{d}\boldsymbol{\psi}_\mathrm{s}}{\mathrm{d}t} = \boldsymbol{u}_\mathrm{s} - R_\mathrm{s}\boldsymbol{i}_\mathrm{s} - \mathrm{j}\omega_\mathrm{s}\boldsymbol{\psi}_\mathrm{s} \tag{1a}$$

$$\frac{\mathrm{d}\boldsymbol{\psi}_\mathrm{r}}{\mathrm{d}t} = -R_\mathrm{r}\boldsymbol{i}_\mathrm{r} - \mathrm{j}\omega_\mathrm{r}\boldsymbol{\psi}_\mathrm{r} \tag{1b}$$

where $\boldsymbol{u}_\mathrm{s}$ is the stator voltage, $\boldsymbol{i}_\mathrm{s}$ is the stator current, $R_\mathrm{s}$ is the stator resistance, $\boldsymbol{i}_\mathrm{r}$ is the rotor current, and $R_\mathrm{r}$ is the rotor resistance. The angular slip frequency $\omega_\mathrm{r} = \omega_\mathrm{s} - \omega_\mathrm{m}$, where $\omega_\mathrm{m}$ is the electrical angular speed of the rotor. In the

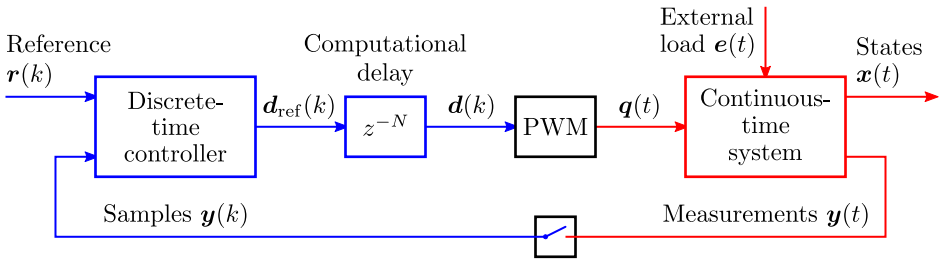


Fig. 1. Block diagram of a sampled-data motor drive system. Argument $k$ marks discrete signals, while $t$ refers to continuous signals.

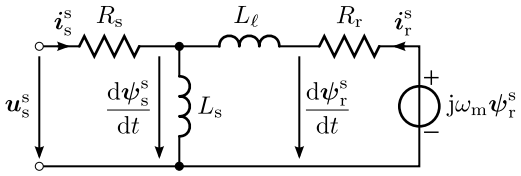


Fig. 2. Induction motor Γ model.

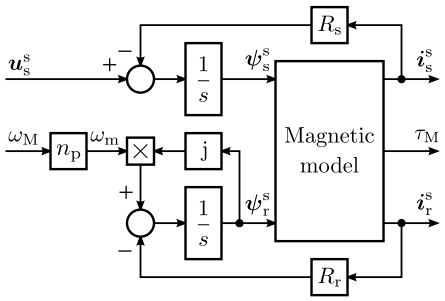


Fig. 3. Block diagram of an induction motor model.

magnetically linear case, the stator and rotor flux linkages, respectively, are given by

$$\boldsymbol{\psi}_{\mathrm{s}} = L_{\mathrm{s}}(\boldsymbol{i}_{\mathrm{s}} + \boldsymbol{i}_{\mathrm{r}}) \tag{2a}$$

$$\boldsymbol{\psi}_{\mathrm{r}} = \boldsymbol{\psi}_{\mathrm{s}} + L_\ell \boldsymbol{i}_{\mathrm{r}} \tag{2b}$$

where $L_{\mathrm{s}}$ is the stator inductance and $L_\ell$ is the leakage inductance. The electromagnetic torque is

$$\tau_{\mathrm{M}} = \frac{3n_{\mathrm{p}}}{2}\mathrm{Im}\{\boldsymbol{i}_{\mathrm{s}}\boldsymbol{\psi}_{\mathrm{s}}^{*}\} \tag{3}$$

where $n_{\mathrm{p}}$ is the pole-pair number.

If magnetic saturation is omitted, the well-known T, Γ, and inverse-Γ models are mathematically equivalent. The Γ model is selected as the base model in *motulator* since it can be extended with a magnetic saturation model in a straightforward manner [10]. For convenience, a method to initialize the motor model using inverse-Γ model parameters in the linear case is also provided. Fig. 3 shows the block diagram corresponding to the implementation in the `InductionMotor` class. This base model includes a linear magnetic model.

The linear model can be extended with magnetic saturation. Since the Γ model is used, magnetic saturation can be sufficiently modeled with a nonlinear stator inductance, while the leakage inductance can be typically assumed constant [11]. The base model is extended with main-flux saturation in `InductionMotorSaturated`. A simple power function is used,

$$L_{\mathrm{s}}(\psi_{\mathrm{s}}) = \frac{L_{\mathrm{su}}}{1 + (\beta\psi_{\mathrm{s}})^S} \tag{4}$$

where $\psi_{\mathrm{s}} = |\boldsymbol{\psi}_{\mathrm{s}}|$ is the flux magnitude, $L_{\mathrm{su}}$ is the unsaturated stator inductance, and $\beta$ and $S$ are positive constants which define the shape of the saturation curve [11], [12].

*2) Synchronous Motor:* Fig. 4 shows the block diagram of a generic synchronous motor model in rotor coordinates. The motor model is more straightforward to compute in this coordinate system since the dependency on the physical orientation of the rotor vanishes from the magnetic model. Fig. 5 illustrates the model as observed from stator coordinates.

The voltage equation in rotor coordinates is

$$\frac{\mathrm{d}\boldsymbol{\psi}_{\mathrm{s}}}{\mathrm{d}t} = \boldsymbol{u}_{\mathrm{s}} - R_{\mathrm{s}}\boldsymbol{i}_{\mathrm{s}} - \mathrm{j}\omega_{\mathrm{m}}\boldsymbol{\psi}_{\mathrm{s}}. \tag{5}$$

Using the complex space vector convention, the stator flux linkage in the magnetically linear case is

$$\boldsymbol{\psi}_{\mathrm{s}} = L_{\mathrm{d}} i_{\mathrm{d}} + \mathrm{j}L_{\mathrm{q}} i_{\mathrm{q}} + \psi_{\mathrm{f}} \tag{6}$$

where $i_{\mathrm{d}} = \mathrm{Re}\{\boldsymbol{i}_{\mathrm{s}}\}$ and $i_{\mathrm{q}} = \mathrm{Im}\{\boldsymbol{i}_{\mathrm{s}}\}$ are the d- and q-axis current components, $L_{\mathrm{d}}$ and $L_{\mathrm{q}}$ are the d- and q-axis inductances, respectively, and $\psi_{\mathrm{f}}$ is the permanent-magnet (PM) flux linkage. The electromagnetic torque is calculated as in (3). Various synchronous motor types, ranging from PM synchronous motors (PMSMs) to synchronous reluctance motors (SyRMs), can be characterized by the magnetic model, which maps the stator flux linkage to the stator current, i.e., $\boldsymbol{f} : \boldsymbol{\psi}_{\mathrm{s}} \to \boldsymbol{i}_{\mathrm{s}}(\boldsymbol{\psi}_{\mathrm{s}})$. As an example, in (6), a surface-PMSM is obtained with $L_{\mathrm{d}} = L_{\mathrm{q}}$ and SyRM with $\psi_{\mathrm{f}} = 0$. In *motulator*, the class `SynchronousMotor` represents this magnetically linear synchronous motor model.

For magnetic saturation in synchronous motors, an extended version of the power function (4) is used [13]. This function adds cross-saturation to the model, and is implemented in the class `SynchronousMotorSaturated` as

$$i_{\mathrm{d}}(\psi_{\mathrm{d}}, \psi_{\mathrm{q}}) = \frac{\psi_{\mathrm{d}}}{L_{\mathrm{du}}}\left[1 + (\alpha|\psi_{\mathrm{d}}|)^a + \frac{\gamma L_{\mathrm{du}}}{d+2}|\psi_{\mathrm{d}}|^c|\psi_{\mathrm{q}}|^{d+2}\right] - i_{\mathrm{f}} \tag{7a}$$

$$i_{\mathrm{q}}(\psi_{\mathrm{d}}, \psi_{\mathrm{q}}) = \frac{\psi_{\mathrm{q}}}{L_{\mathrm{qu}}}\left[1 + (\beta|\psi_{\mathrm{q}}|)^b + \frac{\gamma L_{\mathrm{qu}}}{c+2}|\psi_{\mathrm{d}}|^{c+2}|\psi_{\mathrm{q}}|^d\right] \tag{7b}$$

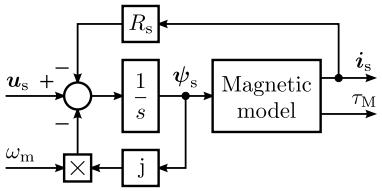


Fig. 4. Block diagram of a synchronous motor model in rotor coordinates.

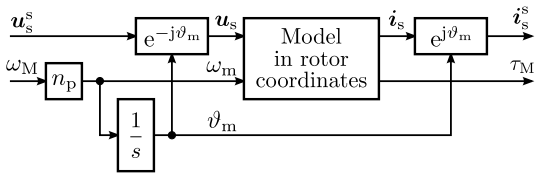


Fig. 5. Block diagram of a synchronous motor model observed from stator coordinates.

where $L_\mathrm{du}$ and $L_\mathrm{qu}$ are the d- and q-axis unsaturated inductances, respectively, $\alpha, \beta, \gamma$ are nonnegative coefficients, and $a, b, c, d$ are nonnegative exponents. The current $i_\mathrm{f}$ corresponds to the magnetomotive force of the PMs. Selecting $\gamma = 0$ yields the saturation model in (4). The saturation model parameters of a physical motor can be estimated, e.g., using the standstill identification method described in [14].

Alternatively, flux maps can be used to represent the nonlinear saturation characteristics. This approach typically requires more data but can represent more detailed magnetic models that simple analytical models may fail to capture. Fig. 6 shows an exemplary flux map for the saturation characteristics of a 5-kW PM-SyRM, whose data is available from the SyR-e project [15]. In *motulator*, this look-up table-based approach is implemented in the class `SynchronousMotorSaturatedLUT`, and tools for importing and plotting the flux-map data are available in `sm_flux_maps.py`.

### B. Mechanics

Fig. 7 shows a block diagram of a mechanical system with stiff mechanics. The mechanical subsystem describes the dynamic relationship between the total torque applied on the rotor shaft and the speed of the rotor. The stiff mechanics are governed by

$$J\frac{\mathrm{d}\omega_\mathrm{M}}{\mathrm{d}t} = \tau_\mathrm{M} - \tau_\mathrm{L} \tag{8a}$$

$$\frac{d\vartheta_\mathrm{M}}{\mathrm{d}t} = \omega_\mathrm{M} \tag{8b}$$

where $J$ is the total moment of inertia, $\omega_\mathrm{M}$ is the mechanical angular speed of the rotor, and $\vartheta_\mathrm{M}$ is the mechanical angle of the rotor. The electrical angular speed is proportional to the mechanical angular speed multiplied by the pole-pair number $n_\mathrm{p}$, i.e., $\omega_\mathrm{m} = n_\mathrm{p}\omega_\mathrm{M}$.

The total load torque $\tau_\mathrm{L}$ is the sum of an external load torque and a speed-dependent load torque component,

$$\tau_\mathrm{L} = \tau_{\mathrm{L},\omega} + \tau_{\mathrm{L,t}} \tag{9}$$

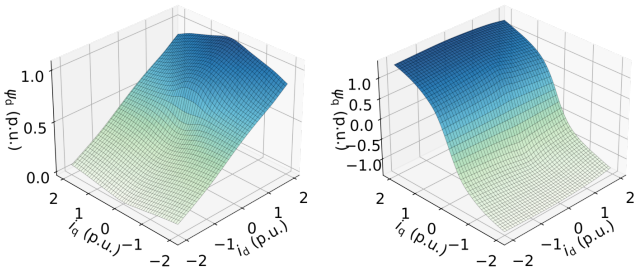


Fig. 6. Saturation characteristics of the *THOR* 5-kW PM-SyRM [15].

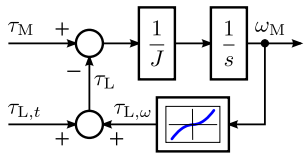


Fig. 7. Block diagram for mechanical subsystem with stiff mechanics corresponding to the `Mechanics` class.

As an example, the speed-dependent load torque could model viscous friction,

$$\tau_{\mathrm{L},\omega} = B\omega_\mathrm{M} \tag{10}$$

where $B$ is the damping coefficient. As another example, the quadratic load torque, typical to pump and fan applications, is

$$\tau_{\mathrm{L},\omega} = k\omega_\mathrm{M}^2 \mathrm{sign}(\omega_\mathrm{M}) \tag{11}$$

where $k$ is a positive friction coefficient.

In addition to this stiff mechanics model implemented in the class `Mechanics`, a two-mass mechanical model example is available in `MechanicsTwoMass`.

### C. Converter

*1) Inverter:* Fig. 8 shows an equivalent model of a three-phase two-level voltage-source inverter implemented in the class `Inverter`. The switches are assumed ideal and the DC-bus voltage $u_\mathrm{dc}$ constant. Hence, the AC-side voltage is

$$\boldsymbol{u}_\mathrm{s} = \frac{2}{3}\left(q_\mathrm{a} + q_\mathrm{b}\mathrm{e}^{\mathrm{j}2\pi/3} + q_\mathrm{c}\mathrm{e}^{\mathrm{j}4\pi/3}\right)u_\mathrm{dc} = \boldsymbol{q}u_\mathrm{dc} \tag{12}$$

where $\boldsymbol{q}$ is the instantaneous complex switching state vector generated using carrier comparison. The DC-side current can be derived from the power balance as

$$i_\mathrm{dc} = \frac{3}{2}\mathrm{Re}\{\boldsymbol{q}\boldsymbol{i}_\mathrm{s}^*\} \tag{13}$$

*2) Three-Phase Diode-Bridge Rectifier:* The inverter model can be also extended with a model for the supply side, e.g., the six-pulse diode bridge illustrated in Fig. 9. The six-pulse diode bridge rectifier with an LC filter is governed by

$$C\frac{\mathrm{d}u_\mathrm{dc}}{\mathrm{d}t} = i_\mathrm{L} - i_\mathrm{dc} \tag{14a}$$

$$L\frac{\mathrm{d}i_\mathrm{L}}{\mathrm{d}t} = u_\mathrm{di} - u_\mathrm{dc} \tag{14b}$$

where $L$ is the DC-bus inductance and $C$ is the DC-bus capacitance. Furthermore, $i_\mathrm{L}$ is the inductor current, which

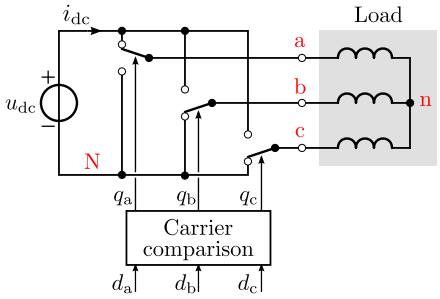


Fig. 8. Equivalent model of a three-phase two-level voltage source inverter. The switching state signals are generated using carrier comparison.

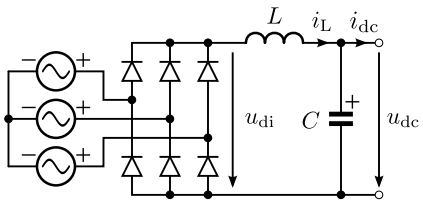


Fig. 9. Six-pulse diode bridge rectifier. The three-phase voltage source and the diode bridge are assumed ideal.

due to the diodes must be nonnegative. Assuming a stiff three-phase supply and ideal diodes, the output voltage of the diode bridge is

$$u_{\mathrm{di}} = \max\left(u_{\mathrm{g,abc}}\right) - \min\left(u_{\mathrm{g,abc}}\right) \tag{15}$$

where $u_{\mathrm{g,abc}}$ is a vector of the three-phase grid voltages. The class `FrequencyConverter` implements the combination of a six-pulse diode bridge rectifier and a two-level voltage source inverter.

*3) Carrier Comparison:* In Fig. 8, carrier comparison is used to calculate the instantaneous switching state signals from the discrete-time duty ratios calculated by the discrete-time controller. Carrier comparison functions as an interface between the discrete-time controller and the continuous-time plant as illustrated in Fig. 10. The discrete-time duty ratios are constant over the sampling period $T_\mathrm{s}$ or optionally over the switching period $T_\mathrm{sw} = 2T_\mathrm{s}$ if a double update is not used.

In *motulator*, the switching time instants are explicitly computed at the beginning of the sampling period. This approach results in faster simulations, as the differential equation solver does not need to search for the zero crossings of the carrier wave and duty ratios. It should be noted that the discrete-time controller returns the sampling time for the following control cycle. This feature allows, e.g., synchronization of the carrier wave to the fundamental frequency, which is typically needed in the case of low switching frequencies.

If the switching ripple is not of interest, the switching-cycle average voltage can be used by replacing the carrier comparison with a zero-order hold (ZOH) of the duty ratios for increased simulation speed. This is the default option in *motulator*.

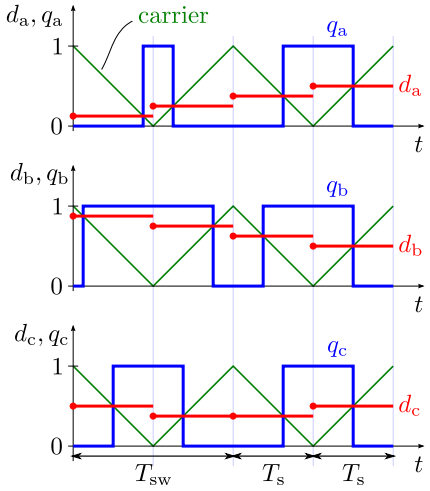


Fig. 10. Carrier comparison. The discrete-time duty ratios take values in the the range [0, 1], while the switching states are either 0 or 1.

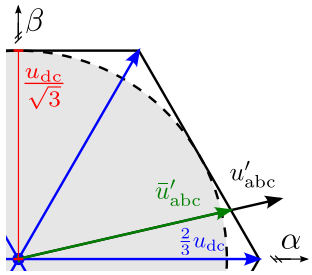


Fig. 11. Voltage reference clipping by means of minimum phase error method. The blue vectors represent the six active voltage vectors.

### D. Discrete-Time Controller

The discrete-time controller implements the designed control law and computes the discrete-time duty ratios for the carrier comparison every sampling period. A library of exemplary control methods and a pulse-width modulation (PWM) algorithm are available in the `motulator.control` package.

*1) Control Algorithms:* The project contains modules for exemplary implementations for, e.g., V/Hz-based control methods, sensored and sensorless vector control, stator-flux-vector control, and vector control with signal injection, based on [16]–[21] and the references therein. The library aims to maintain a thorough collection of algorithms for the most relevant control concepts for electric machine drives with clear, easy-to-follow implementations. With these as a starting point, the user can implement and simulate their own control algorithms.

*2) PWM:* The discrete-time duty ratios for three-phase PWM are calculated using the symmetrical suboscillation method. This approach is mathematically equivalent to the standard space-vector PWM [22]. First, a zero-sequence component

$$u_0 = \frac{\min\left(u_{\mathrm{abc}}\right) + \max\left(u_{\mathrm{abc}}\right)}{2} \tag{16a}$$

is added to the three-phase voltage references,

$$u'_{\mathrm{abc}} = u_{\mathrm{abc}} - u_0 \tag{16b}$$

Algorithm 1. Dynamic induction motor models.

```
class InductionMotor:
    # Γ-equivalent model of an induction motor
    def __init__(self, n_p=2, R_s=3.7, R_r=2.5, L_ell=.023, L_s
         =.245):
        self.n_p = n_p
        self.R_s, self.R_r = R_s, R_r
        self.L_ell, self.L_s = L_ell, L_s
        # Initial values
        self.psi_ss0, self.psi_rs0 = 0j, 0j

    def currents(self, psi_ss, psi_rs):
        # Compute the stator and rotor currents
        i_rs = (psi_rs - psi_ss)/self.L_ell
        i_ss = psi_ss/self.L_s - i_rs
        return i_ss, i_rs

    def magnetic(self, psi_ss, psi_rs):
        # Magnetic model
        i_ss, i_rs = self.currents(psi_ss, psi_rs)
        tau_M = 1.5*self.n_p*np.imag(i_ss*np.conj(psi_ss))
        return i_ss, i_rs, tau_M

    def f(self, psi_ss, psi_rs, u_ss, w_M):
        # Compute the state derivatives
        i_ss, i_rs, tau_M = self.magnetic(psi_ss, psi_rs)
        dpsi_ss = u_ss - self.R_s*i_ss
        dpsi_rs = -self.R_r*i_rs + 1j*self.n_p*w_M*psi_rs
        return [dpsi_ss, dpsi_rs], i_ss, tau_M

class InductionMotorSaturated(InductionMotor):
    # Γ-equivalent model of an induction motor model with main-
        flux saturation.
    def __init__(self, n_p=2, R_s=3.7, R_r=2.5, L_ell=.023,
         L_su=.34, beta=.84, S=7):
        super().__init__(n_p=n_p, R_s=R_s, R_r=R_r, L_ell=L_ell
            )
        # Saturation model
        self.L_s = lambda psi: L_su/(1+(beta*np.abs(psi))**S)

    def currents(self, psi_ss, psi_rs):
        # Override the base class method.
        L_s = self.L_s(psi_ss)
        # Currents
        i_rs = (psi_rs - psi_ss)/self.L_ell
        i_ss = psi_ss/L_s - i_rs
        return i_ss, i_rs
```

The voltage reference should be limited to the edge of the voltage hexagon, as illustrated in Fig. 11. Using the minimum phase error method, the voltage reference is limited as

$$\bar{u}'_{\mathrm{abc}} = \frac{u'_{\mathrm{abc}}}{\max\left(1, \frac{2u'_{\mathrm{abc}}}{u_{\mathrm{dc}}}\right)} \tag{16c}$$

Finally, the discrete duty ratios are calculated as

$$d_{\mathrm{abc}} = \frac{1}{2} + \frac{\bar{u}'_{\mathrm{abc}}}{u_{\mathrm{dc}}} \tag{16d}$$

These duty ratios are the input for carrier comparison.

## III. Examples

Several example simulation scripts are available in the library. Together with the documentation, the implementation of additional control algorithms and models is straightforward. Here, the implementation of a dynamic model of an induction motor and a simulation script with a compatible control algorithm are showcased in more detail. A simulation of a permanent-magnet synchronous reluctance machine is also shown.

### A. Continuous-Time Induction Motor Model

Algorithm 1 shows the Python implementation of the induction motor model illustrated in the block diagram in Fig. 3. The method `InductionMotor.f` computes the stator and rotor flux linkage state derivatives to be used by the ordinary differential equation solver to compute the time evolution given an initial condition. The default values in the initializer method correspond to a 2.2-kW 50-Hz induction motor.

The inherited class `InductionMotorSaturated` extends the base class with main-flux saturation using a simple explicit function. Inheritance enables code reuse simplifying the code base but also helps ensure that a common interface is maintained.

### B. Sensorless Vector Control of Induction Motors

Fig. 12 shows the block diagram for sensorless vector control for an induction motor. The sensorless flux observer corresponds to [18] and a two-degrees-of-freedom proportional-integral (PI) control is used for the current controller. Algorithm 2 shows an exemplary implementation of this controller in *motulator*.

Algorithm 3 shows an example script used to configure a 2.2-kW induction motor drive system and a discrete-time controller with perfect motor model parameter estimates. Magnetic saturation is not considered in the plant model in this example. Switching-cycle averaging is selected with the parameter `pwm=False` and the computational delay is modeled as one sampling period. Fig. 13 shows the results of the simulation.

### C. Observer-Based V/Hz Control of a PM-SyRM

Next, a simulation of a motor model with magnetic saturation is showcased. This example considers a saturated 5-kW four-pole 85-Hz PM-SyRM, whose magnetic characteristics are available from the SyR-e project [15] as flux maps, illustrated in Fig. 6. In `motulator`, these flux maps can be implemented with the class `SynchronousMotorSaturatedLUT` as look-up tables.

For the controller, an observer-based V/Hz control algorithm with constant inductance estimates is used [17]. Naturally, better control performance could be achieved by considering magnetic saturation also in the controller. The quadratic load torque profile in (11) is used, and the speed reference is ramped to the rated speed first in the positive and then in the negative direction. Fig. 14 shows the simulation results.

## IV. Conclusions

A Python-based time-domain simulation platform for electric drives is presented. The package includes continuous-time models for induction motors, synchronous motors, and

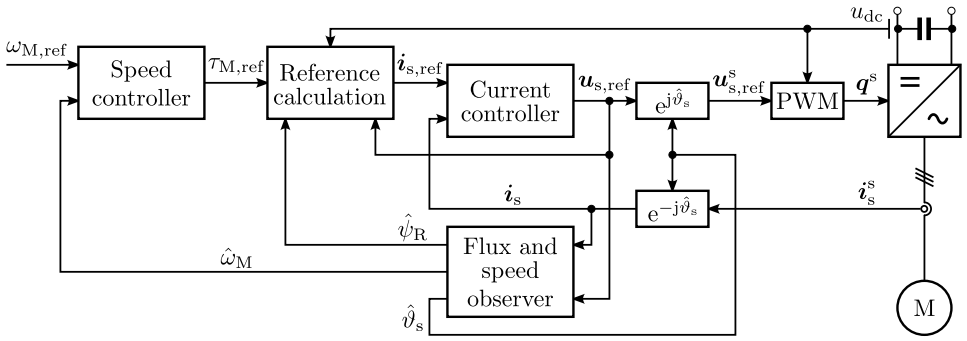


Fig. 12. Block diagram of speed-sensorless vector control for induction motors [18].

Algorithm 2. Control algorithm for speed-sensorless vector control of an induction motor.

```
class InductionMotorVectorCtrl(Ctrl):
# Vector control for an induction motor drive.
def __init__(self, pars):
    super().__init__()
    self.T_s = pars.T_s
    self.w_m_ref = pars.w_m_ref
    self.n_p = pars.n_p
    self.speed_ctrl = SpeedCtrl(pars)
    self.current_ref = CurrentRef(pars)
    self.current_ctrl = CurrentCtrl(pars)
    self.observer = SensorlessObserver(pars)
    self.pwm = PWM(pars)

def __call__(self, mdl):
    # Get the speed reference
    w_m_ref = self.w_m_ref(self.t)

    # Measure the feedback signals
    i_s_abc = mdl.motor.meas_currents() # Phase currents
    u_dc = mdl.conv.meas_dc_voltage() # DC-bus voltage

    # Get the states
    u_s = self.pwm.realized_voltage
    psi_R = self.observer.psi_R
    w_m = self.observer.w_m
    theta_s = self.observer.theta_s

    # Space vector and coordinate transformation
    i_s = np.exp(-1j*theta_s)*abc2complex(i_s_abc)

    # Outputs
    tau_M_ref = self.speed_ctrl.output(w_m_ref/self.n_p,
         w_m/self.n_p)
    i_s_ref, tau_M_ref_lim = self.current_ref.output(
         tau_M_ref, psi_R)
    w_s = self.observer.output(u_s, i_s, w_m)
    u_s_ref = self.current_ctrl.output(i_s_ref, i_s)
    d_abc_ref, u_s_ref_lim = self.pwm.output(u_s_ref, u_dc,
          theta_s, w_s)

    # Update the states
    self.pwm.update(u_s_ref_lim)
    self.speed_ctrl.update(tau_M_ref_lim)
    self.current_ref.update(u_s_ref, u_dc)
    self.current_ctrl.update(u_s_ref_lim, w_s)
    self.observer.update(i_s, w_s)
    self.update_clock(self.T_s)

    return self.T_s, d_abc_ref
```

Algorithm 3. Simulation script for sensorless vector control of induction motor.

```
import motulator as mt

# Define base values for plotting
base = mt.BaseValues(
    U_nom=400, # Line-line rms voltage
    I_nom=5, # Rms current
    f_nom=50, # Frequency
    tau_nom=14.6, # Torque
    P_nom=2.2e3, # Power
    n_p=2) # Number of pole pairs

# Configure the induction motor using its Γ parameters
motor = mt.InductionMotor(R_s=3.7, R_r=2.5, L_ell=.023, L_s
     =.245, p=2)
# Mechanics model
mech = mt.Mechanics(J=.015)
# Inverter model
conv = mt.Inverter(u_dc=540)
# System model
mdl = mt.InductionMotorDrive(motor, mech, conv)

# Configure the control system.
ctrl = mt.InductionMotorVectorCtrl(
    mt.InductionMotorVectorCtrlPars(sensorless=True))

# Configure speed reference and external load
ctrl.w_m_ref = lambda t: (t > .2)*(.5*base.w)
mdl.mech.tau_L_t = lambda t: (t > .75)*base.tau_nom

# Configure the simulation object and run it
sim = mt.Simulation(mdl, ctrl, pwm=False, delay=1)
sim.simulate(t_stop=1.5)

# Plot the results
mt.plot(sim, base=base)
```

converters, as well as an extensive collection of discrete-time control algorithm examples of the most relevant control strategies for machine drives.

## References


[1] P. Bonneel, J. Le Besnerais, R. Pile, and E. Devillers, "Pyleecan: an open-source python object-oriented software for the multiphysic design optimization of electrical machines," in *Proc. ICEM*. IEEE, 2018, pp. 948–954.

[2] "eMach: Open source machine modeling and optimization framework," 2022. [Online]. Available: https://github.com/Severson-Group/eMach

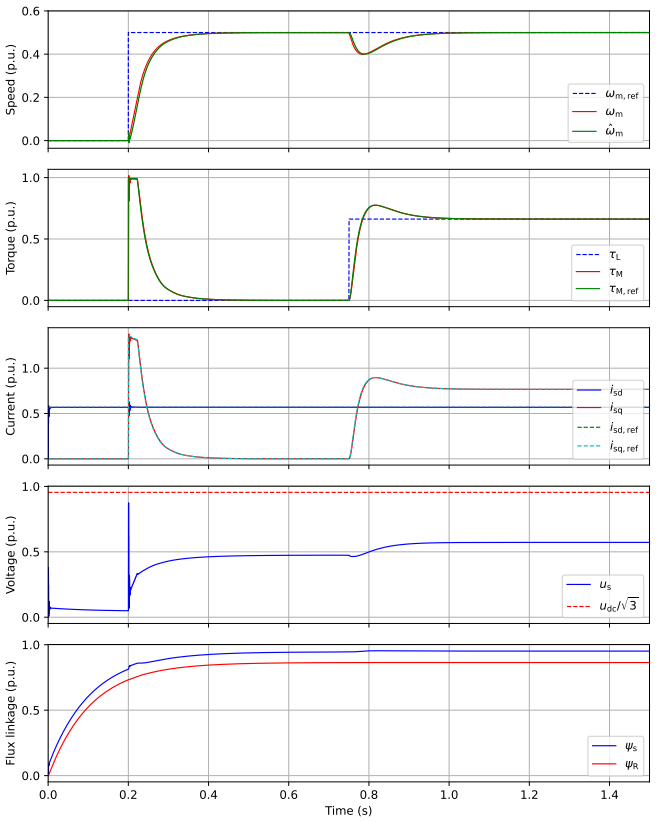


Fig. 13. Sensorless vector control of a 2.2-kW induction motor.

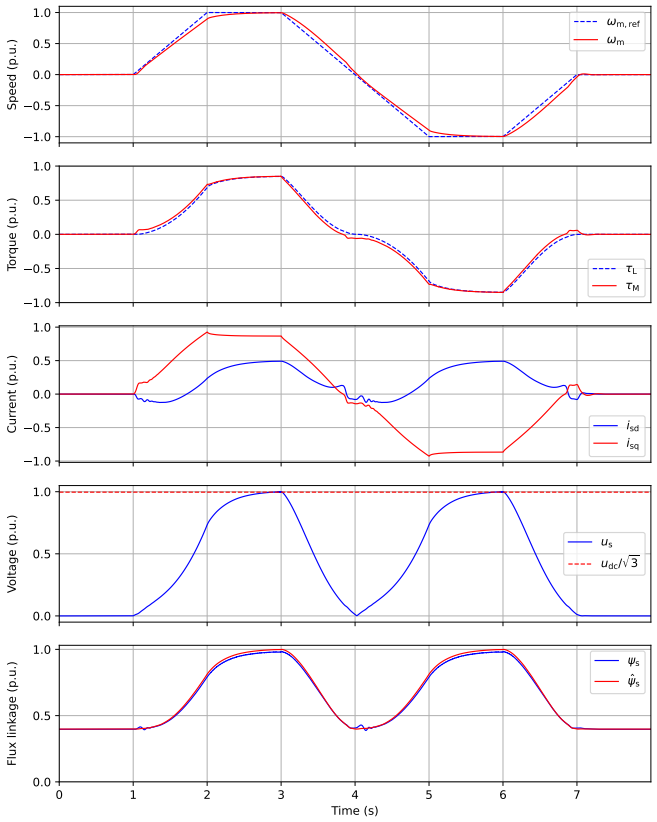


Fig. 14. Observer-based V/Hz control of a saturated 5-kW PM-SyRM.


[3] S. Fuller, B. Greiner, J. Moore, R. Murray, R. van Paassen, and R. Yorke, "The python control systems library (python-control)," in *Proc. IEEE Conf. Decis. Control.*, 2021, pp. 4875–4881.

[4] S. Lucia, A. Tătulea-Codrean, C. Schoppmeyer, and S. Engell, "Rapid development of modular and sustainable nonlinear model predictive control solutions," *Control Engineering Practice*, vol. 60, pp. 51–62, 2017.

[5] P. Balakrishna, G. Book, W. Kirchgässner, M. Schenke, A. Traue, and O. Wallscheid, "gym-electric-motor (GEM): A Python toolbox for the simulation of electric drive systems," *J. Open Source Software*, vol. 6, no. 58, p. 2498, 2021.

[6] G. F. Franklin, J. D. Powell, M. L. Workman *et al.*, *Digital control of dynamic systems*, 3rd ed. Addison-Wesley Reading, MA, 1998.

[7] S. Buso and P. Mattavelli, *Digital control in power electronics*, 2nd ed. Morgan & Claypool Publishers, 2015.

[8] J. R. Dormand and P. J. Prince, "A family of embedded Runge-Kutta formulae," *J. Comput. Appl. Math.*, vol. 6, no. 1, pp. 19–26, 1980.

[9] P. Virtanen, R. Gommers, T. E. Oliphant *et al.*, "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[10] G. R. Slemon, "Modelling of induction machines for electric drives," *IEEE Trans. Ind. Appl.*, vol. 25, no. 6, pp. 1126–1131, Nov./Dec. 1989.

[11] Z. Qu, M. Ranta, M. Hinkkanen, and J. Luomi, "Loss-minimizing flux level control of induction motor drives," *IEEE Trans. Ind. Appl.*, vol. 48, no. 3, pp. 952–961, May/Jun. 2012.

[12] H. De Jong, "Saturation in electrical machines," in *Proc. ICEM*, vol. 3, Athens, Greece, Sep. 1980, pp. 1545–1552.

[13] Z. Qu, T. Tuovinen, and M. Hinkkanen, "Inclusion of magnetic saturation in dynamic models of synchronous reluctance motors," in *Proc. ICEM*, Marseille, France, Sep. 2012, pp. 994–1000.

[14] M. Hinkkanen, P. Pescetto, E. Mölsä, S. E. Saarakkala, G. Pellegrino, and R. Bojoi, "Sensorless self-commissioning of synchronous reluctance motors at standstill without rotor locking," *IEEE Trans. Ind. Appl.*, vol. 53, no. 3, pp. 2120–2129, May/Jun. 2017.

[15] F. Cupertino, G. Pellegrino, P. Cagnetta, S. Ferrari, and M. Perta, "SyR-e: Synchronous reluctance (machines)-evolution." [Online]. Available: www.github.com/SyR-e

[16] L. Tiitinen, M. Hinkkanen, and L. Harnefors, "Stable and passive observer-based V/Hz control for induction motors," in *Proc. IEEE ECCE*, Detroit, MI, Oct. 2022.

[17] L. Tiitinen, M. Hinkkanen, J. Kukkola, M. Routimo, G. Pellegrino, and L. Harnefors, "Stable and passive observer-based V/Hz control for synchronous motors," in *Proc. IEEE ECCE*, Detroit, MI, Oct. 2022.

[18] M. Hinkkanen, L. Harnefors, and J. Luomi, "Reduced-order flux observers with stator-resistance adaptation for speed-sensorless induction motor drives," *IEEE Trans. Power Electron.*, vol. 25, no. 5, pp. 1173–1183, May 2010.

[19] G. Pellegrino, E. Armando, and P. Guglielmi, "Direct flux field-oriented control of IPM drives with variable DC link in the field-weakening region," *IEEE Trans. Ind. Appl.*, vol. 45, no. 5, pp. 1619–1627, Sep./Oct. 2009.

[20] H. A. A. Awan, M. Hinkkanen, R. Bojoi, and G. Pellegrino, "Stator-flux-oriented control of synchronous motors: A systematic design procedure," *IEEE Trans. Ind. Appl.*, vol. 55, no. 5, pp. 4811–4820, Sep./Oct. 2019.

[21] S. Kim, J.-I. Ha, and S.-K. Sul, "PWM switching frequency signal injection sensorless method in IPMSM," *IEEE Trans. Ind. Appl.*, vol. 48, no. 5, pp. 1576–1587, Sep./Oct. 2012.

[22] A. Hava, R. Kerkman, and T. Lipo, "Simple analytical and graphical methods for carrier-based PWM-VSI drives," *IEEE Trans. Power Electron.*, vol. 14, no. 1, pp. 49–61, Jul. 1999.