

SWITCHED RELUCTANCE MOTOR DRIVE SIMULATION USING OPEN-SOURCE SOFTWARE

Bogdan Brković¹, Milovan Majstorović¹, Mateja Ivanović¹, Mladen Terzić¹

1: University of Belgrade, School of Electrical Engineering, 11000 Belgrade, Serbia
e-mails: brkovic@etf.bg.ac.rs, majstorovic@etf.bg.ac.rs, im225002p@student.etf.bg.ac.rs,
terzic@etf.bg.ac.rs

Abstract

The switched reluctance motor is a viable magnet-free option for low and medium power applications. Current challenges limiting widespread application are the inherent noise and vibration issues. Mitigation of these problems while retaining the required performance can be achieved by improving the motor design and control. The research presented in this paper is aimed at developing a detailed simulation model of the switched reluctance motor drive system. This process includes: motor parameter identification based on finite-element analysis, composition of the joined model of the motor and dedicated power converter, automatic simulation running, output data logging, and processing. The entire procedure is conducted using open-source software platforms. The developed modelling and simulation approach enables the determination of motor and converter performance indicators, including a detailed calculation of semiconductor losses and temperature rise. Furthermore, this provides a foundation for applying design and control optimization algorithms to achieve enhanced performance characteristics.

Keywords: switched reluctance motor, power converter, drive, modelling, simulation.

1 Introduction

The switched reluctance motor (SRM) is recognized as a topology of interest in low and medium power applications such as home appliances and small electric vehicles [1–4]. Compared to its main counterpart, the permanent magnet synchronous machine (PMSM), the SRM has a distinct advantage of being a magnet-free device. This feature makes the SRM more cost-effective and eco-friendly, considering the volatility of the permanent magnet market and the devastating procedure of rare-earth material extraction. The major drawbacks of the SRM are noise and vibration inherently related to the double-salient structure [5]. To overcome these challenges, improved motor and converter design and control approaches are required.

The SRM design procedure is based on analytical calculations or finite element analysis (FEA). The two approaches can be combined to take advantage of the FEA accuracy and the time-efficiency of the analytical approach. This is commonly done by using FEA to obtain flux and torque lookup tables which are then used to compose the SRM dynamic model. This model can then be used to simulate SRM operation under various conditions using any software package equipped with standard non-linear differential equation solvers.

The SRM can be supplied exclusively from a dedicated power converter. Multiple topologies have been proposed over the years [6, 7]. The most commonly employed topology is the Asymmetric Half-Bridge Converter (AHBC), due to its relatively low cost, control flexibility, and simplicity.

A simulation of the SRM drive is required in the development stage to assess the drive performance under various operating modes. Both the motor and converter need to be simulated in sufficient detail within a reasonable time frame. As previously stated, the motor can be simulated in great detail using FEA software, however such simulations are by no means time-efficient. On the other hand, there is a wide array of circuit simulation software packages. Notably, LTspice is a powerful open-source simulation tool incorporating actual device models [8]. This allows an accurate prediction of converter performance, most notably calculation of losses and dynamic behaviour. There is only the matter of interfacing the converter model to the SRM model. In this paper, a detailed SRM model based on flux and torque lookup tables is developed and implemented in the LTspice circuit simulation tool. The lookup tables are generated using the FEA software platform FEMM 4.2 [9] in combination with GNU Octave [10]. To the best of our knowledge, no previous implementations of an SRM model in a circuit analysis software such as LTspice have been reported. Finally, the complete drive model is assembled in LTspice. Interaction with the LTspice model (starting and stopping the model execution, output data recording and processing) is conducted entirely using a dedicated Python script.

The paper is composed as follows. Following the Introduction, fundamental information regarding the applied SRM and AHBC are given in section 2. Details regarding the development and implementation of the SRM drive model using open-source software platforms are given in section 3. The model results are displayed in section 4 and the conclusions and plans for future work are presented in section 5.

2 SRM drive overview

The SRM under investigation has six stator poles and ten rotor poles. Such a machine is aptly referred to as a 6/10 SRM. The machine cross-section is displayed in Figure 1. The operating principle is relatively simple - when a given phase is energized a magnetic field established in the corresponding stator poles attracts the nearest rotor poles. By applying excitation in an appropriate order and with suitable timing, high average torque with minimal oscillations is obtained.

Typical SRM waveforms are illustrated in Figure 2. The diagrams display the phase inductance, excitation voltage and current, and electromagnetic torque developed by a single phase. The variation of quantities is shown with respect to the rotor angle. Characteristic positions of the rotor with corresponding qualitative flux line distributions are also displayed. Two characteristic rotor positions are identified: the unaligned position, when the stator pole is between two rotor poles and the inductance value is minimal, and the aligned position, when the axes of stator and rotor poles are aligned and the inductance value is maximal. The inductance variation between these two positions is approximately linear, disregarding magnetic saturation. The phase is usually energized before the overlap between the stator and rotor poles begins and the current is quickly increased. Prior to reaching full alignment, the voltage polarity is reversed and the current is reduced back to zero, after which the voltage is maintained at zero value until the next cycle. The obtained electromagnetic torque is mostly positive, however a slight negative pulse cannot be avoided practically, due to the finite current decrease rate. Positive torque is generated when the inductance is increasing and vice versa. No torque is generated when the inductance is constant. Without diving too deep into the SRM model itself, only the basic

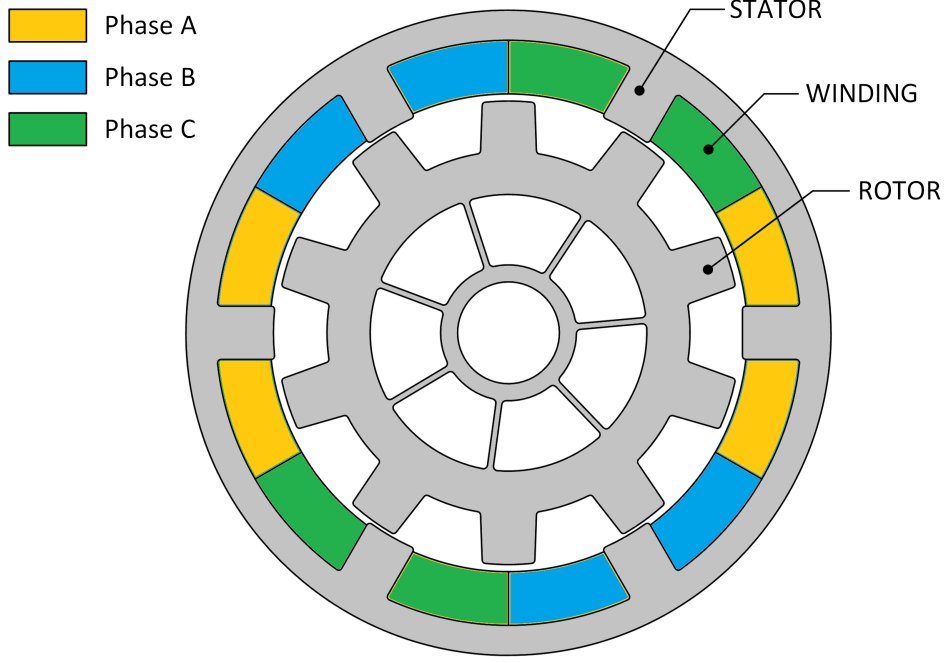


Figure 1: SRM cross-section

equations are given as follows [5]:

$$u_\xi = R_\xi i_\xi + \frac{d\psi_\xi}{dt} = R_\xi i_\xi + L_\xi(i_\xi, \theta) \frac{di_\xi}{dt} + \frac{\partial L_\xi(i_\xi, \theta)}{\partial \theta} i_\xi, \quad (1)$$

$$T_{e\xi} = \frac{1}{2} i_\xi^2 \frac{\partial L_\xi(i_\xi, \theta)}{\partial \theta}, \quad (2)$$

$$\theta = \int \Omega dt. \quad (3)$$

The variables and parameters in the previous expressions are defined as:

- ξ denotes the observed phase (A, B, or C),
- u_ξ , i_ξ , and ψ_ξ are the voltage, current, and flux linkage of the observed phase,
- θ is the rotor position with respect to phase A,
- Ω is the rotor angular speed,
- $T_{e\xi}$ is the electromagnetic torque generated by the observed phase, and
- R_ξ and L_ξ are the resistance and inductance of the observed phase.

It should be emphasized that the inductances are dependent on current values due to magnetic saturation, which makes the SRM model highly non-linear. This non-linearity will be accounted for by modelling the SRM using flux and torque lookup tables obtained from FEM simulations, as elaborated in section 3. The motion equation is excluded from the model, as the rotor speed is considered constant in all upcoming analyses.

The SRM supply voltage obviously needs to be varied frequently which can only be accomplished by means of a power converter. Considering the required voltage values ($+V_{DC}$, $-V_{DC}$, and 0), a two-level converter such as an H-bridge would be suitable. However, considering the fact that the

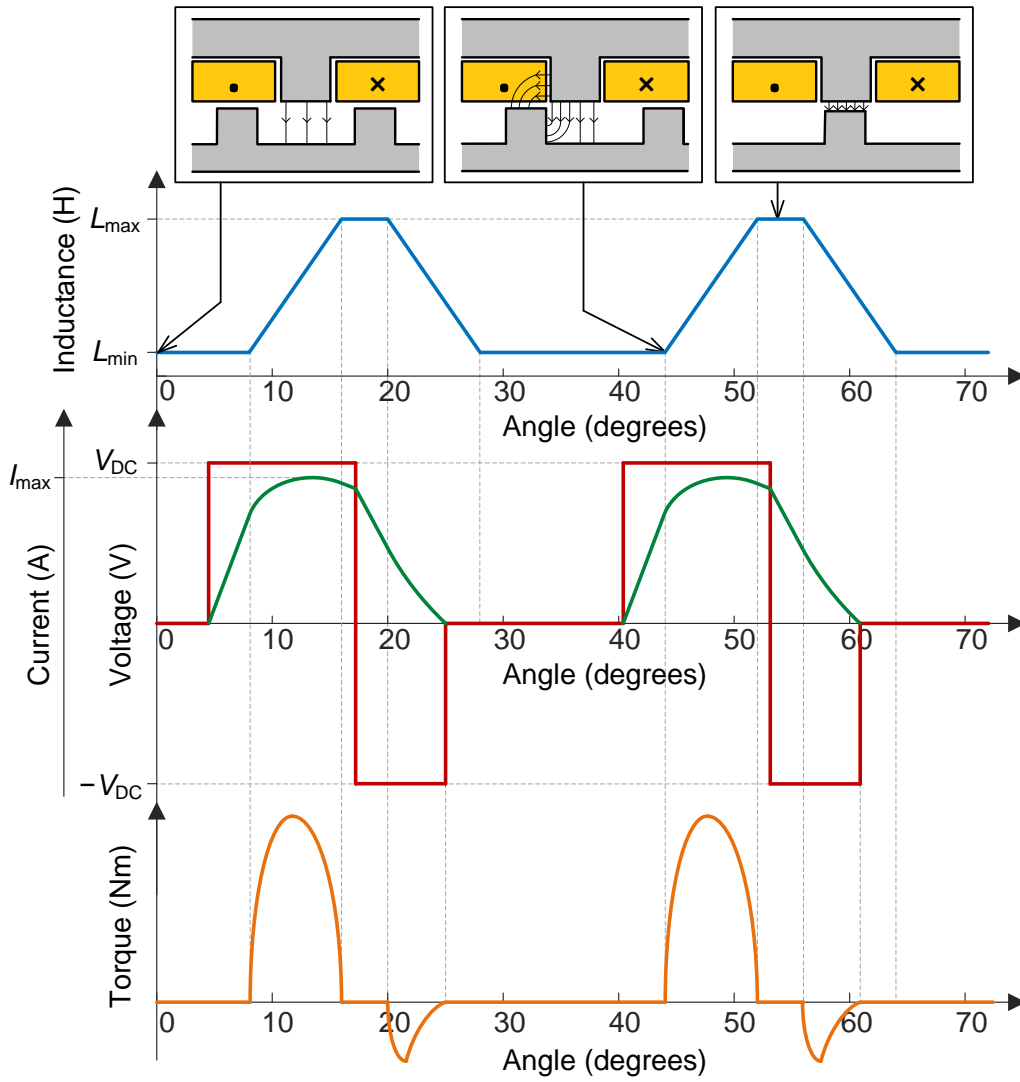


Figure 2: Example of SRM waveforms for a typical operating mode

SRM current is unidirectional, it is very common in practice to apply the asymmetric half-bridge converter (AHBC). The AHBC topology and its connections to respective SRM phases is schematically displayed in Figure 3. The AHBC structurally resembles a standard H-bridge module, except that it only employs two switches (most commonly MOSFET or IGBT) and two diodes for each phase. As previously stated, this is due to the fact that the current is unidirectional, which even eliminates the need for reverse diodes in parallel with the switches (even though these are inherently present in most cases). Reducing the number of power switches reduces the total component cost, which makes the AHBC a preferred option for SRM supply. Moreover, the control algorithm is simpler, as only two switches per phase are controlled, and the switching losses tend to be lower. The only drawback is that the converter is composed of discrete components which may increase manufacturing cost and complexity.

The control approach will now be briefly explained for a typical operating mode. Example voltage and current waveforms along with corresponding AHBC switching states are shown in Figure 4. Hysteresis current control is commonly applied in the following manner:

- positive DC link voltage V_{DC} is applied by turning on the power switches when the turn-on angle (θ_{on}) is exceeded, which causes the current to increase;

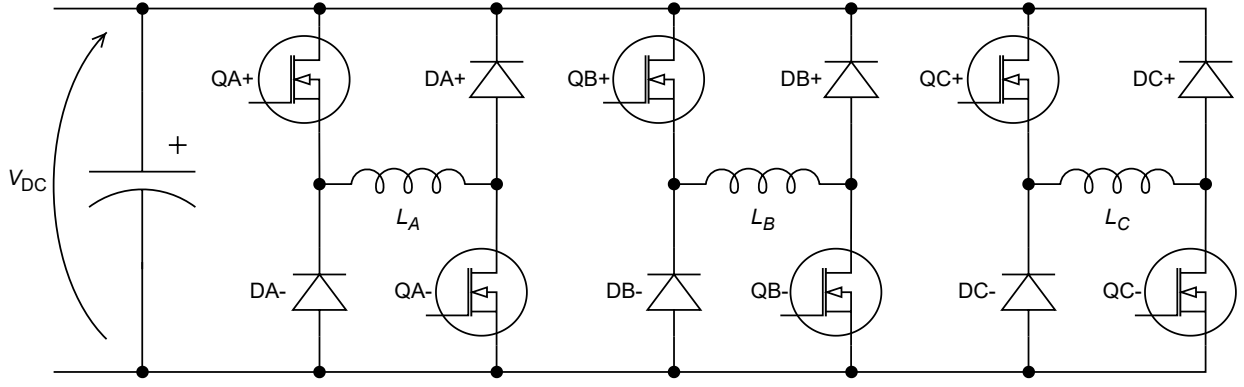


Figure 3: Schematic display of SRM phases supplied from an AHBC

- when the current exceeds the value $I_{ref} + \Delta I$ (I_{ref} is the reference and ΔI the hysteresis band), the lower switch is turned off and approximately zero voltage is applied to the phase winding, causing the current to gradually decrease;
- when the current drops to $I_{ref} - \Delta I$, the lower switch is again turned on and the current increases due to positive DC link voltage;
- the previous two steps are repeated until the turn-off angle (θ_{off}) is reached, after which both switches are turned off and negative voltage $-V_{DC}$ is applied through the diodes;
- the negative voltage reduces the current to zero.

It should be noted that either of the two switches can be manipulated to achieve hysteresis control, and normally both are used alternately to achieve an even loss distribution. The selection of turn-on and turn-off angles is crucial for maximizing the average torque and minimizing the torque ripple. Different values of θ_{on} and θ_{off} should be applied for various speeds and loads to achieve optimal operating conditions.

3 Model synthesis and implementation

The flowchart depicting the model development and execution procedure is shown in Figure 5. The applied software tools are indicated and the key outputs are highlighted. The entire procedure can be divided into three four steps:

- FEM model composition,
- FEM model execution by means of Octave code,
- LTspice model assembly, and
- LTspice model execution by means Python code.

After completing steps (i) and (ii), flux and torque lookup tables are generated. Based on these lookup tables, the LTspice model is composed in step (iii). Finally, a Python script for automating the LTspice simulation execution and output data logging is developed in step (iv). The obtained outputs are the SRM drive performance indicators, such as currents, voltages, power switch losses, etc. Each of the listed steps is analysed in detail in the following subsections. Subsection 3.1 describes steps (i) and (ii), whereas subsection 3.2 is concerned with steps (iii) and (iv).

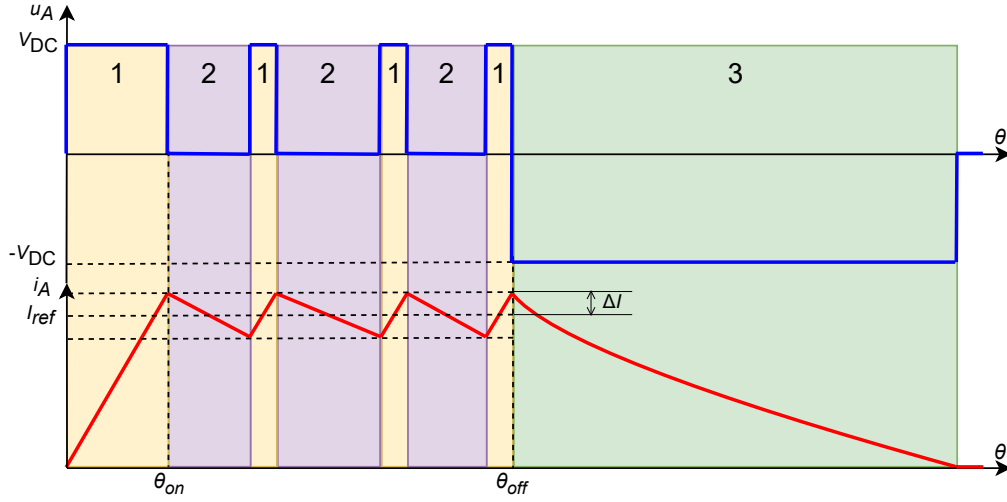
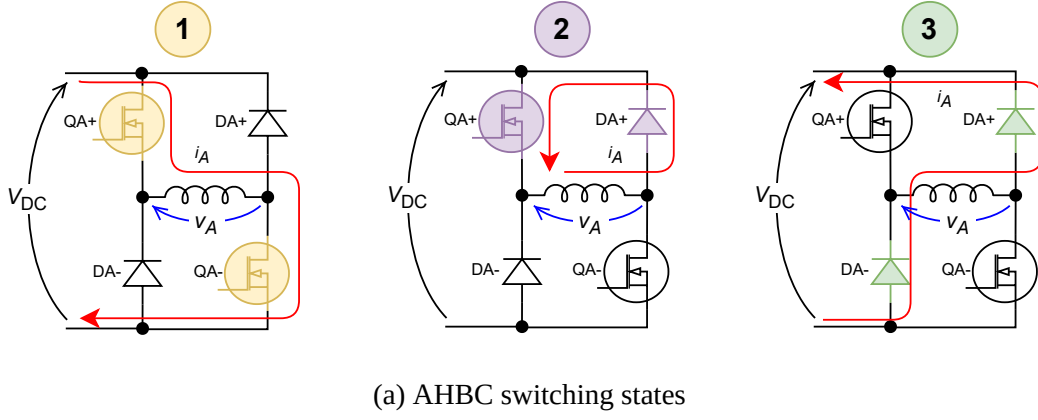


Figure 4: Illustration of SRM current control using AHBC

3.1 Flux and torque lookup table generation (FEMM & Octave)

The FEM model is developed based on the construction data of the 6/10 SRM displayed in Figure 1. The open-source software platform FEMM 4.2 for finite element-based electromagnetic calculations is used. The SRM geometry is plotted and appropriate materials are assigned to different areas. After this is completed, the machine geometry is divided into a great number of small (finite) triangular elements. The obtained finite element mesh is displayed in Figure 6a. When the mesh of appropriate density is formed, the calculation is conducted and the field distribution within the analysed domain (machine) is obtained. An example flux density distribution is displayed in Figure 6b.

Manual model execution is generally inconvenient, as many input parameters need to be set prior to each calculation. This becomes very cumbersome and time-consuming when a wide range of operating modes needs to be analysed. In this paper, the FEM model was used for obtaining flux and torque lookup tables (LUTs) with respect to winding currents and rotor angular position. As the accuracy of the resulting SRM model is directly dependent on the resolution of these LUTs, over 2000 simulations for various current and rotor angle combinations are required. This process is therefore automated by interfacing FEMM with GNU Octave by means of the OctaveFEMM toolbox [11]. This toolbox allows Octave to modify the model, run the simulation, and access the output values.

The process of generating flux and torque LUTs is depicted in the flowchart of Figure 7a. Note that

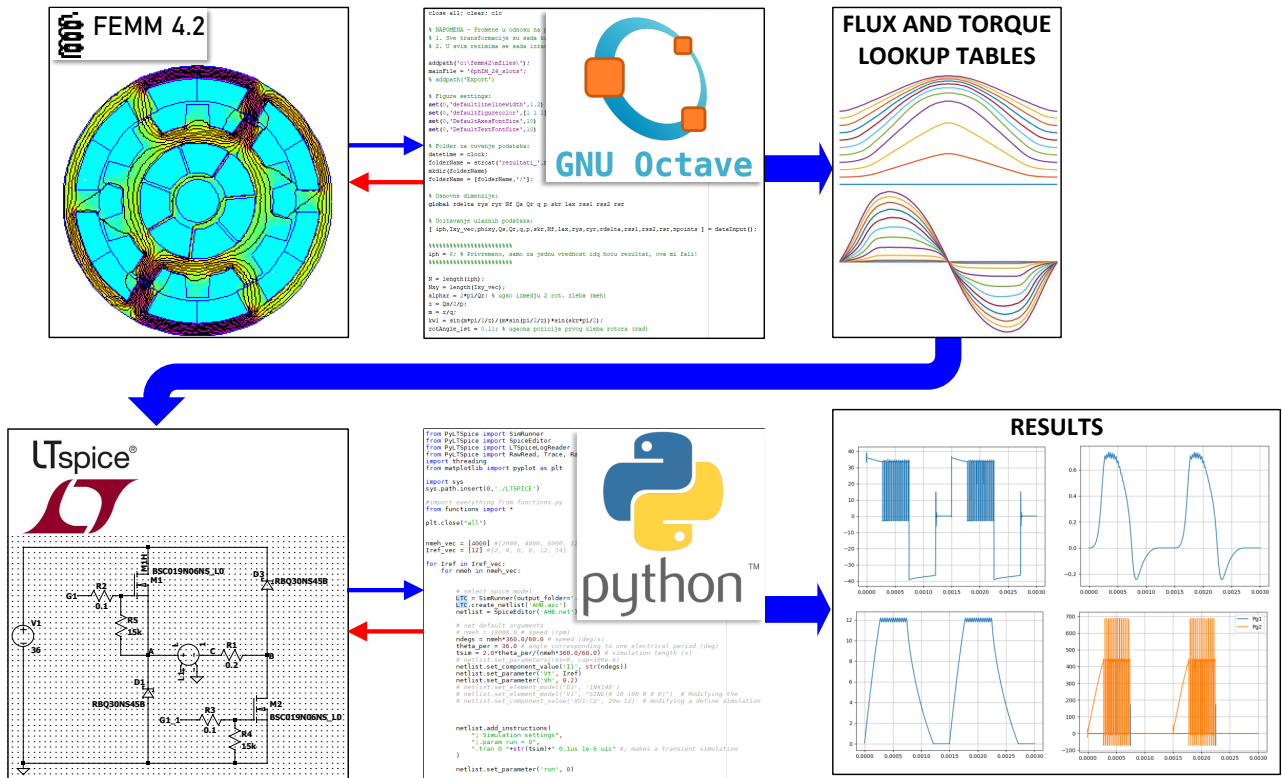


Figure 5: Flowchart of the model synthesis, implementation, and execution procedure

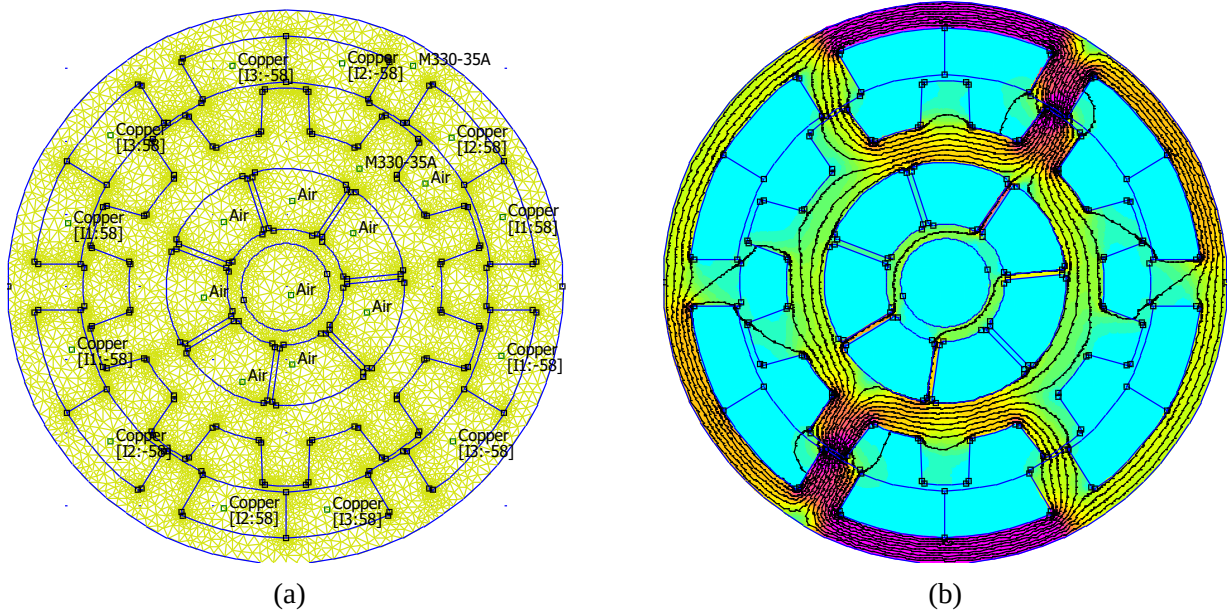
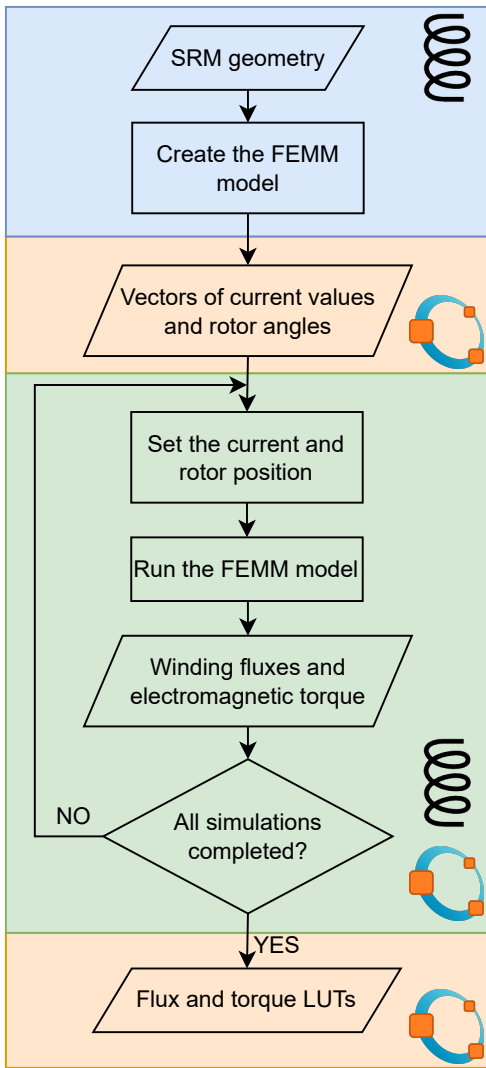
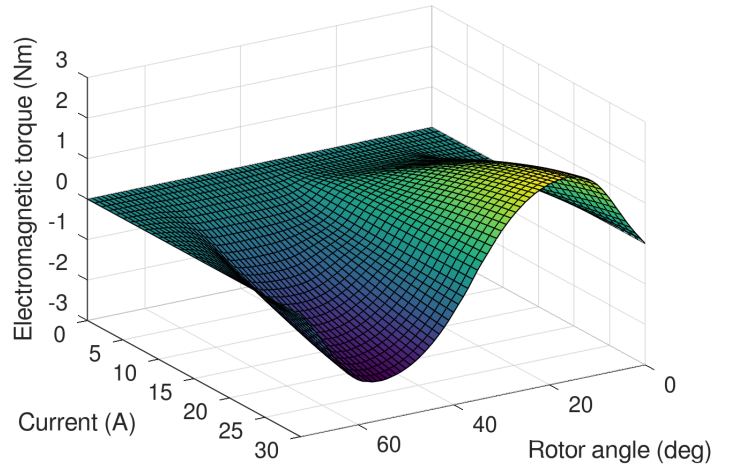
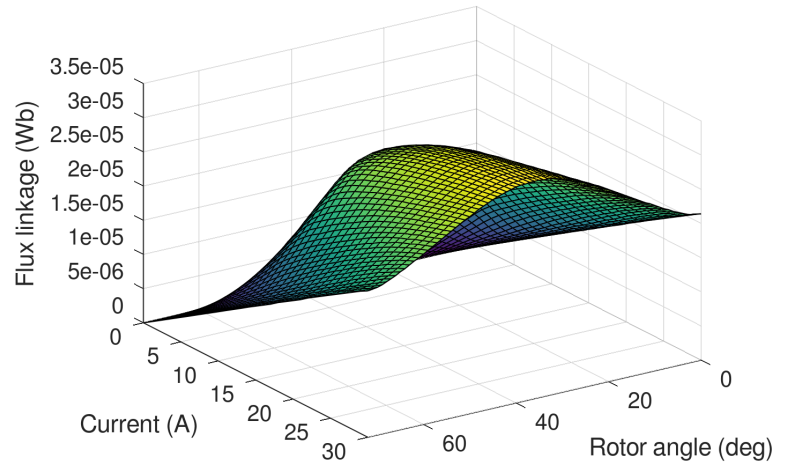


Figure 6: SRM cross-section in FEMM 4.2 environment: (a) finite element mesh, (b) flux density distribution obtained after calculation

the software tools applied in each stage of the procedure are indicated by appropriate icons. The model is generated using only FEMM, although this step can also be carried out using Octave. After the model is formed, input values are provided to Octave, specifically the set of currents and rotor angles for LUT formation. A loop is then initiated, wherein FEMM calculations are conducted for all current and angle combinations. This part of the process is automated: Octave invokes the model, sets the current and angle to the required values, runs the model, and obtains the flux and torque values. Upon exiting the loop, two-dimensional flux and torque LUTs are formed and exported to text files for further use in forming the SRM numerical model. With a 1° angle resolution and 1 A current resolution, the LUT contains a total of 2263 entries. This rule of thumb ensures high modelling accuracy while maintaining the number of executed FEM simulations and total LUT generation time within reasonable limits. Additionally, increasing the number of data points in the LUT eventually reduces the simulation efficiency without significantly improving the accuracy. The graphical representation of the flux and torque LUTs in the form of surface diagrams is given in Figure 7b.



(a)



(b)

Figure 7: (a) Flux and torque LUT calculation flowchart, (b) flux and torque LUTs represented by 3D diagrams

3.2 SRM drive model assembly and execution (LTspice & Python)

As indicated in Figure 5, the flux and torque LUTs are used to form an SRM model in the LTspice environment. While the converter model is implemented in a straightforward manner, the SRM model includes electrical, magnetic, and mechanical quantities. Therefore, appropriate electrical equivalents need to be employed. Moreover, LUT implementation in LTspice is quite involved. Software tools that enable a simpler SRM model formulation are available, but none of them provides accurate modelling of power switching device dynamics and losses. As the power converter is an integral and most sensitive part of the SRM drive, a high-fidelity model is crucial for proper performance prediction.

The complete SRM drive model implemented in LTspice is displayed in Figure 8. The model can be divided into several functional entities as designated in the figure:

1. The main part of the model – converter circuit connected to the phase winding terminals. The phase winding is modelled as resistor connected in series with a dependent voltage source corresponding to the winding electromotive force (EMF). When the EMF is calculated, the current is obtained simply as:

$$i_A = \frac{u_A - e_A}{R_A}, \quad (4)$$

where u_A is the terminal voltage, e_A is the EMF, and R_A is the phase winding resistance.

2. The EMF calculation circuit based on differentiating the winding flux. For this purpose, the flux value is assigned to a voltage controlled current source connected in series with an inductor. The current value corresponds to the flux and the inductance corresponds to the number of turns per phase, thereby essentially obtaining the EMF value as:

$$e_A = N_s \frac{d\Phi_A(i_A, \theta)}{dt}, \quad (5)$$

where N_s is the number of turns per phase (“inductance”), and Φ_A is the flux per pole (“current”). Note that a series RC snubber is added in parallel to avoid numerical issues due to connecting a current source in series with an inductor. The RC element filters out the high-frequency noise caused by the switching actions. The flux value is obtained from the LUT as a function of winding current i_A and rotor angle θ .

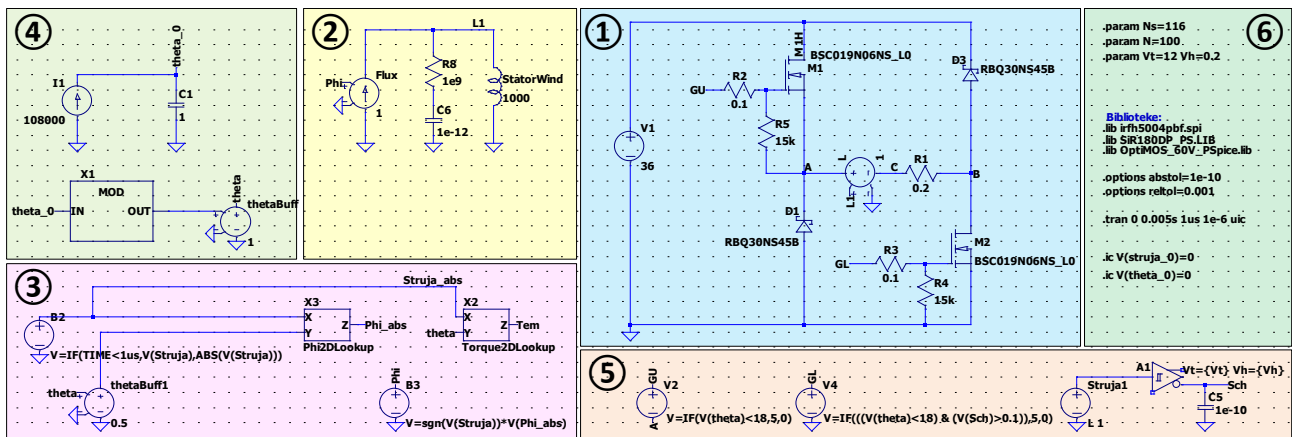


Figure 8: LTspice model of the SRM drive: 1 – converter circuit, 2 – SRM phase winding model, 3 – flux and torque calculation from LUTs, 4 – rotor angle calculation, 5 – control circuits, 6 – model inputs (parameters, libraries, initial conditions) and simulation settings

3. Flux and torque LUTs are implemented by using multiple behavioural voltage sources whose outputs are defined by the LTspice *table* command. A more detailed representation is given in Figure 9, where two behavioural sources are represented. This example refers to the flux LUT, but the same principle is applied for the torque as well. Note that voltages of ports X and Y correspond to input values of the current and rotor angle, respectively. The source denoted Byx0 provides an output value from the associated table, dependent on the rotor angle and corresponding to the current value of 0 A. The model contains 30 similar sources corresponding to current values ranging from 1 A to 30 A, each providing an output dependent on the rotor angle by interpolating the values from the LUT:

$$V(\text{Byxj}) = z_{jk} + (V(Y) - y_j) \cdot \frac{z_{j(k+1)} - z_{jk}}{y_{(k+1)} - y_k}, \quad y_k \leq V(Y) < y_{(k+1)}, \quad (6)$$

where x_j , y_k , and z_{jk} represent LUT values of current, angle, and flux, respectively. The source B3 provides an output corresponding to the actual (input) current value X by interpolating the outputs of voltage sources Byxj and Byx(j+1) :

$$V(Z) = V(\text{Byxj}) + (V(X) - x_j) \cdot \frac{V(\text{Byx(j+1)}) - V(\text{Byxj})}{x_{(j+1)} - x_j}, \quad x_j \leq V(X) < x_{(j+1)}, \quad (7)$$

where $V(Z)$ denotes the output flux value. The presented approach requires a total of 32 behavioural voltage sources, each with a corresponding *table* command. Additionally, parameters x_j , y_k , and z_{jk} need to be defined by including multiple *param* commands. The entire process is very involved and time-consuming, and can be expedited by using a Python script to assign parameter values directly from the previously obtained text files.

4. Rotor angle is calculated by integrating the speed, according to (3). Rotor speed is set to a constant value in each simulation, therefore the angle calculation is performed using a circuit composed of an independent current source with current equal to speed, and a capacitor with a capacitance of 1 F. The capacitor voltage is therefore obtained as the integral of current, i.e., speed, which corresponds to (3). Note that the angle is reset to zero whenever it reaches a multiple of 36° to comply with the range of flux and torque LUTs.
5. Current control is implemented by using two behavioural voltage sources and one current controlled voltage source. The behavioural sources V2 and V4 with the accompanying resistors act as gate driver circuits for the upper and lower MOSFET, hence the labels GU (Gate Upper) and GL (Gate Lower). The current controlled source provides a voltage equal to the SRM phase current which is fed to a Schmitt trigger. The parameters of the Schmitt trigger are the reference current and the hysteresis band, i.e., the corresponding voltage values V_t and V_h . According to Figure 10, the control logic can be formulated as follows:

- when the rotor angle is between 0° and 18° , voltages at GU and GL are equal to zero and both MOSFETs are switched off;
- when the rotor angle is between 18° and 36° , the voltage at GU equals 5 V and the upper MOSFET is switched on;
- when the rotor angle is between 18° and 36° , the voltage at GD depends on the current value:
 - when the input voltage of the Schmitt trigger exceeds $V_t + V_h$, the voltage at GL is set to zero and the lower MOSFET is switched off;

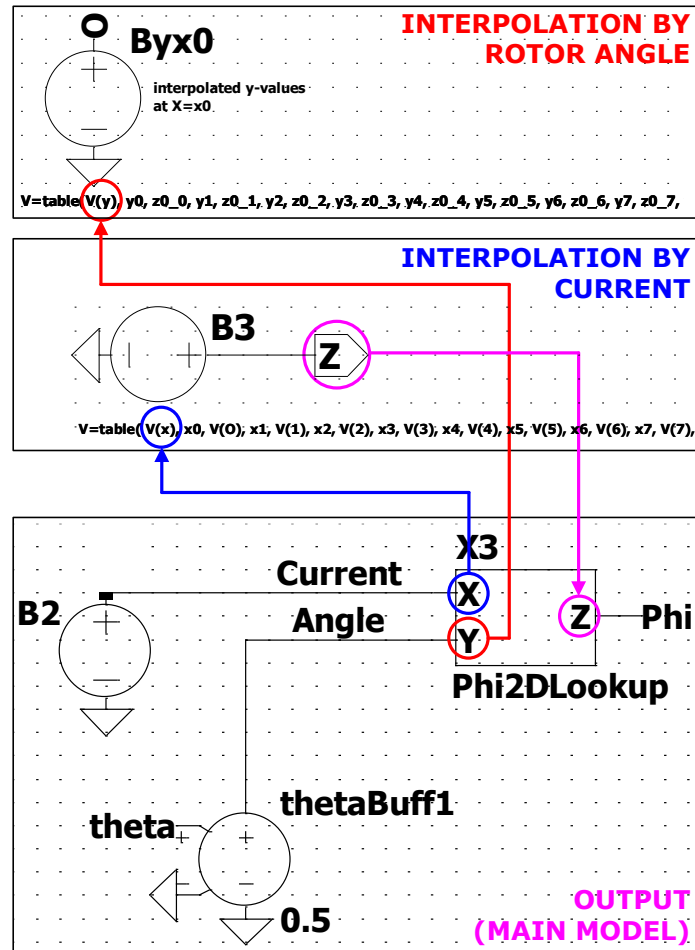


Figure 9: Flux LUT implementation in LTspice

- when the input voltage of the Schmitt trigger drops below $V_t - V_h$, the voltage at GL is set to 5 V and the lower MOSFET is switched on;

The described control algorithm is in accordance with the approach illustrated in Figure 4. Note that only the lower MOSFET is switched on and off during hysteresis control, leading to an uneven loss distribution between the switches.

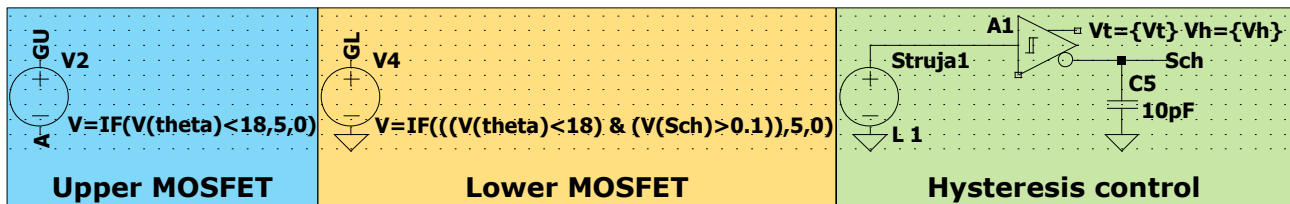


Figure 10: Components of the current control system in LTspice

6. Inclusion of libraries containing the applied switching devices (MOSFETs) and diodes, parameter and initial conditions definitions, and simulation settings are provided through the appropriate LTspice directives. The minimal time step was limited to $1 \mu\text{s}$, but a variable-step solver is used by LTspice to accommodate very fast transients. The step is reduced iteratively until the relative tolerance of 0.001 is satisfied. Such simulation settings would be appropriate even if more complex digital control algorithms were employed, as the sampling period of such algorithms is normally above $100 \mu\text{s}$.

Manipulation and execution of the model via the LTspice interface can be inconvenient, especially if multiple parameters need to be varied prior to each simulation. For instance, in the case of the SRM, it would be of interest to vary the reference current, operating speed, turn-on and turn-off angles, hysteresis band, etc. Moreover, if one were to use the same model structure to analyse a different SRM, modifications to LUTs would be quite cumbersome. For this reason, the dedicated PyLTSpice library [12] is used to interface Python with LTspice and perform these modifications automatically. Moreover, PyLTSpice allows the user to read output files and extract any waveforms of interest for further processing. Specifically, the input parameters varied in this paper are the reference current and operating speed. The waveforms of supply voltage, current, torque, and switching device losses are obtained and analysed.

The flowchart outlining the simulation procedure is displayed in Figure 11. The process is mostly self-explanatory, and the applied software tools used in each stage are denoted. More details can be found in the Python scripts given in the Appendix. Parts of the code using the PyLTSpice library are commented for better understanding.

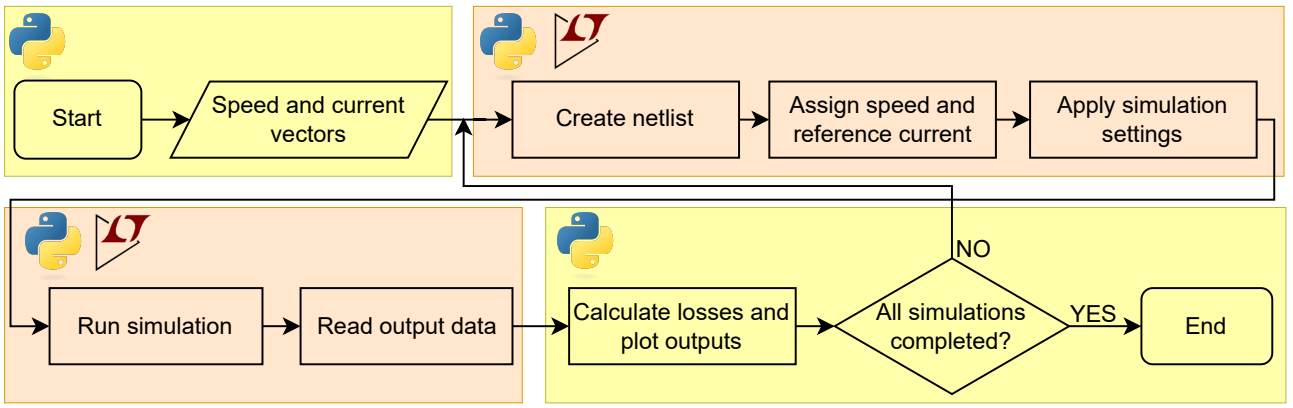


Figure 11: Flowchart of the simulation procedure using the Python/LTspice interface

4 Results

The SRM model is executed for various values of reference currents and operating speeds:

$$\begin{aligned} I_{ref} &\in \{2, 4, 6, 8, 10, 12, 14\} \text{ A,} \\ n &\in \{2000, 4000, 8000, 12000, 18000\} \text{ rpm.} \end{aligned} \quad (8)$$

The hysteresis band can also be modified in the Python script, but was kept constant for the purpose of this analysis.

Waveforms of SRM voltage, current, and torque are obtained for each operating mode. Additionally, switching device losses and converter efficiency are calculated. Simulated waveforms are shown for two characteristic scenarios:

- 1) low operating speed (2000 rpm) and low reference current (4 A),
- 2) high operating speed (12000 rpm) and moderate reference current (8 A).

The waveforms obtained for scenario 1 are displayed in Figure 12. The frequent voltage changes along with pronounced torque and current ripple indicate numerous switching operations during each electrical cycle. Hysteresis control is obviously active in this case, as the current oscillates around the reference value of 4 A. The MOSFET losses are therefore dominated by the switching loss component,

as observed in Figure 12d. Note that the losses are mostly attributed to the lower switch (M2) due to the hysteresis control implementation which does not employ the upper switch (M1). The MOSFET losses are calculated with great accuracy due to the use of detailed switching device models. As previously stated, high-fidelity modelling of semiconductor components is a major advantage offered by LTspice.

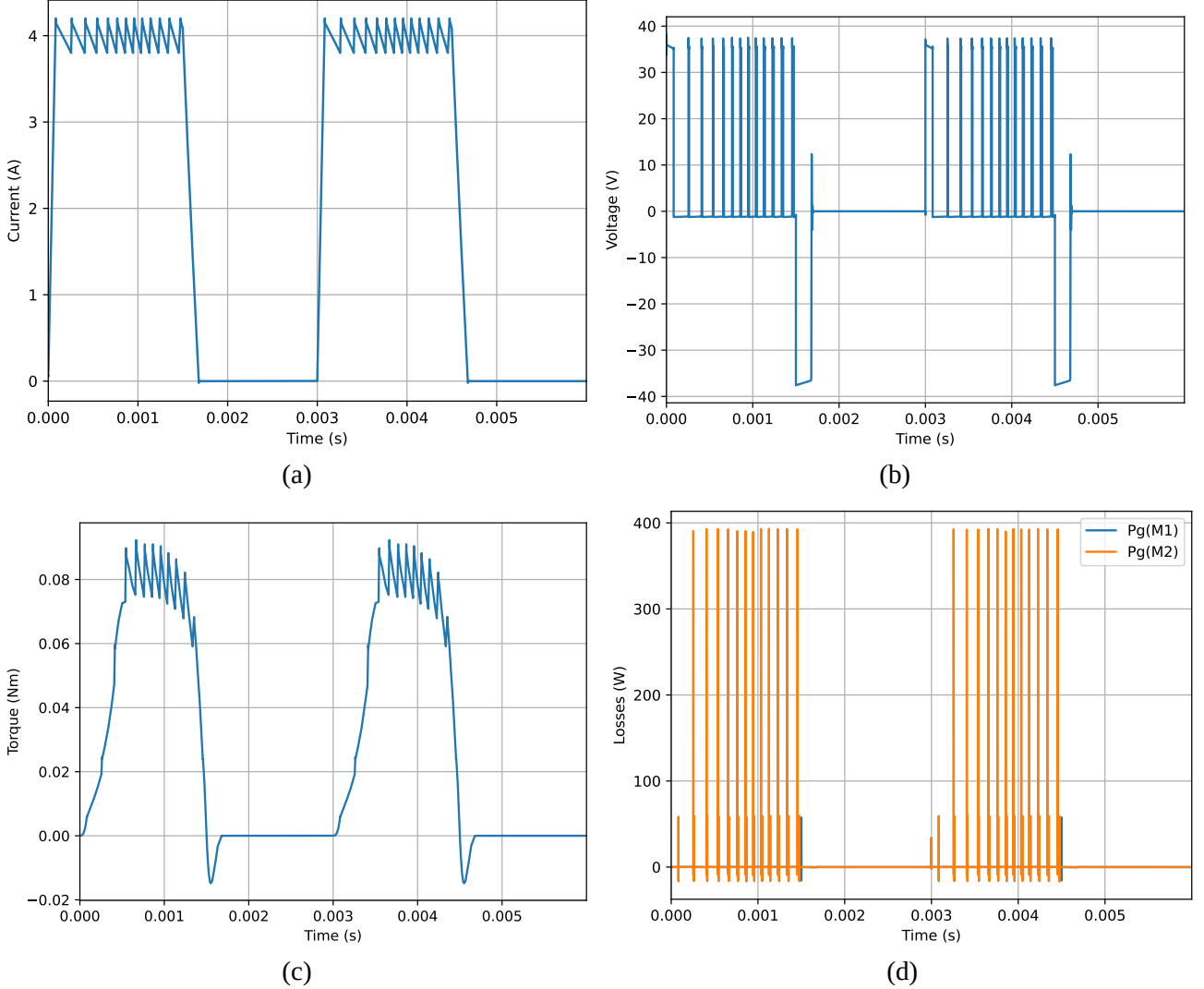


Figure 12: SRM drive waveforms obtained for the operating mode with reference current of 4 A and rotor speed of 2000 rpm: (a) current, (b) voltage, (c) torque, (d) MOSFET losses

The waveforms obtained for scenario 2 are displayed in Figure 13. Unlike the previous operating mode, the voltage now changes only twice per cycle between the positive and negative DC link voltage (zero voltage is never applied). Consequently, the torque and current waveforms are ripple-free. It can therefore be concluded that hysteresis control is not applied in this case. The current does not even approach the reference value of 8 A, therefore both switches remain turned on until the turn-off angle is reached. The MOSFET losses are now dominated by the conduction loss component, as seen in Figure 13d. High switching losses appear only twice per cycle, therefore their cumulative effect is much less pronounced compared to scenario 1. The losses are now evenly distributed between the two MOSFETs as they both conduct the same current and are switched twice in each cycle.

The input and output power of the SRM drive are measured and the drive efficiency is calculated for all operating modes. The obtained results are displayed in Table 1. While the efficiency calculation using

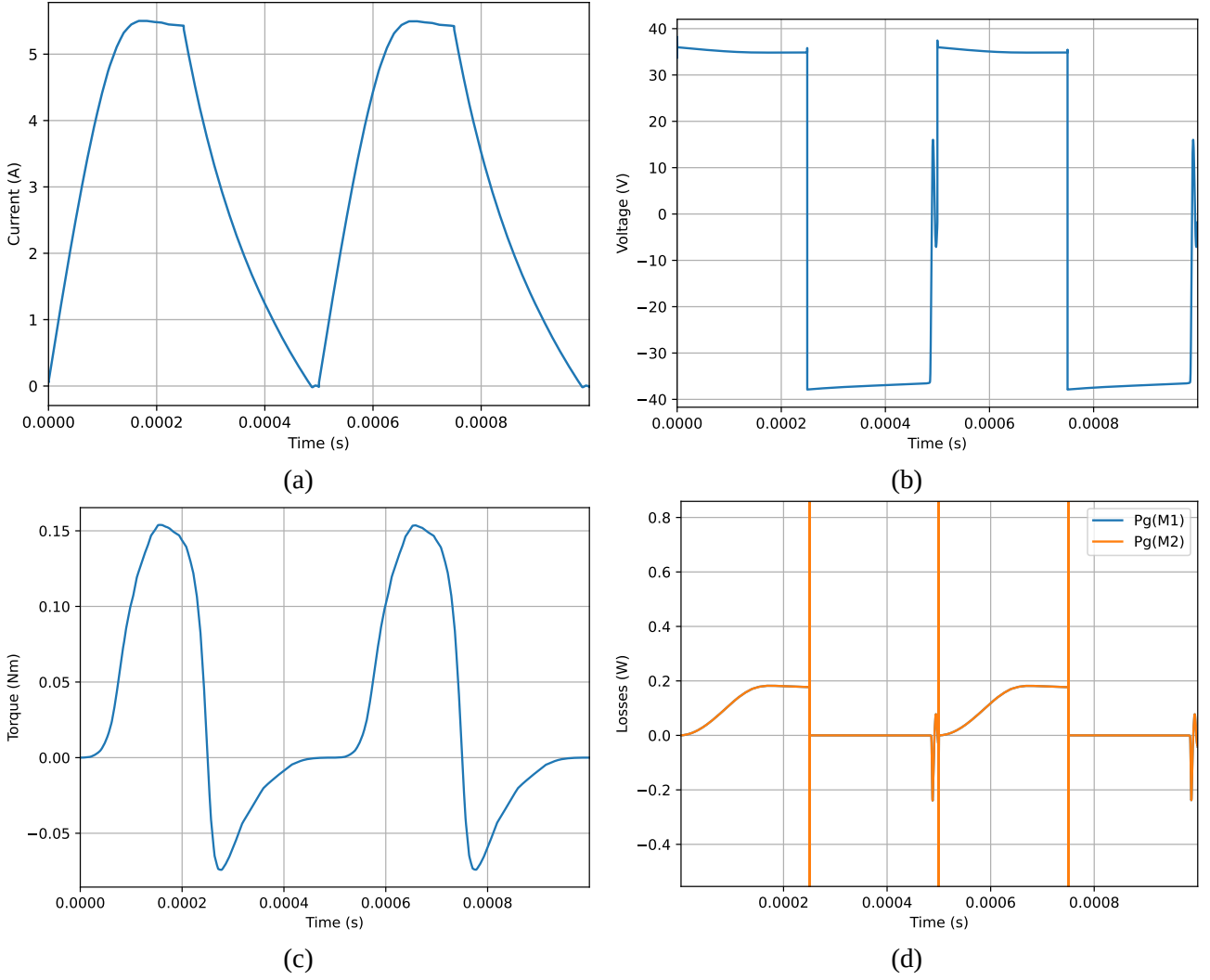


Figure 13: SRM drive waveforms obtained for the operating mode with reference current of 8 A and rotor speed of 12000 rpm: (a) current, (b) voltage, (c) torque, (d) MOSFET losses

the LTspice interface is a lengthy process, the same calculation is performed practically instantaneously using Python. Repeated values of efficiency are observed in the table, particularly at higher operating speeds. This is due to the fact that a certain level of reference current is never reached at elevated speeds. Therefore, any further increase of the current setpoint is non-effective and all waveforms remain unchanged. According to the results of Table 1, the current does not even reach 4 A at the very high speed of 18000 rpm. Note that the converter efficiency is obtained with high accuracy. However, only resistive SRM losses are modelled, therefore there a substantial error may occur when calculating SRM efficiency.

5 Conclusions

An approach for implementing an SRM drive model using exclusively open-source software was devised and described in this paper. The non-linear machine model was developed using the FEMM 4.2 platform in unison with GNU Octave. This model was then implemented in LTspice along with the converter model. An original approach for implementing the SRM model in LTspice was developed for this purpose. The simulation procedure was automated using a dedicated Python library, which allowed various operating modes to be analysed in a short amount of time. Furthermore, Python enables

Table 1: SRM drive efficiency with respect to reference current and rotor speed

	2000 rpm	4000 rpm	8000 rpm	12000 rpm	18000 rpm
2 A	47.2%	64.38%	79.12%	85.66%	90.6%
4 A	52.81%	69.98%	83.75%	89.52%	93.33%
6 A	55.66%	72.98%	86.18%	91.03%	93.33%
8 A	57.52%	74.92%	87.81%	91.03%	93.33%
10 A	58.65%	76.19%	87.81%	91.03%	93.33%
12 A	58.97%	76.73%	87.81%	91.03%	93.33%

seamless processing and comprehensible representation of the simulation results.

The work presented here is only a small part of what may be achieved by integrating various open-source software platforms. Octave was only used to modify the FEM model inputs, run the simulations, and collect and process the output data. However, the model construction can also be performed using Octave, which would allow different designs to be developed and analysed more efficiently. The LTspice model generation can be further simplified, e.g., by using Python to automate the LUT forming process. This would reduce the time required for model construction. A major issue with SRM drives is the control optimization for different operating conditions. The developed model can be used in combination with Python’s Optimize library to determine the optimal control inputs (most commonly, the turn-on and turn-off angles) based on different optimization criteria. Complete cross-platform integration can be attempted to fully automate the entire process, from creating the FEM model to processing the LTspice output data.

More complex digital control algorithms such as torque and flux control can be implemented using LTspice, but this process may prove tedious. Other open-source alternatives such as OpenModelica or SciLab are more convenient in terms of control system composition and SRM modelling, but do not incorporate detailed switching device models, which is a major downside compared to LTspice. The most attractive choice may be the recently introduced QSPICE, which is compatible with the same device libraries as LTspice, but intended specifically for mixed-signal simulations. Moreover, QSPICE allows the control algorithm to be programmed in C++, thereby generating experimental implementation-ready code.

References

- [1] K.M. Rahman, B. Fahimi, G. Suresh, A.V. Rajarathnam, and M. Ehsani. Advantages of switched reluctance motor applications to ev and hev: design and control issues. *IEEE Transactions on Industry Applications*, 36(1):111–121, 2000.
- [2] M. Krishnamurthy, C.S. Edrington, A. Emadi, P. Asadi, M. Ehsani, and B. Fahimi. Making the case for applications of switched reluctance motor technology in automotive products. *IEEE Transactions on Power Electronics*, 21(3):659–675, 2006.
- [3] Akira Chiba, Kyohei Kiyota, Nobukazu Hoshi, Masatsugu Takemoto, and Satoshi Ogasawara. Development of a rare-earth-free sr motor with high torque density for hybrid vehicles. *IEEE Transactions on Energy Conversion*, 30(1):175–182, 2015.
- [4] Zhi Yang, Fei Shang, Ian P. Brown, and Mahesh Krishnamurthy. Comparative study of interior permanent magnet, induction, and switched reluctance motor drives for ev and hev applications. *IEEE Transactions on Transportation Electrification*, 1(3):245–254, 2015.
- [5] T.J.E. Miller. *Switched Reluctance Motor and their Control*. Magna Physics Publishing, 1993.
- [6] S. Vukosavic and V.R. Stefanovic. Srm inverter topologies: a comparative evaluation. *IEEE Transactions on Industry Applications*, 27(6):1034–1047, 1991.
- [7] M. Barnes and C. Pollock. Power electronic converters for switched reluctance drives. *IEEE Transactions on Power Electronics*, 13(6):1100–1111, 1998.
- [8] Get up and running with ltspice. <https://www.analog.com/en/resources/analog-dialogue/articles/>

- [get-up-and-running-with-ltspice.html](http://www.femm.info/Archives/doc/manual.pdf). Accessed: 2024-30-8.
- [9] D. Meeker. Finite element method magnetics version 4.2 user's manual, May 2020. Available at <http://www.femm.info/Archives/doc/manual.pdf>.
- [10] Gnu octave 9.2.0 manual, 2024. Available at <https://docs.octave.org/interpreter/>.
- [11] D. Meeker. Finite element method magnetics: Octavefemm version 1.2 user's manual, Mar 2018. Available at <http://www.femm.info/Archives/doc/octavefemm.pdf>.
- [12] Pyltspice 5.3.2 documentation, 2024. Available at <https://pyltspice.readthedocs.io/en/latest/modules/modules.html>.

Appendix

The Python code used to manipulate the LTspice model of the SRM drive is given below. Parts of the code not specific to the Python–LTspice interface are omitted and replaced by vertical dots. User defined functions `read_data` and `calcVals` dedicated to reading, processing, and plotting output data, are given after the main program.

Main program

```

1 # Import PyLTSpice modules
2 from PyLTSpice import SimRunner
3 from PyLTSpice import SpiceEditor
4 from PyLTSpice import LTSpiceLogReader
5 from PyLTSpice import RawRead, Trace, RawWrite
6 .
7 .
8 .
9 # Import user defined functions
10 from functions import *
11 .
12 .
13 .
14 # Define vectors of rotating speeds and currents
15 nmeh_vec = [2000, 4000, 8000, 12000, 18000] # rotating speeds (rpm)
16 Iref_vec = [2, 4, 6, 8, 10, 12, 14] # reference currents (A)
17
18 # Run the simulations for all speed and ref. current values
19 for Iref in Iref_vec:
20     for nmeh in nmeh_vec:
21
22         # Select Spice model and generate netlist
23         LTC = SimRunner(output_folder='./temp'+str(nmeh), timeout=6000.0)
24         LTC.create_netlist('AHB.asc')
25         netlist = SpiceEditor('AHB.net')
26
27         # Calculate speed in (deg/s) and define the angular period
28         ndegs = nmeh*360.0/60.0 # speed (deg/s)
29         theta_per = 36.0 # angle corresponding to one electrical period (deg)
30         tsim = 2.0*theta_per/(nmeh*360.0/60.0) # simulation length = two periods
31         # netlist.set_parameters(res=0, cap=100e-6)
32
33         # Set the values of operating speed, reference current, and hysteresis band
34         netlist.set_component_value('I1', str(ndegs))
35         netlist.set_parameter('Vt', Iref)
36         netlist.set_parameter('Vh', 0.2)
37
38         # Simulation settings
39         netlist.add_instructions(
40             "; Simulation settings",
41             ".param run = 0",
42             ".tran 0 " + str(tsim) + " 0.1us 1e-6 uic" # makes a transient simulation
43         )
44         netlist.set_parameter('run', 0)
45
46         # Run the simulation
47         print("simulating test")
48         LTC.run(netlist, timeout=60.0)
49         netlist.reset_netlist()
50
51         # Sim Statistics
52         print('Successful/Total Simulations: ' + str(LTC.okSim) + '/' + str(LTC.runno))
53

```

```

54 # Read data from the output (*.raw) file
55 read_data("temp"+str(nmeh)+"/AHB_1.raw")
56
57 # Calculate the average losses of switching devices
58 gubici = calcVals("temp"+str(nmeh)+"/AHB_1.raw")
59 print(gubici)

```

Functions

```

1 #####
2 # read_data --> reads and plots the voltage, current, and torque waveforms
3 def read_data(filename):
4
5     # Import the RawRead module for reading output traces
6     from PyLTSpice import RawRead
7     from matplotlib import pyplot as plt
8
9     # Read the output traces
10    LTR = RawRead(filename)
11    IR1 = LTR.get_trace("I(R1)") # Current
12    Torque = LTR.get_trace("V(VTem)") # Torque
13    VSRM = LTR.get_trace("V(L1)") # Voltage
14
15    x = LTR.get_trace('time') # Gets the time axis
16    steps = LTR.get_steps()
17
18    # Plot the waveforms
19    for step in range(len(steps)):
20        # print(steps[step])
21        plt.figure()
22        plt.plot(x.get_wave(step), -IR1.get_wave(step), label=steps[step])
23        plt.grid()
24
25        plt.figure()
26        plt.plot(x.get_wave(step), Torque.get_wave(step), label=steps[step])
27        plt.grid()
28
29        plt.figure()
30        plt.plot(x.get_wave(step), VSRM.get_wave(step), label=steps[step])
31        plt.grid()
32
33    plt.show()
34
35    #####
36    # calcVals --> reads and plots switching device losses and calculates average losses
37    def calcVals(filename):
38
39        # Import the RawRead module for reading output traces
40        from PyLTSpice import RawRead
41        from matplotlib import pyplot as plt
42
43        # Read the output traces
44        LTR = RawRead(filename)
45        IM1 = LTR.get_trace("Ix(M1:DRAIN)")
46        IM2 = LTR.get_trace("Ix(M2:DRAIN)")
47        VM1H = LTR.get_trace("V(M1H)")
48        VM1L = LTR.get_trace("V(A)")
49        VM2 = LTR.get_trace("V(A)")
50
51        x = LTR.get_trace('time') # Gets the time axis
52        steps = LTR.get_steps()
53
54        # MOSFET losses
55        # Array initialization
56        PgM1 = []
57        PgM2 = []
58
59        # Loss calculation and data plotting
60        for step in range(len(steps)):
61            # Calculate losses at the current time step and add to array
62            PgM1.append ((VM1H.get_wave(step)-VM1L.get_wave(step)) * IM1.get_wave(step))
63            PgM2.append (VM2.get_wave(step) * IM2.get_wave(step))
64
65            # Plot the data
66            plt.figure()
67            plt.plot(x.get_wave(step), PgM1[step], label='Pg1')

```

```

68     plt.plot(x.get_wave(step), PgM2[step], label='Pg2')
69     plt.legend() # order a legend
70     plt.grid()
71     plt.show()
72
73     # Calculate the average losses over the simulation interval
74     PgM1_avg = sum(PgM1[0])/len(PgM1[0])
75     PgM2_avg = sum(PgM2[0])/len(PgM2[0])
76
77     # Return the average values of losses
78     return [PgM1_avg, PgM2_avg]

```