Politecnico di Torino

Master Thesis in Electrical Engineering

Advanced Modelling of Three-Phase and Multi-Three-Phase Motor Drives Under Faulty Conditions

Author

Matteo Sterpa

Supervised by
Prof. Gianmario Pellegrino
Dr. Paolo Pescetto
Dr. Andrei Bojoi

15 March 2024

In loving memory of my grandmother Milvia,
the Woman who instilled in me the true value of knowledge.

Acknowledgements

At this significant milestone in both my academic and personal journey, I wish to extend my deepest gratitude to those who made the completion of this thesis possible. The road to this point has been filled with challenges, learning, and unforgettable moments, all made easier by the support and love of many special people.

First and foremost, my heart is brimming with gratitude towards my dear parents. Words fall short of expressing how much your unconditional love, support and encouragement have meant to me over these years. Your belief in my abilities and your examples of dedication and strength have been the guiding light through moments of doubt and uncertainty. Thank you for always believing in me and allowing me to chase my dreams.

To my sister Chiara, a pillar of support and unconditional love: your words of encouragement and your constant presence gave me the strength to overcome obstacles and continue on my path. Your closeness has been a source of joy and inspiration to me. Thank you for always being by my side.

I cannot express enough gratitude to my girlfriend, Sofia, who has also stood by my side, supporting me through the intense periods of study and hard work. Sofia, your unwavering support, understanding, and love have been indispensable to me. Your encouragement and belief in my abilities have helped me navigate the toughest times. Thank you for being my partner, my confidante, and my source of inspiration.

I would like to express my sincere gratitude to my professors, Gianmario Pellegrino, Paolo Pescetto, and Andrei Bojoi. Your passion for teaching and your dedication have ignited in me a spark for the subjects you shared, guiding me in the choice of this thesis. Your availability, support, and valuable advice have made not only the realization of this work possible but have also enriched my educational journey. Thank you for believing in me and guiding me with wisdom and patience.

Last, but certainly not least, I want to thank my 'Green Friends': Vincenzo, Gianluca, Stefano, Filippo, Matteo, Francesco, Tiziano and Riccardo. Together, we shared this adventure in Turin, filled with studies, laughter, and unforgettable moments. Your friendship, mutual support, and the countless experiences we lived together have made these years unique and unforgettable. Thank you for being my family away from home.

To all of you, my most sincere thanks for enriching my life with your presence, your support, and your love. This thesis is also a fruit of your contribution, and for this, I will be eternally grateful.

Acronyms

MS Multi Stator

VSD Vector Space Decomposition

PMSM Permanent Magnet Synchronous Machine

SMPMSM Sourface Mounted Permanent Magnet Synchronous Machine

IM Induction Machine

IPM Internal Permanent Magnet

PM-SyR Permanent Magnet assisted Synchronous Reluctance

SYR Synchronous Reluctance

EMF Electro Motive Force

LUT Look Up Table

MTPV Maximum Torque Per Volt

HIL Hardware in the loop

FGPA Field Programmable Gate Array

MTPA Maximum Torque Per Ampere

SC Short Circuit

ASC Active Short Circuit

ITSC Inter Turn Short Circuit

IL Iron Losses

ISR Interrupt Service Routine

ISD Inverter Shut Down

OLF Open Leg Fault

OSF Open Switch Fault

Summary

The increasing complexity of electric propulsion systems requires significant advances in the design, analysis and control of electric motors. In particular, the reliability of the drive and its robustness under fault scenarios are of primary interest for the carmakers, in order to guarantee a safe operation of the vehicle in case of fault, either if this is voluntarily commanded by the vehicle control unit (e.g. active short circuit or open phase) or not. The main KPIs to evaluate the robustness of the drive w.r.t. the faults are the transient and steady-state torque and current and its eventual demagnetization. In this scenario, multi-three-phase motors emerge as a promising solution to overcome the limitations of traditional three-phase motors, offering improvements in efficiency, reliability and fault tolerance.

The design process of an electric motor begins with the definition of the electromagnetic design, using Finite Element Analysis to determine the motor's characteristics and performance parameters. This phase represents the primary intent behind the development of SyR-e (Synchronous Reluctance-evolution), an open source design environment developed by researchers at the Politecnico di Torino. Subsequently, based on the data obtained from the initial eMotor design, the SyreDrive extension was introduced. This add-on was designed to create offline dynamic simulation models for control system calibration and the capture of waveforms of the quantities of interest.

This thesis explores the development and implementation of unified circuit models in PLECS [7] of electric drives (e-Motor + inverters + machine control) applicable to three-phase and multi-three-phase motors through integrated use in SyreDrive, with a special focus on fault simulation and validation via Hardware-in-the-Loop (HIL) technologies.

The first chapter introduces the context and motivation behind this study, outlining the importance of advances in electrical engineering and the need for effective simulation and control tools. An overview of Syre, a state-of-the-art modelling software, is presented, along with the fundamentals of modelling and control of multiphase motors.

In the second chapter, we focus on the modelling of multi-three-phase motors in PLECS environment, illustrating modelling techniques, control strategies adopted and the development of a vectorised model for efficient and accurate simulation. Such vectorized model can be automatically generated regardless of the number of three-phase sets, thus enabling its integration with Syredrive. The importance of the decoupling algorithm and control strategy in handling the complexity of multiphase systems is also discussed.

The third chapter discusses the implementation of fault scenarios of three-phase and multi-three-phase motors, highlighting the ability to simulate realistic failures and evaluate mitigation strategies. This section emphasises the importance of fault tolerance and resilience in advanced electric propulsion systems.

Finally, the thesis explores the use of Hardware-in-the-Loop (HIL) technology for the validation of multi-phase motor models, using the Plexim RT-BOX. This approach enables accurate verification of motor performance and behaviour under real operating conditions, providing a valuable platform for testing and refining control and fault management strategies.

Through this work, the thesis aims to contribute to the advancement of electric motor technology by offering new perspectives on the design, control and validation of multi-phase motors, focusing on the modelling of faulty scenarios. The presented research and methodologies offer a step towards the realisation of more efficient, reliable and resilient electric propulsion systems, with significant implications for automotive, aerospace and other critical applications.

Contents

List of Figures

List of Tables

Chapter 1

Introduction, state of the art and motivations

In an ever-evolving world of technology, where energy efficiency and sustainability become increasingly pressing imperatives, electrical engineering is at the centre of revolutionary innovations. The design and optimisation of electric motors play a crucial role in this landscape, with applications ranging from automotive and industry to emerging sectors such as unmanned aerial vehicles and robotics. In this scenario, guaranteeing the reliability of the drive in case of fault occurrence is of top importance, and the development of an accurate method for predicting the machine's behaviour in faulty conditions is crucial for enabling a larger penetration in the market of high efficient electric drives. Among the various research frontiers, multi-three-phase motors emerge as a particularly promising field, offering innovative solutions to overcome the limitations of traditional three-phase motors.

Multi-three-phase motors, characterised by an electrical configuration that overcomes the canonical triad of phases, offer significant improvements in terms of torque harmonic reduction, increased fault tolerance and optimised operating efficiency. However, the inherent complexity of these systems requires sophisticated modelling and control approaches capable of capturing the nuances of electromagnetic behaviour and effectively managing control dynamics.

This thesis fits into this innovative context, aiming to explore the potential of multi-phase motors through an integrated approach that combines advanced modelling techniques with fault simulation and validation using Hardware-in-the-Loop (HIL) technologies. Through the use of state-of-the-art tools such as SyreDrive and PLECS, this work aims not only to expand the theoretical understanding of such motors but also to provide practical solutions for their implementation and optimisation.

1.1 Background on Synchronous Electric Motors

The exploration of electric motors, particularly emphasizing Permanent Magnet Synchronous Machines (PMSM), Permanent Magnet assisted Synchronous Reluctance (PM-SyR) and Synchronous Reluctance (SyR) machines, is a fundamental aspect of this thesis. The definition of the dq-axes in the rotor reference frame is critical for motor modeling and follows two distinct conventions.

1.1.1 Three Phase Motors

Three-phase synchronous motors are pivotal in the field of electric drives, thanks to their efficiency and robustness in converting electrical energy into mechanical motion.

The abc model

The abc dynamic model, or 'phase domain' model, delineates the motor in abc phase coordinates, offering insights into the motor's operational dynamics. The voltage equation in this model is given by:

$$\mathbf{v_{abc}} = R_s \cdot \mathbf{i_{abc}} + \frac{d\lambda_{\mathbf{abc}}}{dt} \tag{1.1}$$

Where R_s is the stator winding resistance. The flux linkage equation is formulated as:

$$\lambda_{abc} = \mathbf{L}(\theta_\mathbf{r}) \cdot \mathbf{i}_{abc} + \lambda_\mathbf{m}(\theta_\mathbf{r}) \tag{1.2}$$

for SPM, IPM, and PM-SyR motors. For SyR machines, it simplifies to:

$$\lambda_{\mathbf{abc}} = \mathbf{L}(\theta_\mathbf{r}) \cdot \mathbf{i_{abc}} \tag{1.3}$$

where $L(\theta_r)$ is the self and mutual stator inductance matrix, dependent on the mechanical rotor position θ_r.

The torque equation is expressed as:

$$T_{em} = \frac{p}{2}\left(\mathbf{i}_{abc}^T \cdot \frac{\partial \mathbf{L}(\theta_r)}{\partial \theta_r} \cdot \mathbf{i}_{abc} + \mathbf{i}_{abc}^T \cdot \lambda_m(\theta_r)\right) \tag{1.4}$$

for specific motor types, adjusting for SyR machines where the inductance matrix's time variance introduces complexity, necessitating streamlined models.

The dq model

The dq dynamic model encapsulates the motor's behavior in the rotating dq frame, yielding:

$$\mathbf{v}_{dq} = R_s \cdot \mathbf{i}_{dq} + \frac{d\lambda_{dq}}{dt} + [\mathbf{J}] \cdot \omega \cdot \lambda_{dq} \tag{1.5}$$

where $[J] = \left[\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right]$, ω represents the electrical frequency.
The magnetic flux equations are:

$$\lambda_{dq} = \begin{bmatrix} L_{dd} & L_{dq} \\ L_{qd} & L_{qq} \end{bmatrix} \cdot \mathbf{i}_{dq} + \begin{bmatrix} \lambda_m \\ 0 \end{bmatrix} \tag{1.6}$$

tailored for SPM and IPM, with adjustments for other types.

The unified electromagnetic torque expression is:

$$T_{em} = \frac{3}{2} \cdot p \cdot (\lambda_d \cdot i_q - \lambda_q \cdot i_d) \tag{1.7}$$

highlighting the direct influence of dq components on motor torque. This advanced model facilitates precise control and efficiency optimization in modern electric drives.

1.1.2 Multiphase Motors

Multiphase motors represent a class of electrical machines distinguished by the adoption of a number of phases greater than three. This characteristic confers considerable advantages, including smoother operation and reduced torque pulsation, in addition to an increase in redundancy that translates into greater fault tolerance. In this chapter, we will explore the modeling of multiphase motors, focusing on the fundamental principles that guide their design and operation. Among the possible multiphase topologies, the multi-three-phase machines will be considered

As can be understood from [2], the operation of multiphase motors is based on the understanding of their symmetric and asymmetric configurations, determined by the electrical phase shift between adjacent 3-phaese sets. In a symmetric machine, this phase shift is equal to $120°/n_{ph}$ electrical degrees, where n_{3ph} indicates the number of phases. Asymmetric machines, on the other hand, present a phase shift between the three-phase sets equal to $60°/n_{ph}$. In both cases, the phases shift between phases belonging to the same three-phase set is 120°.

The technical literature provides numerous configurations of multiphase machines, including Induction Motors (IM), Permanent Magnet Machines (PM), Brushless DC Motors (BLDC) and many more that significantly affects the motor's performance, particularly regarding Joule losses and fault tolerance.

The most widespread drive configurations in the academic field are:

1. A conventional multiphase system with a single neutral point per machine;

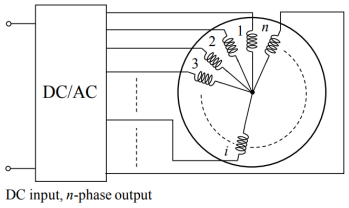

Figure 1.1: Multiphase machine - Single neutral layout [2]

2. Groups of independent single-phase units;

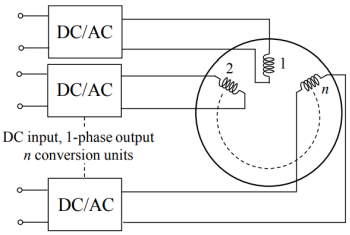

Figure 1.2: Multiphase machine - Single phase layout [2]

3. Groups of independent three-phase units.

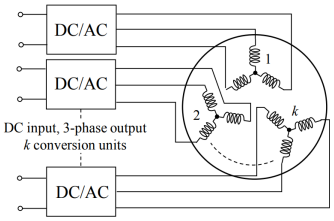

Figure 1.3: Multiphase machine - Multi-Three phase layout [2]

Only the last two configurations prove suitable for ensuring reliable operation in applications where safety is a priority. The adoption of independent single-phase units allows each phase to be powered by its own single-phase inverter.
The configuration based on independent three-phase units, also called multi-three-phase, is the main subject of this study, and it is suitable when the number of phases is a multiple of three and the stator presents separate three-phase windings with isolated neutral points. Each three-phase group is managed by an autonomous three-phase inverter. This approach, although less favorable for fault tolerance compared to single-phase units, facilitates the reduction of size, cost, and design time through the use of consolidated three-phase power modules. Moreover, it tends to simplify control due to the fewer number of independent currents to manage, compared to the solution based on single-phase units.

1.2 Multi-triphase Machines Modeling

In this section, we will delve into various modeling techniques employed for multi-triphase machines, aiming to understand their complex behavior and facilitate the design of efficient control strategies.

1.2.1 Vector Space Decomposition Technique

The Vector Space Decomposition technique [2] [10] is a fundamental method for modeling multi-triphase machines. Based on symmetrical components theory, VSD decomposes the machine's variables into orthogonal subspaces, simplifying analysis and control. By focusing on flux and torque production, VSD provides valuable insights into electromechanical energy conversion, akin to conventional three-phase machines.

The VSD approach offers versatility and generality, making it applicable to a wide range of multi-triphase machine configurations. It simplifies the analysis by decomposing the machine's original

space into decoupled subspaces, where the primary subspace typically represents the flux- and torque-producing components. Other subspaces accommodate harmonics and zero-sequence components, ensuring accurate modeling without compromising computational efficiency.

In recent years, advancements in VSD modeling have focused on addressing challenges associated with asymmetrical winding configurations. Various techniques, including advanced decoupling algorithms and adaptive control strategies, have been proposed to enhance the accuracy and robustness of VSD-based models for asymmetrical machines.

According to VSD, the main subspace of the multiphase syncronous motor in a generic stationary rotating frame x, y is governed by the equations:

$$\begin{cases} \overline{\mathbf{v}}_{s,xy,m} = [\mathbf{R}_s] \cdot \bar{\mathbf{i}}_{s,xy,m} + \frac{d}{dt}\overline{\lambda}_{s,xy,m} \\ \overline{\lambda}_{s,xy,m} = [\mathbf{L}_s] \cdot \bar{\mathbf{i}}_{s,xy,m} + \overline{\lambda}_{m,xy,m} \end{cases} \tag{1.8}$$

on the other hand, the harmonic subspaces refers to ther own $h = (n-1)$ laws:

$$\begin{cases} \overline{\mathbf{v}}_{s,xy,h} = [\mathbf{R}_s] \cdot \bar{\mathbf{i}}_{s,xy,m} + \frac{d}{dt}\overline{\lambda}_{s,xy,m} \\ \overline{\lambda}_{s,xy,m} = [\mathbf{L}_s] \cdot \bar{\mathbf{i}}_{s,xy,m} + \overline{\lambda}_{m,xy,m} \end{cases} \tag{1.9}$$

Regarding the n zero-sequence components, they can be ingored because every set of three-phase windings possesses its own neutral point, which is isolated from the others.

Using this method, the generated torque is calculated the cross-product of fluxes and currents in the main subspace, the only one to contribute on the energy conversion.

$$T = \frac{3 \cdot n}{2} \cdot p \cdot (\overline{\lambda}_{s,dq,k} \wedge \bar{\mathbf{i}}_{s,dq,k}) \tag{1.10}$$

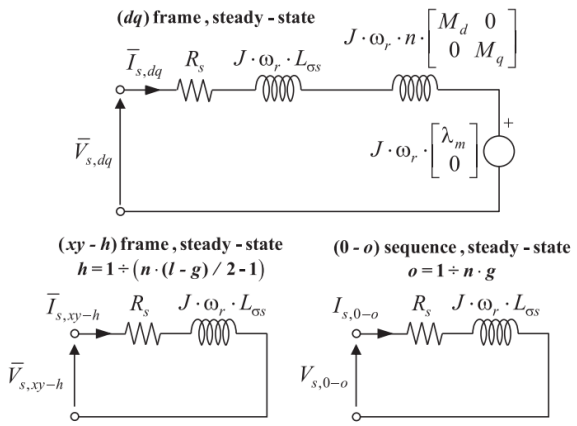

Figure 1.4: VSD machine model of a PMSM motor [10]

1.2.2 Multi Stator

The Multi Stator approach [2] [10] is another significant modeling technique for multi-triphase machines. In the MS method, the multi-triphase machine is conceptualized as the sum of several independent machines, each with its own Clarke transformation. This approach enables comprehensive

control of the motor, even if the structure lacks the standard displacement angle for a specific number of phases.
MS-based modeling provides detailed insights into the contributions of each individual stator set to the machine's flux and torque characteristics. By describing each stator in the same dq frame, MS aligns the d-axis with the main flux direction for all three-phase systems. However, addressing the natural coupling between different three-phase sets requires sophisticated decoupling algorithms to improve motor dynamics and usability.

The modeling analysis of this type starts from the voltage equations of the k-th $k = [1 : n_{\text{set}}]$ set of phases. For each of them, the following relation holds:

$$\overline{\mathbf{v}}_{s,abc,k} = [\mathbf{R_s}] \cdot \bar{\mathbf{i}}_{s,abc,k} + \frac{d}{dt}\overline{\lambda}_{s,abc,k} \tag{1.11}$$

Similarly, the magnetic equations of a set of phases can be written as follows:

$$\overline{\lambda}_{s,abc,k} = [\mathbf{L_s}] \cdot \bar{\mathbf{i}}_{s,abc,k} + \sum_{z=1}^{n}([\mathbf{M_{sk-sz}}] \cdot \bar{\mathbf{i}}_{s,abc,z}) + \overline{\lambda}_{m,abc,k} \tag{1.12}$$

Where $\overline{\lambda}_{m,abc,k}$ represents the rotor magnet flux linkage, with the further assumption of magnetic linearity.

$$\overline{\lambda}_{m,abc,k} = \begin{bmatrix} \lambda_{ma,k} \\ \lambda_{mb,k} \\ \lambda_{mc,k} \end{bmatrix} \tag{1.13}$$

The natural continuation of the study occurs by reporting the above-mentioned quantities in a stationary two-phase reference frame, through the use of the classical Clarke transform of three-phase systems.

$$\overline{\mathbf{v}}_{s,\alpha\beta,k} = [\mathbf{R_s}] \cdot \bar{\mathbf{i}}_{s,\alpha\beta,k} + \frac{d}{dt}\overline{\lambda}_{s,\alpha\beta,k} \tag{1.14}$$

$$\overline{\lambda}_{s,\alpha\beta,k} = [\mathbf{L_s}] \cdot \bar{\mathbf{i}}_{s,\alpha\beta,k} + [\mathbf{M(2\theta_r)}] \cdot \sum_{z=1}^{n} \bar{\mathbf{i}}_{s,\alpha\beta,z} + \overline{\lambda}_{m,\alpha\beta,k} \tag{1.15}$$

Where the matrix $[M(2\theta_r)]$ characterizes the rotor anisotropy percentage in the following way:

$$[\mathbf{M(2\theta_r)}] = \frac{3 \cdot M_I}{2} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \frac{3 \cdot M_A}{2} \cdot \begin{bmatrix} \cos(2\theta_r) & \sin(2\theta_r) \\ \sin(2\theta_r) & -\cos(2\theta_r) \end{bmatrix} \tag{1.16}$$

With an isotropic rotor (for example, SMPMSM technology), the anisotropy term will be identically null.
Finally, the magnetic and voltage equations in the d, q reference of the k-th set of phases will be:

$$\overline{v}_{s,dq,k} = R_s \cdot \bar{i}_{s,dq,k} + \frac{d\overline{\lambda}_{s,dq,k}}{dt} + j \cdot \omega \cdot \overline{\lambda}_{s,dq,k} \tag{1.17}$$

$$\overline{\lambda}_{\mathbf{s,dq,k}} = \begin{bmatrix} L_d & 0 \\ 0 & L_q \end{bmatrix} \cdot \mathbf{overlinei_{s,dq,k}} + \frac{1}{n} \cdot \begin{bmatrix} M_d & 0 \\ 0 & M_q \end{bmatrix} \sum_{z=1}^{n} \bar{\mathbf{i}}_{\mathbf{s,dq,z}} + \begin{bmatrix} \lambda_m \\ 0 \end{bmatrix} \tag{1.18}$$

Whose d, q axis inductances are defined as

$$L_d = \left(L_{\sigma,s} + \frac{M_d}{n}\right), \quad L_q = \left(L_{\sigma,s} + \frac{M_q}{n}\right) \tag{1.19}$$

With this approach, the torque production can be computed as the sum of the single k-th contribution of each set as follows

$$T = \sum_{k=1}^{n} T_k = \frac{3}{2} \cdot p \cdot \sum_{k=1}^{n} (\bar{\lambda}_{\mathbf{s,dq,k}} \wedge \bar{\mathbf{i}}_{\mathbf{s,dq,k}}) \tag{1.20}$$

In figure -1.5, it is possible to observe a circuit diagram of the machine in the rotating d, q reference system of a generic synchronous machine (IPM, PMSM, PM-SyR, SYR)

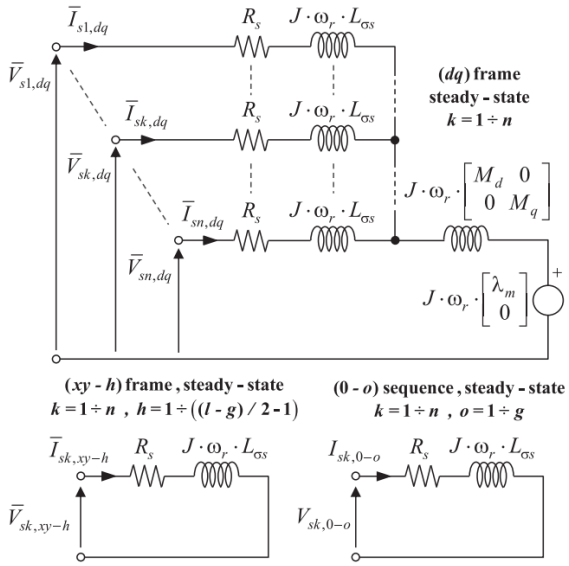

Figure 1.5: MS machine model of a PMSM Machine[10]

Recent developments in MS modeling have focused on enhancing fault tolerance and control performance. Advanced MS-based control schemes offer seamless reconfiguration after open three-phase fault events, ensuring uninterrupted operation and enhancing system reliability. Moreover, MS-based control strategies facilitate deep flux-weakening operation with MTPV, enabling optimal performance across a wide range of operating conditions.

In summary, both VSD and MS approaches play critical roles in modeling and controlling multi-triphase machines. While VSD offers versatility and generality, MS provides detailed insights into machine behavior and facilitates advanced control strategies.

1.3 Fault Events

This section delves into the simulation of fault events in multi-three-phase electrical drives to assess their resilience under various fault conditions. The simulation aims to highlight the effectiveness of the designed fault mitigation strategies, providing a robust framework for understanding and managing faults in electrical drives.

1.3.1 Three-Phase Unit Faults

This subsection focuses on faults affecting a three-phase unit of the motor. These faults, are also known as simmetric faults in the three phase motor field.

Three-phase Short Circuit / Active Short Circuit

In classical fault studies for electric motors the Threephase short circuit stands as one of the most relevant academic study to understand the machine behaviour.This fault occurs due to direct contact among the motor phase terminals. More recently, what was once considered a disastrous contingency has been well integrated into the protection systems of synchronous motors under the name of Active Short Circuit.
In fact, in ASC the three-phase short circuit is implemented by appropriately controlling the inverter switches and is used as a safe shutdown scheme to bring the motor into a safe state in PMSM e-motors. In some cases ASC is preferred to the canonical inverter shut off, especialli at high speed and flux weakening operating point where this can cause UGO.
An active short circuit involves the direct electrical shorting of a motor's three-phase unit through simultaneous activation of all low-side (or high-side, like figure 1.6) switches in the case of a two-level inverter, while the opposite switches remain off

For what concerns multi-three-phase motors, in contrast to single three-phase motors,if this event appears as a fault, it can be mitigated by the redundancy of the different sets contributing to the torque production.

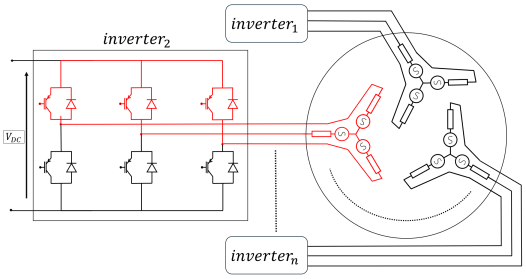

Figure 1.6: Active Short Circuit

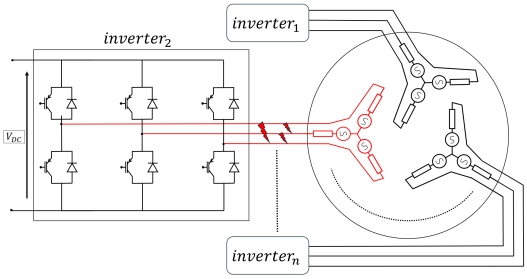

Figure 1.7: Three Phase SC

However, as noted in [13], multi-three-phase motors the n_{set} isolated subsystems may have magnetic flux coupling through a shared flux path, known as mutual inductance, between the subsystems. This mutual inductance allows the current flowing in one of the subsystems to generate an EMF in the other subsystem, making the subsystems interdependent. Therefore, it is crucial to study the influence of this mutual inductance on the overall machine operation, especially under short-circuit conditions, in order to optimise the selection of the most suitable short-circuit protection method

Inverter Shut Down

The open phase fault represents an accidental (or volunteer) disconnection of a motor's three-phase unit, typically resulting from an inverter lock-up where all switches are turned off.
The risk associated with this fault, especially in PMSM motors, lies in UGO.

The *Uncontrolled Generator Operation* (UGO) describes a critical condition that occurs in inverter-powered permanent-magnet synchronous motors, particularly in scenarios where, following a sudden inverter shut-off (*inverter shut off*), the motor continues to operate without regulatory control. This phenomenon occurs when the motor, operating at high speed, loses power from the inverter. If the electromotive force (EMF) generated by the permanent magnets, minus the resistive drops on the windings, exceeds the sum of DC bus voltage of the inverter and the diode threshold voltage, the diodes in the inverter may go in conduction state.

As a result, the motor-inverter system turns into an equivalent generator with a three-phase diode bridge, injecting current back to the DC bus. This situation can cause a dangerous overvoltage at the DC bus, endangering the physical integrity of the capacitor, which may even explode under extreme conditions.

To mitigate the risks associated with UGO, it is crucial to implement specific protection systems, such as advanced DC bus voltage monitoring to intervene quickly in the event of overvoltage, the use of electronically controlled discharge resistors to dissipate excess energy, and the adoption of sophisticated inverter control algorithms to prevent potentially dangerous situations. These measures are essential to maintain operational safety and prevent damage to hardware in systems using inverter-driven permanent magnet synchronous motor technology.

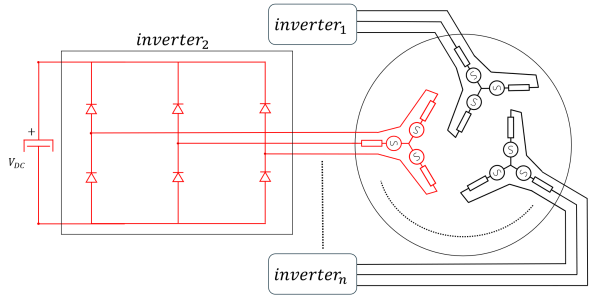

Figure 1.8: UGO in multi-threephase motors

1.3.2 Asymmetric Faults

If the three-phase short circuit and open circuit faults maintain the electromagnetic symmetry of the machine, the faults events concerning individual phases produce an unbalance in the magnetic model of the machine. Similar to three-phase faults in terms of voltage and current considerations, phase faults introduce system imbalances, leading to unbalanced voltage and current components.

1.3.3 Open Phase

A phase-opening fault, commonly known as phase loss or single-phase fault, occurs in three-phase electric motors when one of the three power supply phases is interrupted. This can occur for various reasons, including leg switch off, blown fuses, broken cables, faulty contactors or loose connections. Such a failure prevents the motor from operating as intended, affecting its performance and potentially

causing damage.
The loss of a phase significantly alters motor operation. Without a balanced three-phase supply, the motor cannot develop an efficient rotating magnetic field, resulting in reduced torque and increased overheating due to irregular currents. If not addressed, this problem can lead to permanent damage to the motor.
Timely diagnosis of a phase-opening fault is crucial. Modern motor protection systems include sensors and monitoring circuits capable of detecting such faults, protecting the motor from further damage through alarm signals or by shutting down the power supply.
Operating an engine under phase-loss conditions for prolonged periods increases the risk of catastrophic failure. This not only implies reduced efficiency, but also potential mechanical damage and the need for costly repairs or replacements.
To mitigate the risks associated with phase failure, it is important to carry out regular maintenance of the electrical system and install appropriate protective devices. These steps are essential to ensure the reliability and safety of three-phase electric motors in industrial use.

1.3.4 Open Switch Faults

An open switch fault within a two-level inverter signifies a malfunction where one (or more) semiconductor switches (such as IGBTs or MOSFETs) fail to close or conduct as intended during regular operation.

Several factors can contribute to the occurrence of an open switch fault:

Component defects, such as physical damage or wear within the switches, can hinder their proper functioning.
Driving errors may result from inadequate driving signals that fail to activate the switches correctly.
Power supply issues, such as interruptions or instability in the power supply to the driver circuits, can impede switch operation.
Overtemperature conditions can damage the circuitry or alter switch behavior.
The consequences of an open switch failure can be significant. Inefficient switching can lead to higher energy losses, impacting the overall efficiency of the inverter. Moreover, it may result in inadequate power supply to connected loads, potentially causing interruptions or malfunctioning of equipment.
Additionally, irregular switching can generate unwanted harmonics and electrical noise, which could damage other electrical devices. The stress on adjacent switches to compensate for the malfunctioning one can lead to thermal damage and reduce the overall lifespan of the inverter.

To address the risks associated with open switch failures, modern inverters are equipped with advanced monitoring and diagnostic systems. These systems monitor currents, voltages, and temperatures, allowing for the rapid detection of anomalies in switch operation. Upon identification of an open switch fault, the damaged component must be promptly replaced or repaired to restore proper inverter functionality. Regular maintenance, along with careful design considerations such as safety margins and effective cooling systems, can help mitigate the likelihood of such failures.

In summary, open switch failures in two-level inverters represent a serious issue that can compromise performance and reliability, with potential negative impacts on the entire system. Understanding the causes, consequences, and mitigation strategies of such failures is crucial for ensuring the safe and efficient operation of inverter-based systems.

1.3.5 Inter Turn Short Circuit

The interturn short circuit fault involves a short between turns within the same winding, critically impacting the motor's electromagnetic field and torque generation. Unlike other faults simulated at the inverter level, the interturn short circuit is modeled within the motor model, highlighting its unique impact on motor performance and the necessity of detection and mitigation strategies.

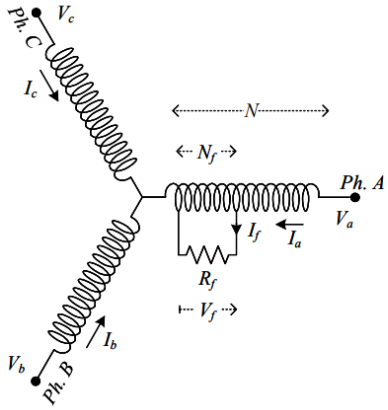

Figure 1.9: Inter-turn short circuit in phase a of a three-phase motor winding [15]

From [15], it is understood that ITSC manifest due to the degradation of insulation in the stator winding, potentially evolving into more severe faults unless promptly detected and isolated. These faults are primarily caused by high mechanical, thermal, and electrical stresses, with the application of wide bandgap devices exacerbating the situation through high switching frequencies and voltage gradients. Research indicates that stator winding faults account for approximately 21% to 37% of motor failures, with a significant portion being ITSCs, which begin as minor electrical contacts between turns, eventually leading to significant damage including demagnetization of the permanent magnet.

For the modeling of ITSC in a three-phase machine, the voltage behind reactance (VBR) approach is adopted, representing each phase of the Permanent Magnet Synchronous Motor (PMSM) as controlled voltage sources in series with an RL branch. The fault ratio μ is defined as the ratio of the number of shorted turns to the total number of turns in one phase ($\frac{N_f}{N}$).

abc model

For this study is used the approach presented in [8]: in the abc reference frame, the ISCF model assumes the fault in phase a there the branch is splitted as follows

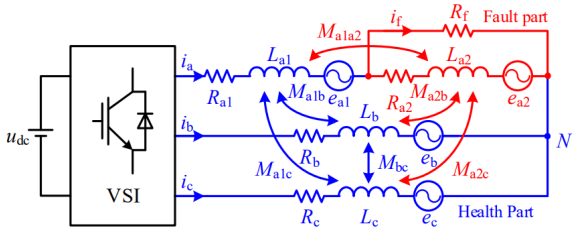

Figure 1.10: Inter-turn short circuit abc-VBR model [8]

The faulted branch is divided into a remaining healthy part (illustrated in blue) and a faulty one

(depicted in red). The healthy branch is characterized by a $(1-\mu)$ portion of the winding, indicating that this part possesses $R_{a1} = (1-\mu)R_s$ and $L_{a1} = (1-\mu)L_s$. Conversely, the faulted windings are characterized by $R_{a2} = \mu R_s$ and $L_{a2} = \mu L_s$. Hence, the model can be encapsulated within the following 4x4 state-space model:

The model presented offers a comprehensive representation of the dynamics within a faulted electric motor system, specifically addressing the impact of inter-turn short circuits (ITSC) on motor performance. Each equation within this model elucidates different aspects of the system under fault conditions:

The voltage vector $[V_s]$ represents the individual phase voltages, including the split voltage components V_{a1} and V_{a2} for the faulted phase 'a', along with the healthy phases V_b and V_c:

$$[\mathbf{V_s}] = [V_{a1} \quad V_{a2} \quad V_b \quad V_c] \tag{1.21}$$

The same characteristic is found in the currents, where the term $(i_a - i_f)$ models the impact of the fault current i_f on the faulted phase 'a':

$$[\mathbf{i_s}] = [i_a \quad (i_a - i_f) \quad i_b \quad i_c] \tag{1.22}$$

Then, the matrices $[\mathbf{R_{sf}}]$ and $[\mathbf{L_{sf}}]$, accounts for the resistance and inductance variations due to the fault. The resistance matrix $[R_{sf}]$ varies with the fault ratio μ, affecting the electrical characteristics of both faulted and healthy windings:

$$[\mathbf{R_{sf}}] = R_s \cdot \begin{bmatrix} (1-\mu) & 0 & 0 & 0 \\ 0 & \mu & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.23}$$

The inductance matrix $[L_{sf}]$, incorporates the mutual inductance M and adjusted self-inductance L, capturing the electromagnetic interactions within the motor under fault conditions:

$$[\mathbf{L_{sf}}] = \begin{bmatrix} (1-\mu)^2 L & (1-\mu)\mu L & (1-\mu)M & (1-\mu)M \\ (1-\mu)\mu L & \mu^2 L & \mu M & \mu M \\ (1-\mu)M & \mu M & L & M \\ (1-\mu)M & \mu M & M & L \end{bmatrix} \tag{1.24}$$

The magnetic linkage vector influenced by the fault, relating the magnetic linkage to the mechanical position θ and the fault ratio μ, providing insights into the electromagnetic coupling:

$$[\mathbf{\Lambda_{mf}}] = \begin{bmatrix} \Lambda_{a1} \\ \Lambda_{a2} \\ \Lambda_b \\ \Lambda_c \end{bmatrix} = \Lambda_m \cdot \begin{bmatrix} (1-\mu)cos(\theta) \\ \mu cos(\theta) \\ cos(\theta - \frac{2\pi}{3}) \\ cos(\theta + \frac{2\pi}{3}) \end{bmatrix} \tag{1.25}$$

In conclusion, the model is condensed inside its voltage and magnetic equations:

$$\begin{cases} [\mathbf{V_{a1a2bc}}] = [\mathbf{R_{sf}}] \cdot [\mathbf{i_{a1a2bc}}] + [\mathbf{L_{sf}}] \cdot \frac{d[\mathbf{i_{a1a2bc}}]}{dt} + \frac{d[\mathbf{\Phi_{a1a2bc}}]}{dt} \\ \\ [\mathbf{\Phi_{a1a2bc}}] = [\mathbf{L_{sf}}] \cdot [\mathbf{i_{a1a2bc}}] + [\mathbf{\Lambda_{mf}}] \end{cases} \tag{1.26}$$

This model meticulously captures the nuanced effects of ITSCs on the electrical and electromagnetic properties of the motor, providing a comprehensive framework for this kind of fault scenario.

1.4 Syre environment

Figure 1.11: Syre logo

Syr-e

Syre, short for 'Synchronous Reluctance - evolution', is an open-source design environment developed specifically for the modelling, analysis and optimisation of electric motors, with a particular focus on synchronous reluctance machines (SyR), including permanent magnet assisted machines (PM-SyR), as well as internal permanent magnet (IPM) and surface mounted motors (SPM). Created to run within the MATLAB/Octave ecosystem, Syre stands out for its ability to combine detailed electromagnetic design analyses with optimisation techniques based on finite element simulations (FEM).

Key features:

- **Integrated Environment:** Syre provides an intuitive interface for the design and analysis of electric motors, integrating directly with FEM simulation software such as FEMM (Finite Element Method Magnetics), facilitating accurate simulations of magnetic behaviour and motor performance under various loads and operating conditions.

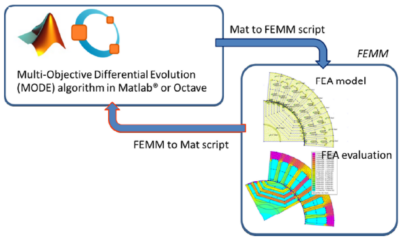

Figure 1.12: Syre Flowchart

- **Analysis and Optimisation:** Through the use of optimisation algorithms and the development of motor-specific design equations, Syre allows engineers to explore vast design spaces, identifying optimal configurations that maximise performance and energy efficiency.

- **Versatile outputs:** One of Syre's main goals is to simplify the design evaluation phase by providing users with a range of useful outputs, including flow maps, torque curves, and ferrous loss assessments, that can be easily interpreted to guide further design iterations.

SyreDrive

Recently, the Syre ecosystem was enriched with the introduction of SyreDrive [12], an add-on that extends Syre's capabilities to the control simulation domain. SyreDrive allows Simulink models to be automatically generated from eMotor design results in Syre, facilitating control calibration and accurate simulation of electrical waveforms. This integrated approach makes it possible to comprehensively evaluate not only the static performance of the motor but also its dynamic response to various control algorithms.

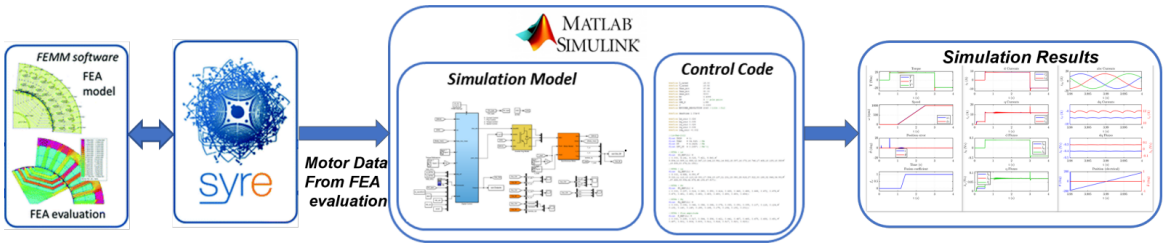

Figure 1.13: SyreDrive flowchart

The user interface of SyreDrive is represented in Fig. 1.14. From the tab the user can select:

1. Model setup: averaged or instantaneous;

2. Control type: current, torque or speed control;

3. Motor model type: fundamental (based on dq model) or with harmonics (based on $dq\theta$ model);

4. Control strategy: FOC (Field Oriented Control) or DFVC (Direct Flux Vector Control);

5. The representation of Iron losses in the motor model;

6. Converter data: PWM carrier frequency, threshold voltage ,internal resistance of the power modules and the dead time;

7. Sensorless control type if needed.

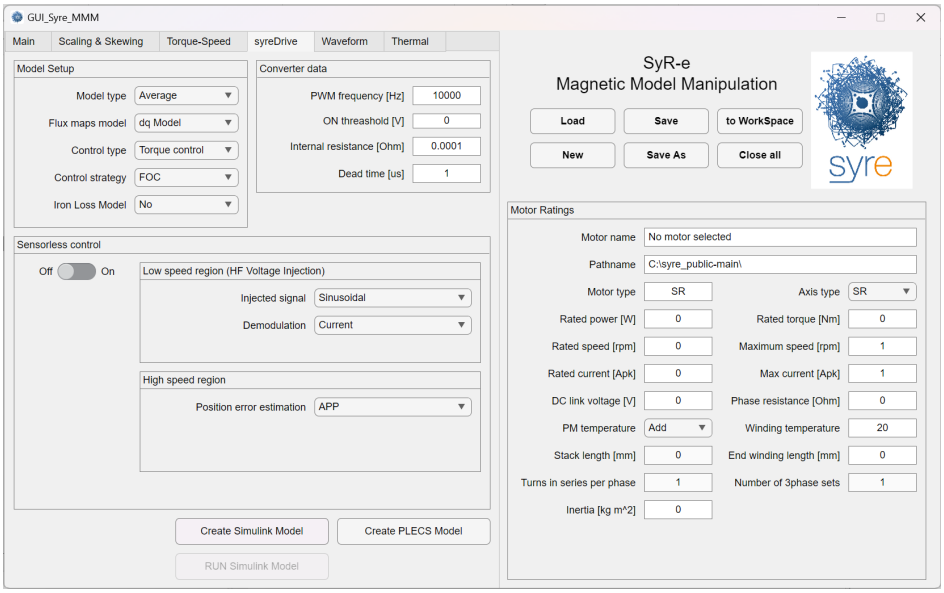

Figure 1.14: SyreDrive Interface

1.4.1 Background Available Models

SyreDrive can generates Simulink or PLECS models. These models will be considered the benchmark and the starting point for the development of a general and automatically generated vectorized model of multi-three phase motor drives, which is one of the key contributions of the thesis, and which will be adopted for simulating the drive under faulty conditions. The benchmark model is made of three main blocks:

1. The digital control;
2. The inverter block;
3. Motor model.

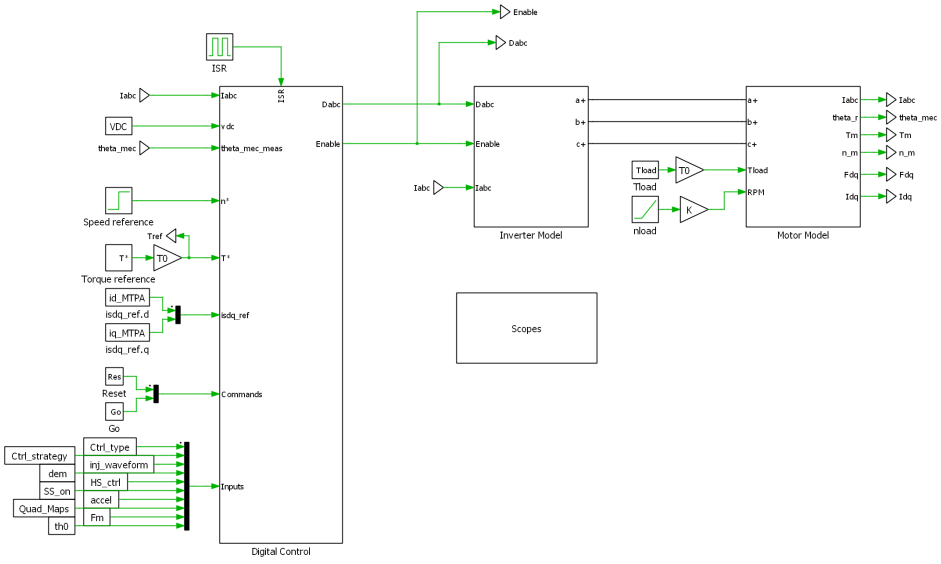

Figure 1.15: PLECS three phase electrical drive in SyreDrive

Digital Control

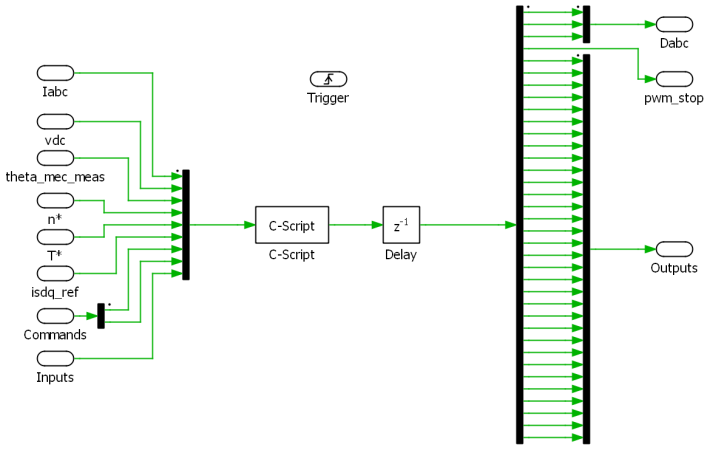

Figure 1.16: PLECS digital control block

The digital control block is designed to mimic the operational dynamics of a microcontroller used in experimental setups. It features a trigger mechanism that initiates the execution of the Motor Control code at a predetermined sampling time, mirroring the stringent timing requirements observed in actual real-time systems. This design choice ensures that the model accurately reflects the temporal precision necessary for controlling multi-phase motors.

Inputs to this 'virtual' microcontroller are efficiently aggregated, and outputs are diligently de-aggregated, simulating the process of data collection and actuation seen in physical control units. This methodical handling of signals reinforces the model's fidelity to actual control scenarios, where data flow management is crucial for system stability and performance.

Additionally, to enhance the realism of the simulation, a deliberate time delay is introduced in the signal outputs. This delay accounts for the inevitable latency found in hardware implementations, further bridging the gap between simulation and practical application. Through these meticulously designed features, the digital control block serves as a realistic proxy for the intricate control strategies employed in six-phase motor systems.

the structure of the motor control script can be broken down into several key sections:

- **Measurements Import**: This section is dedicated to importing all necessary quantities for computing control signals to manage the motor effectively.

- **State Machine**: The state machine serves as a critical code component, providing insights into the control phase and dictating subsequent actions. Capable of executing various operations based on user inputs (start and stop commands) or through its iterations (transitioning from WAKE UP to READY phase after charging the inverter's bootstrap capacitors), the state machine is versatile. It is further divided into:

  - **ERROR STATE**: The default state assumed when no input is activated, and the state for stopping the electrical drive. It's utilized for initializing variables and defining constants such as the proportional and integral gains of controllers.

– **WAKE UP**: Activated by an external start pulse (a signal in offline simulations or button press in real-time applications), this state maintains inverter duty cycles at 0.95 to charge gate circuits, essential for the electrical drive's operation. It can also be used for commissioning tasks like differential encoder setup, requiring sensorless control techniques to ensure encoder offset recording.

– **READY**: Marks the completion of commissioning functions, indicating the drive is prepared to enter the START state.

– **START**:Activated by an external start pulse, in this phase the electrical drive operates in the desired control mode, executing code sections that yield the intended outputs.

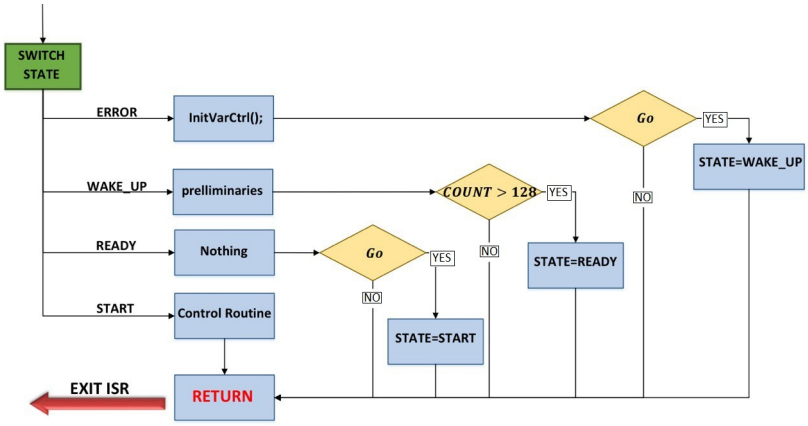

Figure 1.17: State machine flowchart

- **Duty Cycles Saturation**: This code segment tempers any control signals that could drive the operation into unsafe or overly demanding conditions. In some cases, duty cycles adhere to predefined maximum and minimum values.

- **Export of Commands and Observed Quantities**

Motor Model

Under the motor model mask it can be found the parallelism between mechanical and electrical world for the electrical machine.

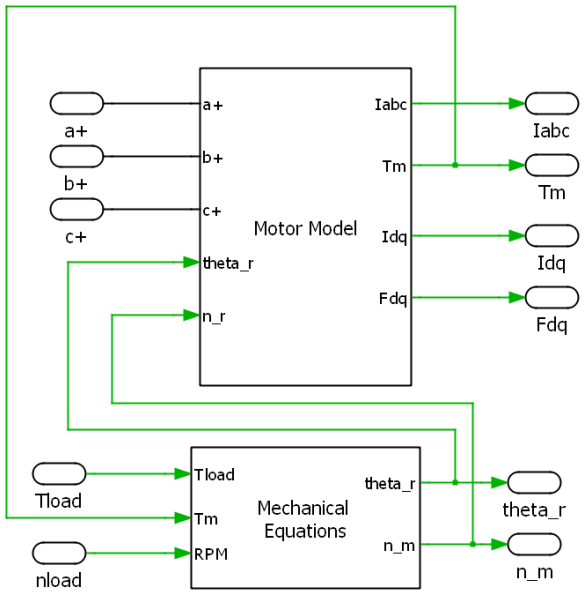

Figure 1.18: PLECS three phase motor model

For what concerns the electrical circuital model, two types are proposed based on [1]: Controlled Current Generators and Voltage Behind Reactance models.

CCG motor model

The motor is modeled as three current generators, operating in parallel with very large resistors, and all together in series with the phase resistances R_s. The current generators are piloted by the three-phase currents of the machine, and the voltage drop across the resistors represents the back electromotive force of the machine.

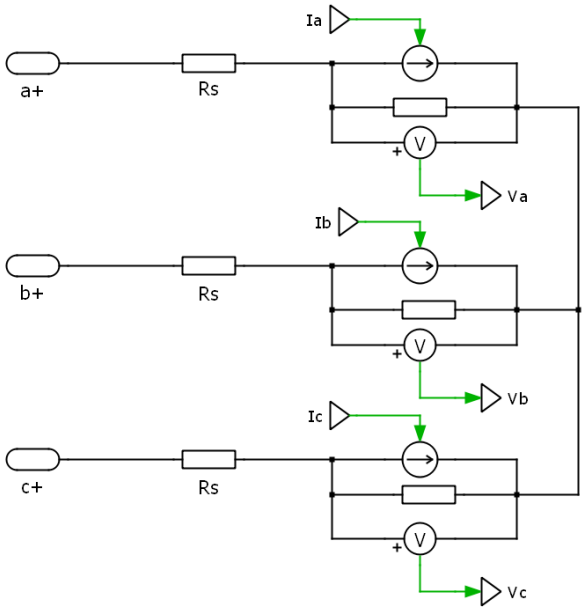

Figure 1.19: CCG model in PLECS

Hereafter is presented the block scheme for the calculation of the stator currents of the motor.

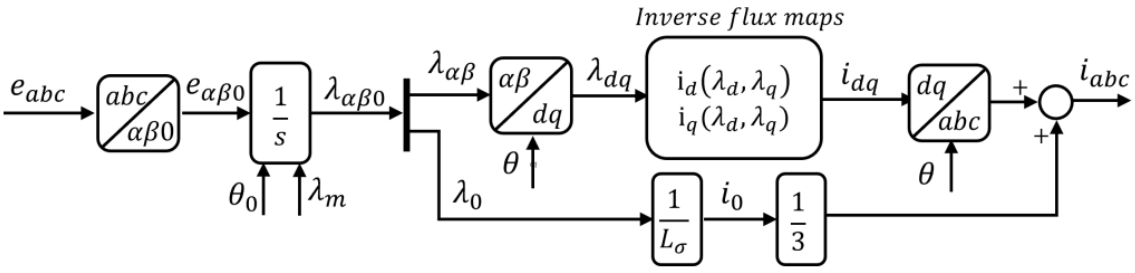

Figure 1.20: Block scheme for stator currents calculation in CCG model [1]

The transformations adopted to change from a frame model to another are the followings:

- **Direct Clarke transformation** $e_{a,b,c}$ **-** $e_{\alpha,\beta,0}$

$$\begin{bmatrix} e_\alpha \\ e_\beta \\ e_0 \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ 0 & \frac{\sqrt{3}}{3} & -\frac{\sqrt{3}}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix} \cdot \begin{bmatrix} e_a \\ e_b \\ e_c \end{bmatrix} \tag{1.27}$$

- **Direct Park transformation** $\lambda_{\alpha,\beta}$ **-** $\lambda_{d,q}$

$$\begin{bmatrix} \lambda_d \\ \lambda_q \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} \lambda_\alpha \\ \lambda_\beta \end{bmatrix} \tag{1.28}$$

- **Inverse Clarke-Park transformation** $e_{a,b,c}$ **-** $e_{\alpha,\beta,0}$

$$\begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} = \frac{2}{3} \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \cos\left(\theta - \frac{2\pi}{3}\right) & -\sin\left(\theta - \frac{2\pi}{3}\right) \\ \cos\left(\theta + \frac{2\pi}{3}\right) & -\sin\left(\theta + \frac{2\pi}{3}\right) \end{bmatrix} \cdot \begin{bmatrix} i_d \\ i_q \end{bmatrix} \tag{1.29}$$

- **Inverse flux maps** $i_{dq} = f(\lambda_{dq})$: these maps are retrieved from FEA simulations of the e-motor object of study.

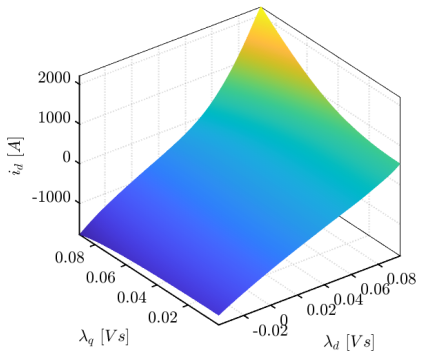

Figure 1.21: $I_d = f(\lambda_{dq})$ example

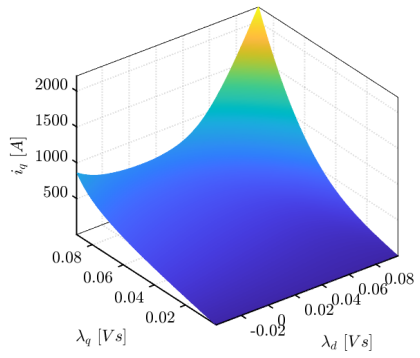

Figure 1.22: $I_q = f(\lambda_{dq})$ example

VBR motor model

In the Voltage Behind Reactance model, the machine is represented as a three-phase RLE load.
Here, the resistive parameters correspond to the winding resistances, while the coupled inductors represent the differential inductances of the machine.
Additionally, the controlled voltage sources serve as representations of the back-EMFs.

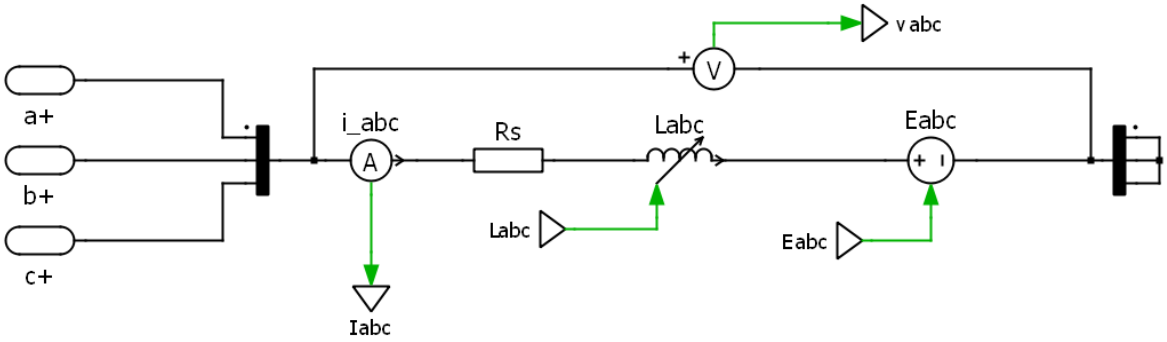

Figure 1.23: VBR model in PLECS

Despite the machine variables being initially expressed in the *abc* reference frame, the computation of the back electromotive forces occurs in the *dq* reference frame.
This is achieved through the utilization of inductances and flux maps as follows:

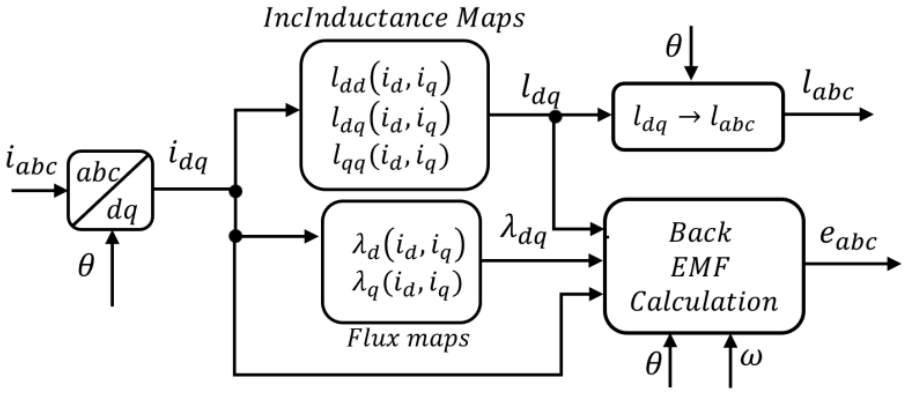

Figure 1.24: Block scheme for stator back EMFs calculation in VBR model [1]

Firstly, the measured currents are transformed from the *abc* to the *dq* reference frame. Subsequently, these currents are converted into fluxes and incremental inductances through the interpolation of two-dimensional lookup tables. Then, utilizing the information regarding currents, fluxes, and incremental inductances in the *dq* reference frame, the *abc* back electromotive force quantities are retrieved, along with the incremental inductances.

Hereafter, an in-depth focus on the calculations involved in the VBR model is made.

- **Direct Clarke-Park transformation** $i_{a,b,c}$ **-** $i_{d,q}$

$$\begin{bmatrix} id \\ iq \end{bmatrix} = \frac{2}{3} \begin{bmatrix} \cos(\theta) & \cos\left(\theta - \frac{2\pi}{3}\right) & \cos\left(\theta + \frac{2\pi}{3}\right) \\ -\sin(\theta) & -\sin\left(\theta - \frac{2\pi}{3}\right) & -\sin\left(\theta + \frac{2\pi}{3}\right) \end{bmatrix} \cdot \begin{bmatrix} ia \\ ib \\ ic \end{bmatrix} \tag{1.30}$$

- **Back-EMFs calculation** $e_{a,b,c}$=$f(i_{d,q}, \lambda_{d,q}, l_{d,q}, \theta, \omega)$
  From the stator voltage equation:

$$\overline{\mathbf{v}}_{abc} = [\mathbf{R_s}] \cdot \bar{\mathbf{i}}_{abc} + [\mathbf{l}_{abc}] \cdot \frac{d\bar{\mathbf{i}}_{abc}}{dt} + \overline{\mathbf{e}}_{abc} \tag{1.31}$$

  Using the direct and inverse Clarke transforms

$$[\mathbf{T}] = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ 0 & \frac{\sqrt{3}}{3} & -\frac{\sqrt{3}}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix} \quad , \quad [\mathbf{T}]^{-\mathbf{1}} = \begin{bmatrix} 1 & 0 & 1 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & 1 \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} & 1 \end{bmatrix} \tag{1.32}$$

  The voltage equation in the α, β frame is obtained:

$$\overline{\mathbf{v}}_{\alpha\beta} = \mathbf{R_s} \cdot \bar{\mathbf{i}}_{\alpha\beta} + [\bar{\mathbf{l}}_{\alpha\beta}] \cdot \frac{d\bar{\mathbf{i}}_{\alpha\beta}}{dt} + \cdot\overline{\mathbf{e}}_{\alpha\beta} \tag{1.33}$$

  Then, with the Park transformations

$$\mathbf{A}(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \quad , \quad \mathbf{A^{-1}}(\theta) = \mathbf{A}(-\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{1.34}$$

  The voltage equation in the d, q frame is:

$$\overline{\mathbf{v}}_{\mathbf{dq}} = \mathbf{R_s} \cdot \bar{\mathbf{i}}_{\mathbf{dq}} + [\mathbf{l_{dq}}] \cdot \frac{d\bar{\mathbf{i}}_{\mathbf{dq}}}{dt} + [\mathbf{J}] \cdot \omega \cdot \overline{\lambda}_{\mathbf{dq}} \tag{1.35}$$

  Where $[\mathbf{J}]$ is the complex derivate operator, expressed in matricial form

$$[\mathbf{J}] = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{1.36}$$

  From here the back-EMFs information is retrieved with few passages

$$\overline{\mathbf{e}}_{\mathbf{dq}} = [\mathbf{l_{dq}}] \cdot (-\omega) \cdot [\mathbf{J}] \cdot \bar{\mathbf{i}}_{\mathbf{dq}} + [\mathbf{J}] \cdot \omega \cdot \overline{\lambda_{dq}} \tag{1.37}$$

$$\overline{\mathbf{e}}_{\alpha,\beta} = \mathbf{A}(-\theta) \cdot \overline{\mathbf{e}}_{dq} = \mathbf{A}(-\theta) \cdot \left([\mathbf{l}_{dq}] \cdot (-\omega) \cdot [\mathbf{J}] \cdot \bar{\mathbf{i}}_{dq} + [\mathbf{J}] \cdot \omega \cdot \overline{\lambda}_{dq} \right) \tag{1.38}$$

$$\overline{\mathbf{e}}_{a,b,c} = \mathbf{T^{-1}} \overline{\mathbf{e}}_{\alpha,\beta} = \mathbf{T^{-1}} \cdot \mathbf{A}(-\theta) \cdot \overline{\mathbf{e}}_{dq} \tag{1.39}$$

- **Incremental inductances** $l_{a,b,c}$

  From the inductance LUTs is it possible to obtain incremental inductances in the dq frame

$$\begin{aligned} l_{dd} &= \frac{\partial \lambda_d(i_d, i_q)}{\partial i_d} \qquad & l_{dq} &= \frac{\partial \lambda_d(i_d, i_q)}{\partial i_q} \\ l_{qd} &= \frac{\partial \lambda_q(i_d, i_q)}{\partial i_d} \qquad & l_{qq} &= \frac{\partial \lambda_q(i_d, i_q)}{\partial i_q} \end{aligned} \tag{1.40}$$

  The steps to get the $l_{a,b,c}$ tensor are:

$$[\mathbf{l}_{\alpha\beta}] = \mathbf{A}(-\theta) \cdot [\mathbf{l}_{dq}] \cdot \mathbf{A}(\theta) \tag{1.41}$$

$$[\mathbf{l}_{abc}] = [\mathbf{T}]^{-1} \cdot [\mathbf{l}_{\alpha\beta}] \cdot [\mathbf{T}] \tag{1.42}$$

- **Flux maps** $i_{dq} = f(\lambda_{dq})$: as for the inverse ones, these maps are retrieved from FEA simulations.

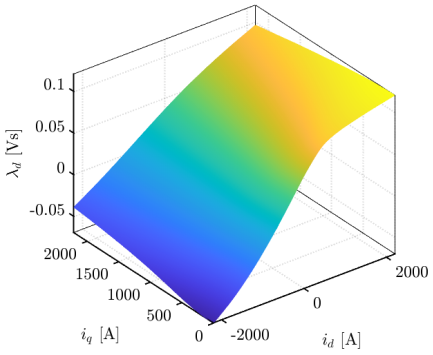

Figure 1.25: $\lambda_d = f(I_{dq})$ example

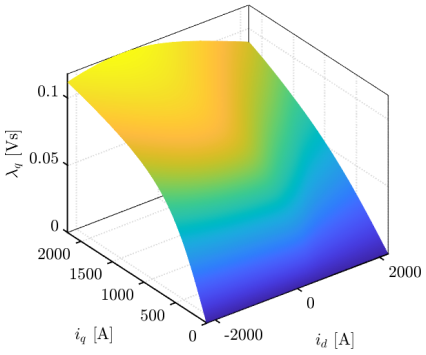

Figure 1.26: $\lambda_q = f(I_{dq})$ example

Mechanical Model

The mechanical component of the electric motor modeling is defined according to the control strategy employed.

Specifically, when the electric motor is governed via current control or torque control, it is assumed, as depicted in the upper part of Figure 1.27, that the speed is set by an external prime mover responsible for speed regulation.

Conversely, in scenarios where the electric motor is to be commanded under speed control (illustrated in the lower part of Figure 1.27), the mechanical aspect of the motor model incorporates both load and friction torques, which are proportional to ω and ω^2, respectively.

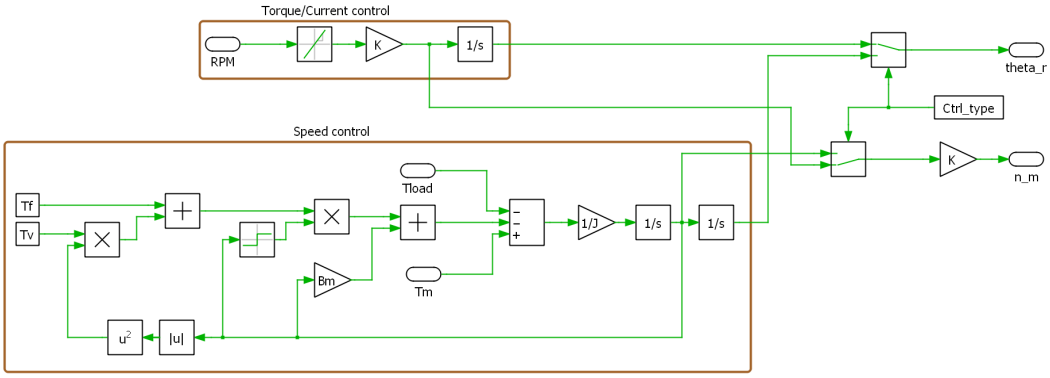

Figure 1.27: Representation of the Mechanical Model

This approach is of general validity and will be used throughout this discussion; only in this section its characteristics have been explored, for the rest of the discussion it will be taken for granted.

1.5 Hardware-in-the-loop Simulations

Hardware-in-the-loop (HIL) simulations stand at the forefront of testing and validating real-time electronic control systems.

By utilizing mathematical models to mimic the behavior of complex physical systems, HIL simulations allow hardware to undergo extensive testing under a wide range of scenarios without the associated risks and costs of using actual equipment.

This approach is invaluable during the development and prototyping phases of controllers for electric motors, drive systems, and other power electronics, enabling engineers to detect and correct issues well before physical implementation.

1.5.1 Plexim RT-BOX

The RT Box is a real-time simulator designed with power electronics applications in mind. It distinguishes itself in HIL simulations through its exceptional capability to accurately replicate electrical and electronic systems, offering an essential tool for engineers and researchers alike.

The RT Box's versatility stems from its sophisticated hardware, featuring numerous analog and digital I/O channels, along with FPGA embedded CPU cores. This makes it highly effective for both real-time HIL testing and rapid control prototyping across a variety of sectors. In our study, particular use was made of the RT Box CE version, a compact and cost-effective model of the RT Box, designed for smaller-scale HIL applications. This version is especially suited for educational and research purposes, where its portability allows for easy transportation by students and researchers for remote work or study, without sacrificing the high performance of the larger RT Box model.

Figure 1.28: RT-Box CE Photo

Key technical specifications of the RT Box CE are summarized below:

Feature	Specifications
Processor	Xilinx Zynq Z-7030
CPU Cores	2 x ARM Cortex-A9, 1 GHz
Analog Inputs	8 channels, 16 bit, simultaneous sampling
Analog Outputs	16 channels, 16 bit, simultaneous update
Digital Inputs	32 channels, logic levels 3.3 V (5 V tolerant)
Digital Outputs	32 channels, logic levels 3.3 V / 5 V
I/O Protection	Permanent short-circuit, Overvoltage -24 ... 24 V
Connectivity	Gigabit Ethernet, 2 CAN bus, USB A 2.0
Power Supply	100 ... 240 Vac, 50 ... 60 Hz, 30 VA
Dimensions	225 x 165 x 55 mm

Table 1.1: RT Box CE Technical Specifications

1.5.2 ST Microelectronics Nucleo-64 G4 Micro Board

The STM32 Nucleo-64 G4 board provides a cost-effective and flexible platform for users to test new ideas and build prototypes, taking advantage of the diverse performance and power consumption options offered by the STM32 microcontroller. Ideal for educational and rapid prototyping environments, it allows for easy experimentation with various designs and concepts.

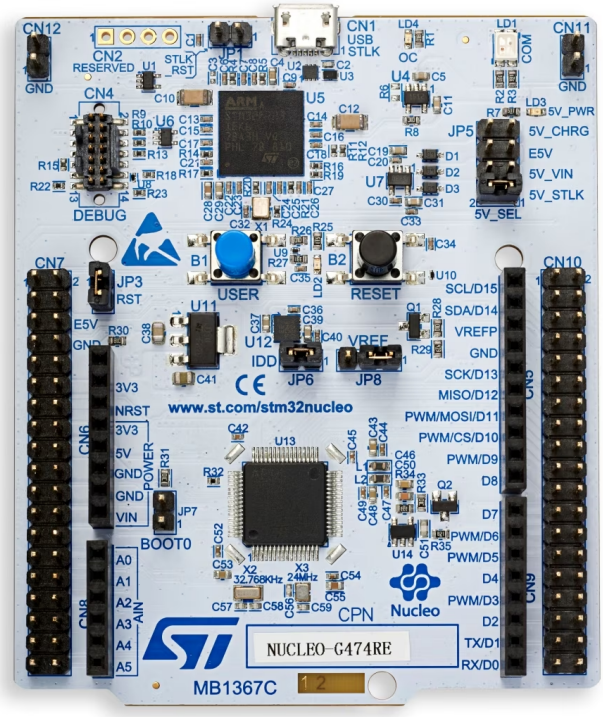

Figure 1.29: STM Nucleo Board Photo

The primary specifications of the Nucleo G4 board are detailed in the following table:

Feature	**Specifications**
Microcontroller	STM32G4 series in LQFP64 package
Core	ARM Cortex-M4 with FPU and DSP instructions
Maximum Frequency	Up to 170 MHz
Memory	Up to 512 KB Flash, up to 128 KB SRAM
Analog Inputs	12-bit ADCs with 16 channels
Analog Outputs	2 DAC channels
Debugging	On-board ST-LINK/V2-1 debugger/programmer
Connectivity	ARDUINO® Uno V3 connectivity, ST morpho extension pin headers
Power Options	USB VBUS or external source
Additional Features	USB OTG FS, TIM, I2C, SPI, USART, LPUART, CAN, SAI
Operating Voltage	1.7 V to 3.6 V

Table 1.2: STM32 Nucleo-64 G4 Micro Board Technical Specifications

This hardware setup, combining the RT Box CE with the Nucleo G4 microcontroller board and leveraging STM32CubeIDE for code development, offers a powerful and adaptable platform for conducting complex HIL projects, including the simulation of electrical motor circuit models.

1.6 Modelling Strategy

For the simulation of three-phase and multi-three-phase motors, we adopted a hybrid machine modelling strategy, distinct for the electric motor model in the PLECS simulation environment (RT-BOX) and for the machine control in C code (Motorctrl.c for offline simulations, STM32 IDE project in real time environment).

1.6.1 Motor Model

Regarding the circuital models of the e-motors within PLECS, we opted for the Vector Space Decomposition approach. This methodology, as demonstrated in [3] [10] [9], offers an excellent representation of the machine due to its ability to condense the electromechanical conversion into a single subspace dq, simplifying the treatment of the multiphase machine as an ordinary rotating motor.
For the automation and integration in SyreDrive of multiphase machine models, an algorithm will be developed for the generation of the transformation matrix from phase quantities to VSD subspaces, as illustrated in [14]. From this point it is possible to treat the motors as already seen in 1.4.1, with VBR and CCG models as possible solutions.

1.6.2 Control Strategy

In contrast to the motor model, the control of three-phase and multi-three-phase machines will be done through the Multi Stator approach. This choice is aimed at preserving the modularity of these drive systems, ensuring an efficient response to the various types of faults simulated. To address the problem of cross-coupling between the three-phase systems that make up the stator windings, we will resort to the implementation of the decoupling algorithms presented in [11] [9]. In this way, the machine will be protected from this problem, simplifying the analysis as shown in 3.20, resulting in currents referring to the common and differential modes.

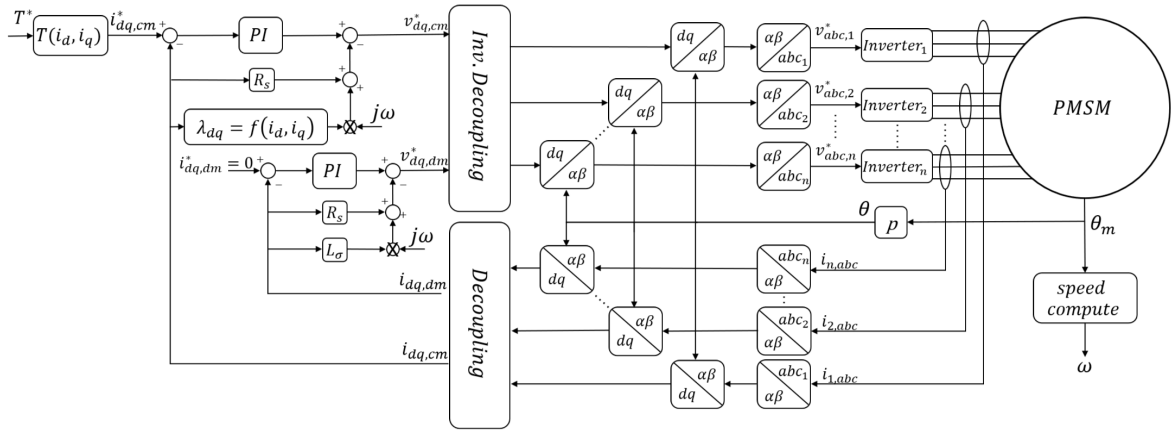

Figure 1.30: Adopted control scheme

From the figure 1.30 it can be seen that, after the appropriate transformations, the machine is pair-controlled $T_{ref} = f(i_{d,cm}, i_{q,cm})$ in the common-mode axes dq. This decision is motivated by the focus of the thesis on the development of simulations under fault conditions, offering considerable simplicity and debugging capabilities. To support proportional-integral controllers, a voltage feedforward will be employed for more effective compensation and quicker response to torque references.
Clarke transformations from axes $a_k b_k c_k$ to axes $\alpha_k \beta_k$ are obtained as described in [11], exploiting

the asymmetry of the e-motors under study:

$$[\mathbf{T_{Clarke,k}}] = \frac{2}{3} \cdot \begin{bmatrix} cos(\theta_k) & cos(\theta_k - \frac{2\pi}{3}) & cos(\theta_k + \frac{2\pi}{3}) \\ sen(\theta_k) & sen(\theta_k - \frac{2\pi}{3}) & sen(\theta_k + \frac{2\pi}{3}) \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \tag{1.43}$$

Furthermore, there is no difference in the definition of the Park transformation from $\alpha_k\beta_k$ to d_k, q_k.

1.7 Motivations

The exploration of the current landscape in electrical motor drive systems, particularly in the domain of three-phase and multi-phase machines, reveals a significant gap in comprehensive simulation tools capable of addressing the complex nature of fault conditions.
Despite considerable advancements in the field, the development and implementation of robust, fault-resilient systems remain a critical challenge.
This is especially true in applications where reliability and performance under adverse conditions are paramount.

Literature review underscores the multifaceted nature of fault scenarios in electrical drives, ranging from open-circuit faults to more intricate issues like inter-turn short circuits and phase unbalances. While existing methodologies provide a foundation for understanding and mitigating these issues, there remains a conspicuous absence of an all-encompassing tool that integrates motor modeling, control strategy simulation, and fault analysis within a singular, user-friendly platform.

This gap not only hinders the optimization of motor drives for fault tolerance but also complicates the process for engineers and researchers to simulate and analyze the effects of various fault conditions on motor performance.
The necessity for manual integration between different software tools for modeling, control design, and fault simulation adds layers of complexity and potential for error, making the development cycle longer and less efficient.

Moreover, the evolving landscape of electric motor applications, particularly in critical sectors like automotive, aerospace, and renewable energy, demands more sophisticated and adaptable solutions.
The integration of multi-phase systems, with their inherent advantages in terms of fault tolerance and efficiency, further accentuates the need for a simulation tool that can seamlessly model and analyze these complex systems under a wide range of operating conditions and fault scenarios.

Therefore, this thesis is motivated by the clear necessity to bridge this gap by developing an integrated simulation environment that can offer comprehensive capabilities for the modeling, control strategy evaluation, and fault simulation of three-phase and multi-phase electrical drives.
Such a tool would not only facilitate a deeper understanding of fault dynamics and mitigation strategies but also significantly accelerate the development and optimization of fault-resilient motor drive systems.
This endeavor aims to contribute to the advancement of electric motor technology, ensuring higher reliability and efficiency in applications where failure is not an option.

Chapter 2

Motors Under Test

This chapter introduces the electric motors selected for detailed study and simulation throughout this thesis. Each motor, with its unique configuration and specifications, represents a different aspect of multiphase motor design and application. The motors range from standard three-phase configurations to more complex twelve-phase systems, providing a broad spectrum for analysis in terms of operational efficiency, control strategies, and fault resilience.

Notably, these motors are readily available for study due to their inclusion in the Syr-e library, a comprehensive repository of motor models designed for extensive analysis and optimization in multiphase systems. This accessibility facilitates the exploration of various motor configurations and enhances the scope of simulation studies.

The reader can easily observe that there are correlations between the quantities obtained from the Hexaphase motor and the Dodecaphase motor, as well as between the Three-phase motor and the Nine-phase motor. Indeed, at their core, these are the same motor and differ only in the stator winding, which has been rewound for a different number of three-phase sets. Therefore, these pairs of motors will present the same nominal torque and speed, while the nominal phase currents will be scaled accordingly to the number of three-phase sets comprising the e-motor.

The diverse selection of motors—3 and 9-phase motors characterized by their permanent magnet synchronous reluctance with a high degree of anisotropy for high-speed applications delivering modest torque, contrasted with the 6 and 12-phase motors, which are surface-mounted permanent magnet types with a high degree of isotropy for high torque output at low speed ranges—underscores the thesis's objective to develop a universally applicable automatic generation model for all types of three-phase and multi-three-phase machines. This heterogeneity in motor cases serves as a deliberate strategy to validate the developed models, illustrating the comprehensive applicability and robustness of the proposed simulation approach across a wide spectrum of electric motor designs.

2.1 THOR_1x3ph

The THOR motor is a PM-SyR motor. Since it is a PM-SyR with a high rotor anisotropy, it is treated with the axis convention proper to SyR motors, i.e. with the axis d in the direction of minimum rotor reluctance and the q axis in the direction opposite to the magnets flux. It is a high-speed motor that delivers modest torque. From the motor geometry in the figure 2.2 it is clear that this is a 3 slot/pole/phase technology. The parameters of the THOR motor are tabulated below, indicating its nominal and maximum operational capabilities.

Parameter	Value
Motor Name	THOR
Nominal Current	22 A
Maximum Current	50 A
DC Voltage	310 V
Maximum Speed	9000 rpm
Nominal Speed	2500 rpm
Nominal Torque	19 Nm
Nominal Power	5 kW
Number of Pole Pairs	2
Number of Phases	3
Stator Resistance	0.2625 Ω
Copper Temperature	130 °C
Permanent Magnet Temperature	20 °C
Number of Turns	72
Moment of Inertia	0.0045 kg·m^2
Axis Type	SR
Motor Type	PM
PM Temperature	[20, 180] °C
Target PM Temperature	80 °C

Figure 2.1: Main Parameters of the THOR Electric Motor

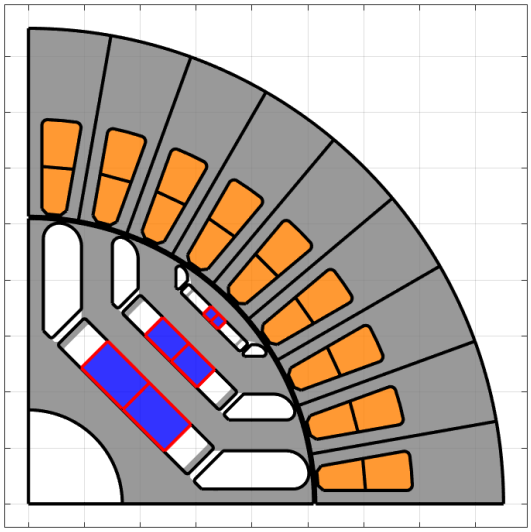

Figure 2.2: THOR motor geometry

Figures 2.3, 2.4 show the torque and power characteristic curves as a function of the motor's speed, respectively, even under flux-weakening operating conditions.

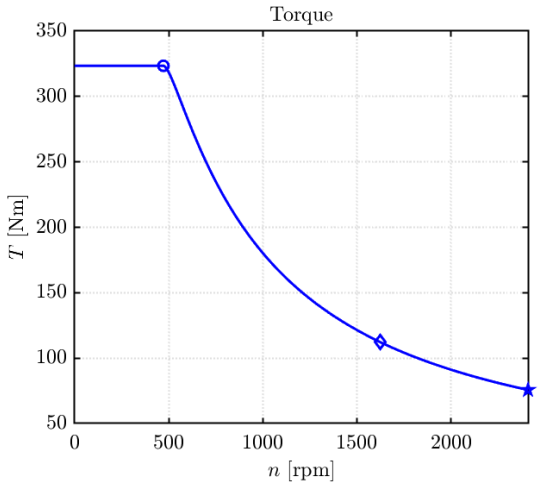

Figure 2.3: T-n Characteristic

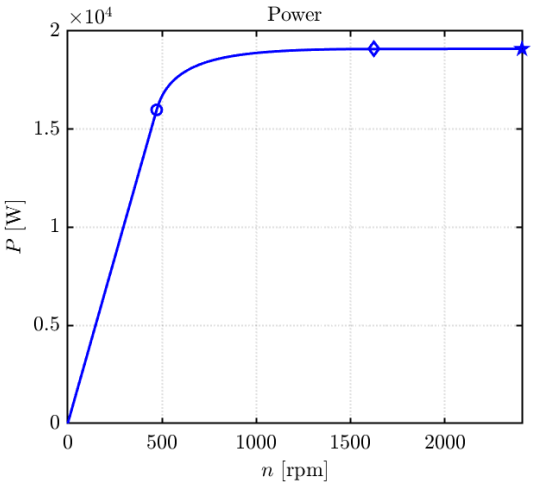

Figure 2.4: P-n Characteristic

Figures 2.5 show the flux maps *dq* of the THOR_1x3ph motor.

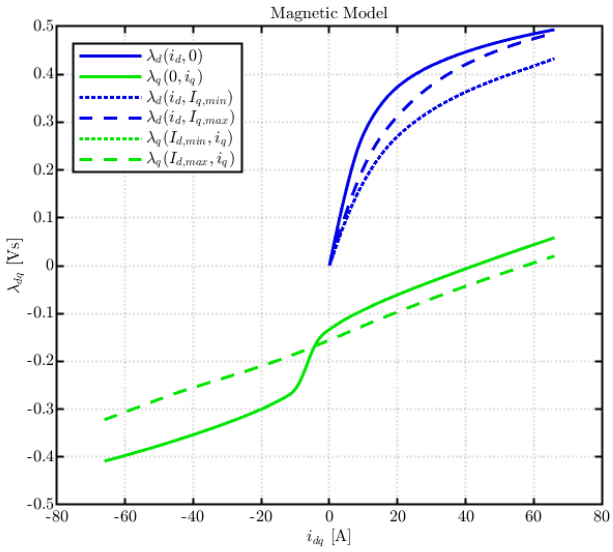

Figure 2.5: 2D Flux Maps

In figure 2.6 are represented respectively isotorque and isocurrents, MTPA and MTPV characteristics as a function of currents. Figures 2.7 show the curves that estimate the torque ripple in the *dq* currents plane. Since this motor adopt PM-Syr axes convention, these curves are symmetrical with respect to the i_q axis.

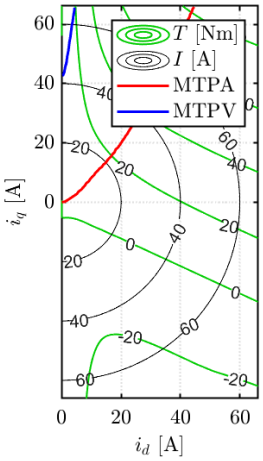

Figure 2.6: MTPA, MTPV, T curves

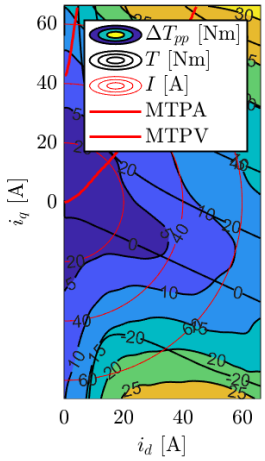

Figure 2.7: Torque Ripple Estimation

2.2 THOR_3x3ph

The evolution from the three-phase THOR motor to the THOR_3x3ph represents a significant leap in multiphase motor technology. This version has been adapted to house three sets of three-phase windings, effectively tripling the complexity and control capabilities compared to its predecessor. This modification led to a scaling of the slots/pole/phase from 3 to 1, maintaining the high-speed characteristics inherent to this PM-Syr motor design while adjusting its operational dynamics.

Parameter	Value
Motor Name	THOR_3x3ph
Nominal Current	7.5 A
Maximum Current	15 A
DC Voltage	310 V
Maximum Speed	12000 rpm
Nominal Speed	2400 rpm
Nominal Torque	20 Nm
Nominal Power	5 kW
Number of Pole Pairs	2
Number of Phases	9
Stator Resistance	0.79 Ω
Copper Temperature	130 °C
Permanent Magnet Temperature	20 °C
Number of Turns	72
Moment of Inertia	0.0042 kg·m^2
Axis Type	SR
Motor Type	PM
Target PM Temperature	80 °C

Table 2.1: THOR_3x3ph Data

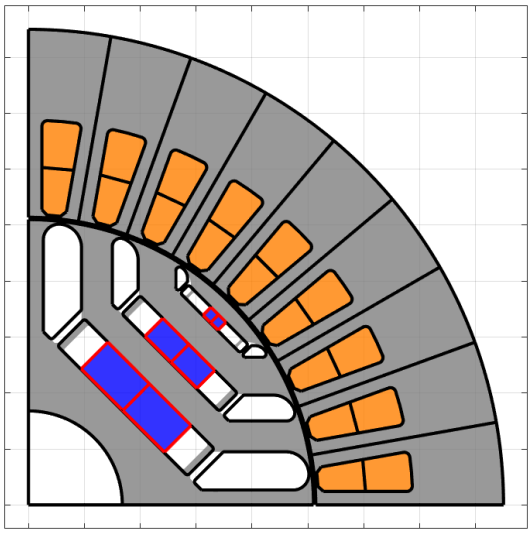

Figure 2.8: THOR motor geometry

The transition to a nine-phase system is not just a numerical increase in phase count but a strategic enhancement to improve fault tolerance, control granularity, and operational flexibility. The THOR_3x3ph's specifications underscore its high-speed operation and precise torque control capabilities, essential for advanced applications requiring meticulous performance tuning.

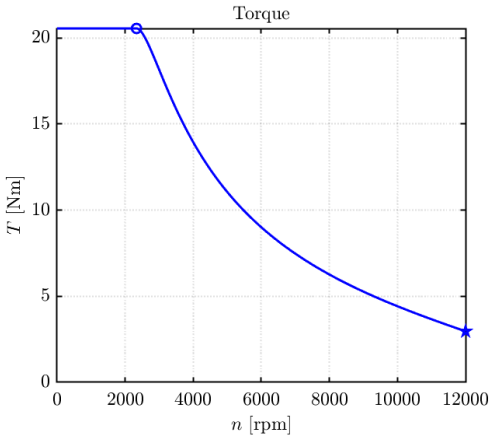

Figure 2.9: T-n Characteristic

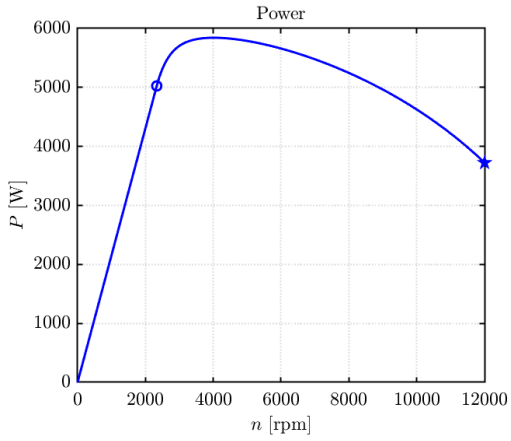

Figure 2.10: P-n Characteristic

The *dq* flux maps, characteristic torque, and power curves of the THOR_3x3ph motor, illustrated

in Figures 2.9, 2.10, and 2.11, respectively, offer a detailed view into the motor's performance under various conditions. These visual representations provide invaluable insights for optimizing the motor's operation to achieve desired outcomes.

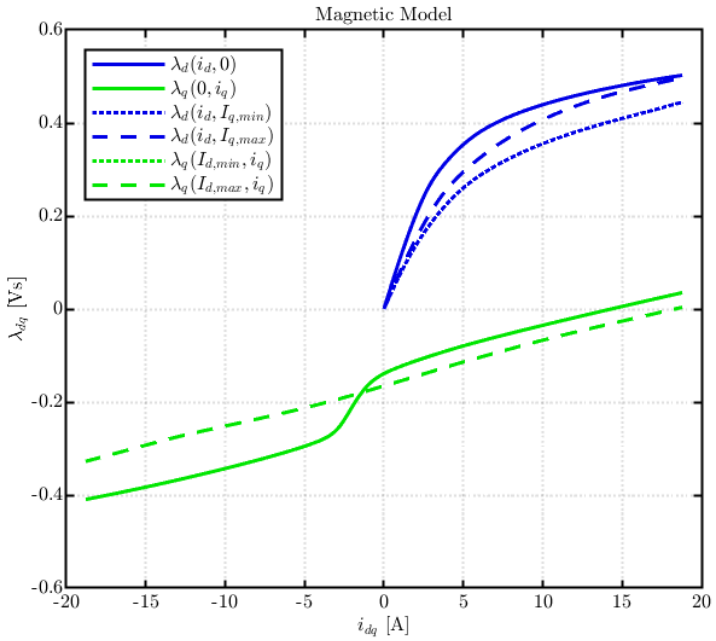

Figure 2.11: 2D Flux Maps

Moreover, the torque ripple estimation, as shown in Figure 2.13, and the MTPA, MTPV, T curves in Figure 2.12, provide a comprehensive understanding of the motor's efficiency and reliability. The symmetrical design regarding the i_q current axis, as a result of the PM-SyR axes convention, highlights the motor's balanced performance, minimizing torque ripple and enhancing operational smoothness.

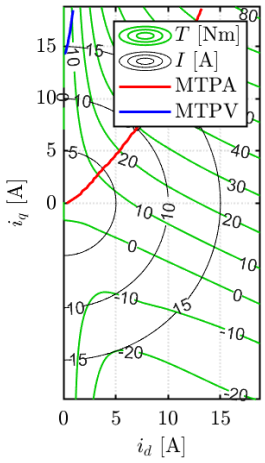

Figure 2.12: MTPA, MTPV, T curves

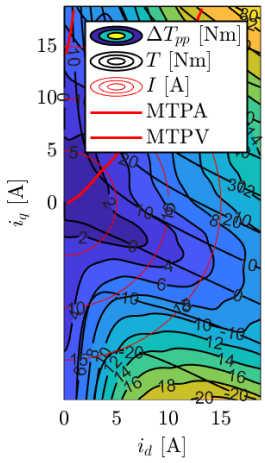

Figure 2.13: Torque Ripple Estimation

2.3 PM_2x3ph

The PM_2x3ph motor, represented in the table below, is distinguished by its six-phase design, enhancing its capability to deliver high torque and power levels efficiently. This motor is a testament to advanced engineering, optimizing performance for demanding applications that require robust power delivery and reliability.

Parameter	Value
Motor Name	PMSM_2x3ph
Nominal Current	23 A
Maximum Current	69 A
DC Voltage	565 V
Maximum Speed	2400 rpm
Nominal Speed	520 rpm
Nominal Torque	320 Nm
Nominal Power	17.5 kW
Number of Pole Pairs	22
Number of Phases	6
Stator Resistance	1.6 Ω
Copper Temperature	130 °C
Permanent Magnet Temperature	80 °C
Moment of Inertia	0.0576 kg·m^2
Axis Type	PM
Motor Type	PM
Target PM Temperature	80 °C

Table 2.2: Main Parameters of the Six-phase Electric Motor

The dynamic characteristics and operational capabilities of the PM_2x3ph motor are showcased through its *dq* flux maps and torque-speed, power-speed characteristic curves. These graphical representations, as seen in Figures 2.14, 2.15, and 2.16, provide valuable insights into the motor's efficiency across a range of operating conditions.

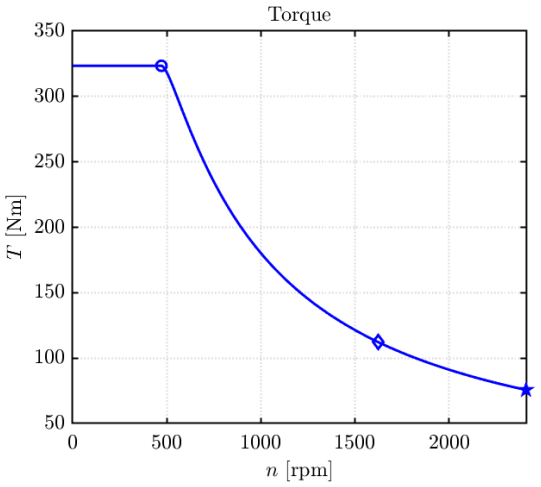

Figure 2.14: T-n Characteristic

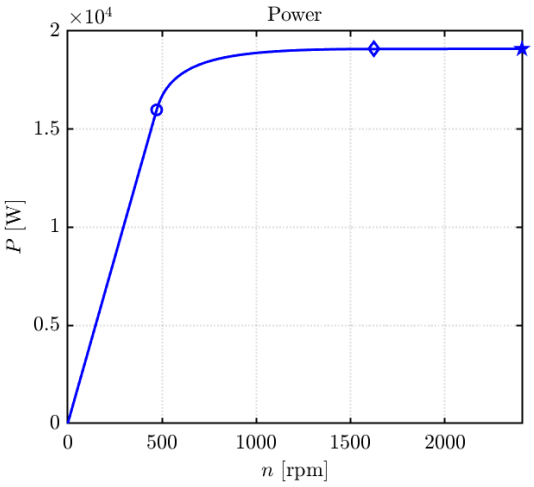

Figure 2.15: P-n Characteristic

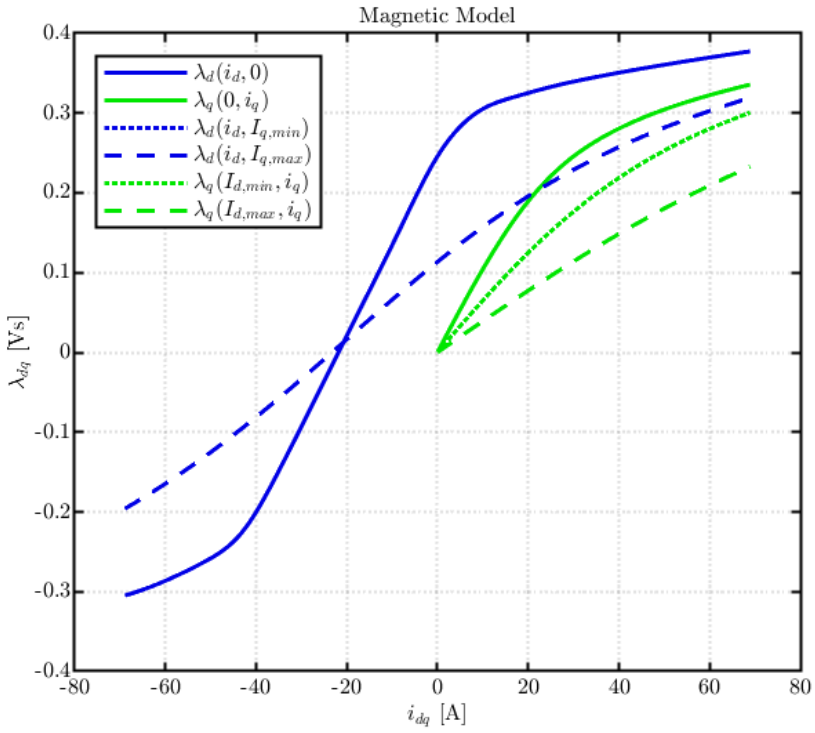

Figure 2.16: 2D Flux Maps

The analysis is further deepened by examining the MTPA, MTPV, and torque ripple characteristics, as illustrated in Figures 2.17 and 2.18. These figures emphasize the motor's operational efficiency, showcasing its capacity to maintain stable performance and minimize torque ripple, a crucial factor for applications demanding smooth and precise power delivery.

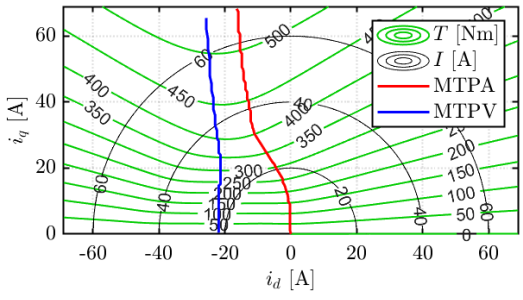

Figure 2.17: MTPA, MTPV, T curves

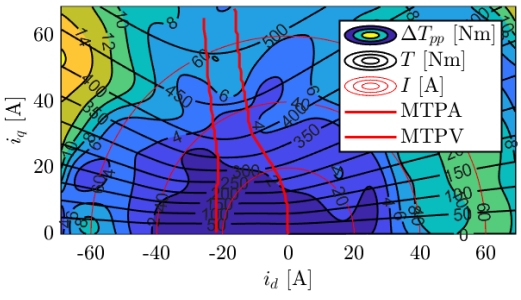

Figure 2.18: Torque Ripple Estimation

2.4 PM_4x3ph

The PM_4x3ph motor, detailed in the specifications below, is tailored for challenging applications that demand exceptional reliability and performance. Its twelve-phase architecture sets a new benchmark in electric motor design, offering unmatched flexibility and efficiency in power distribution and control.

Parameter	Value
Motor Name	PMSM_4x3ph
Nominal Current	11 A
Maximum Current	35 A
DC Voltage	565 V
Maximum Speed	2400 rpm
Nominal Speed	500 rpm
Nominal Torque	325 Nm
Nominal Power	17.5 kW
Number of Pole Pairs	22
Number of Phases	12
Stator Resistance	3.0 Ω
Copper Temperature	130 °C
Permanent Magnet Temperature	80 °C
Moment of Inertia	0.0574 kg·m^2
Axis Type	PM
Motor Type	PM
Target PM Temperature	80 °C

Table 2.3: Main Parameters of the Twelve-phase Electric Motor

The dynamic performance and adaptability of the PM_4x3ph motor are illustrated through its *dq* flux maps, torque-speed, and power-speed characteristic curves, as depicted in Figures 2.19, 2.20, and 2.21. These visual aids convey the motor's proficiency in maintaining high operational standards across various speeds and loads.

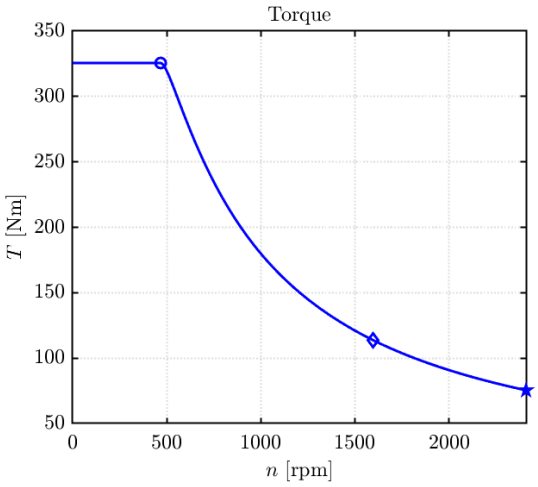

Figure 2.19: T-n Characteristic

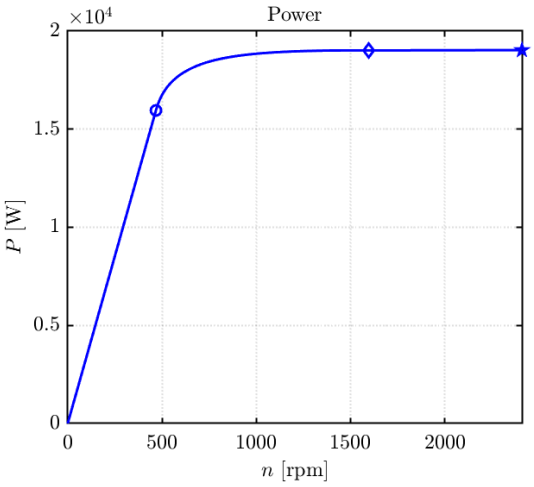

Figure 2.20: P-n Characteristic

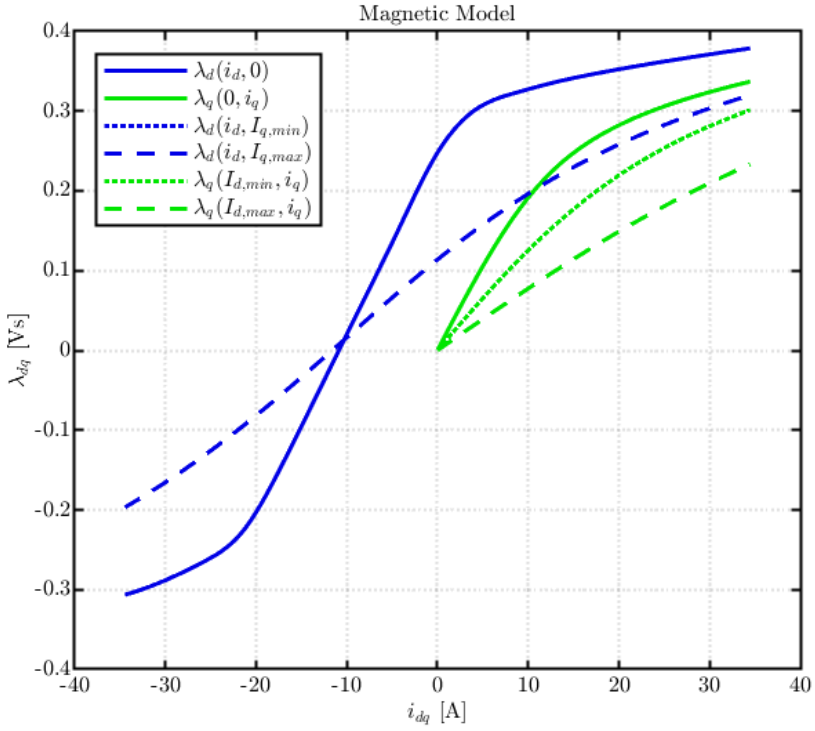

Figure 2.21: 2D Flux Maps

Further exploration into the motor's efficiency and control optimization is provided by the MTPA, MTPV, and torque ripple characteristics, illustrated in Figures 2.22 and 2.23. These analyses highlight the motor's exceptional capability to deliver smooth power output while minimizing torque ripple, ensuring stability and reliability in its application.

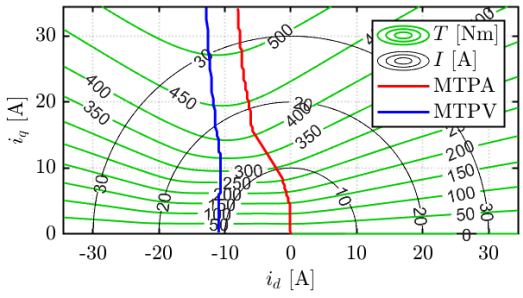

Figure 2.22: MTPA, MTPV, T curves

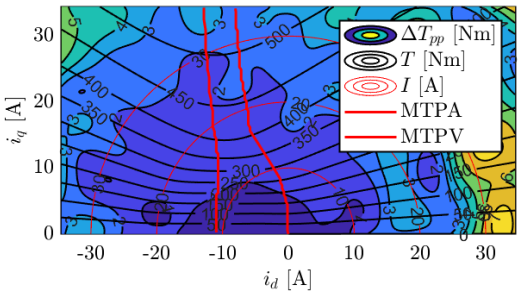

Figure 2.23: Torque Ripple Estimation

Chapter 3

PLECS Modeling of Three-Phase and Multi-Three-Phase Motors

This chapter is dedicated to an in-depth exploration of the modelling of three-phase and multi-three-phase motors using the advanced PLECS simulation environment. Our exploration begins with the development of an extended model of a six-phase motor, which will serve not only as a practical demonstration of the modelling methodology, but also as a starting point to discuss the inherent challenges and increasing complexity associated with visually representing a larger number of phases.

A significant turning point in our modelling journey is introduced with the development of the 'vectorised' model for multi-phase motors. This innovative representation not only promises a simplification of the graphical interface, but also proves essential in facilitating the automatic generation of models, showing a natural predisposition towards a more fluid and integrated approach. The transition to the implementation of the model within SyreDrive marks a crucial phase of our study, where we will examine in detail the procedures adopted to develop the control scripts and codes required for the auto-generation of the working model.

The evolution of modelling reaches a further level of concretisation through the application of models in Hardware-in-the-Loop (HIL) contexts, exploiting the capabilities of the rt-box real-time simulator. This approach not only demonstrates the effectiveness of our models in faithfully replicating engine behaviour under real operating conditions, but also opens up new perspectives for the validation and optimisation of control strategies.

We will conclude the chapter by presenting simulation results for four different motor configurations - three-phase, six - phase, nine-phase and twelve-phase - generated through our methodological approach. The analysis of the results will not only confirm the validity and effectiveness of the vectorised model and integration in SyreDrive, but also provide crucial insights for future research and development in the field of multi-three-phase motors, with a focus on HIL implementation via rt-box, outlining an innovative path for modelling, simulation and control of these advanced electric propulsion solutions.

3.1 Extended Six-Phase Machine Model

The fuoundation element of our exploration into multi-phase motor modeling within the PLECS environment is the extended six-phase motor model.

The extended six-phase model is distinctively characterized by its dual inverter setup, where each inverter operates independently.

Internally, the motor circuit model is ingeniously designed as a unified block that houses dual three-phase Controlled Current Generator (CCG) systems. These systems are arranged to ensure their signal routings—encompassing both output currents and voltage measurements—remain distinct and parallel, mirroring the dual inverter configuration.

In a similar vein, the digital control scheme of the model is adeptly structured to accommodate the dual nature of the motor. Inputs from current measurements and outputs to Duty Cycles are processed in parallel streams.

In Figure 3.14 is presented the shematic of this extended multiphase electrical drive.

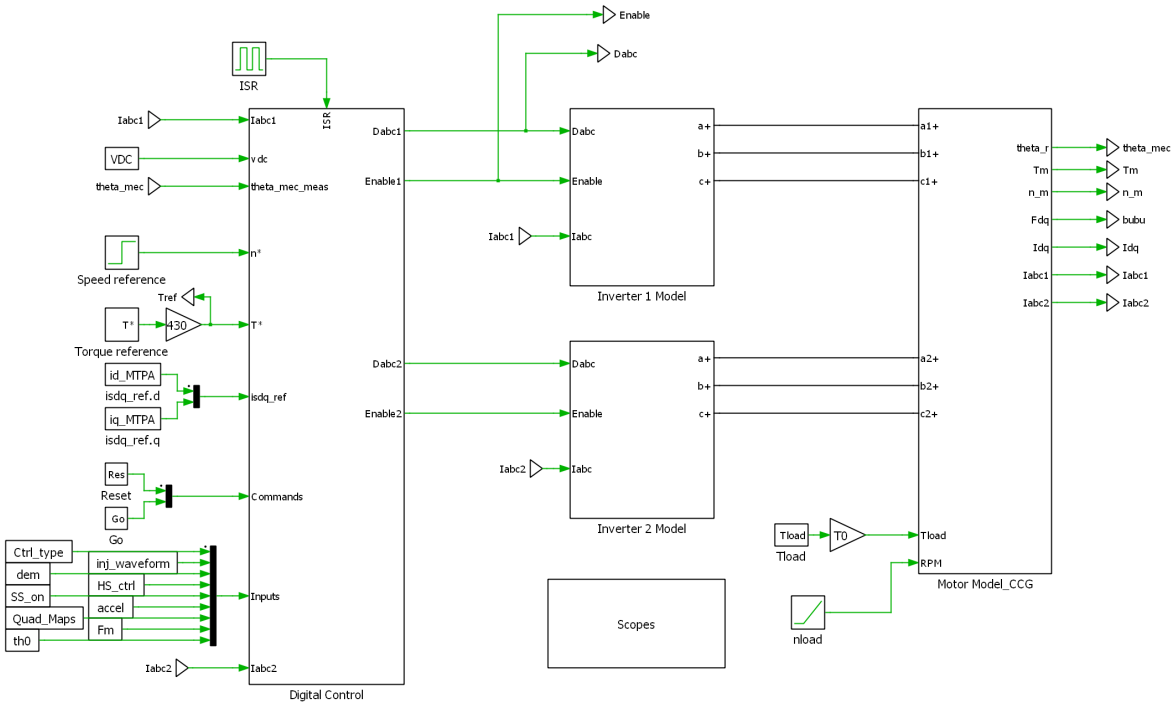

Figure 3.1: Six Phase Machine Schematic

3.1.1 Digital Control

At the core of our modeling efforts within the PLECS environment lies the digital control block, depicted in Figure 3.2. This component is fundamental to the simulation, embodying the sophisticated logic and precision typically found in real-world microcontroller applications.
The approach used is entirely equivalent to that observed in 1.4.1, so the reader who wishes to recall the characteristics of this method may refer to it.

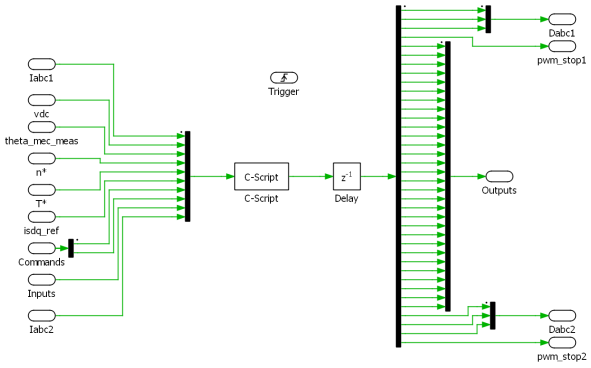

Figure 3.2: Six-phase Control Block

Control Script

Delving into the control script reveals the algorithmic backbone central to the digital control block. This script, presented in the 4.5.2 incorporates a comprehensive logical framework and sophisticated control strategies, guiding the motor's behavior across a spectrum of operational conditions.

The initial part of the code is dedicated to variable initialization and acquiring input signals. These signals represent motor currents (isabc1, isabc2), DC link voltage (vdc), mechanical rotor angle (theta_mec_meas), speed reference (n_ref_in), external torque reference (T_ext), current references (isdq_ext.d and isdq_ext.q), and several control flags and parameters such as reset, go, control type, waveform injection type, demodulation type, high-speed control activation, sensorless operation flag, acceleration, quadrature maps, and magnet flux (lambda_M).

Following the input signal acquisition, the code transitions into a state machine with states including ERROR, WAKE_UP, READY, and START. Each state is designed to handle specific phases of motor operation:

- ERROR: Initial state for variable initialization and system reset.
- WAKE_UP: Prepares the motor for operation by setting up PWM signals to a neutral position.
- READY: Awaits the start command while keeping the motor ready.
- START: The main operational state where speed computation, position error estimation, and control type selection take place. Depending on the control type, current, torque, or speed control is executed.

For torque and speed control, lookup tables (ReadLut) are used to determine appropriate current references. The control algorithms then compute the necessary current vector control commands to achieve the desired motor performance.

The Current_loop function is pivotal in generating reference voltages (vsdq_cm_ref and vsdq_dm_ref1) for common-mode and differential-mode operations, respectively. These references are adjusted for decoupling, ensuring precise control over motor currents.

Finally, duty cycles for PWM signals are calculated, ensuring they remain within acceptable bounds to prevent overdriving the motor. The computed duty cycles (duty_abc1 and duty_abc2) are then outputted to drive the motor's power electronics.

3.1.2 Motor Model

As expected, the circuital model of this extended machine is made of two sets of CCG three phase systems.

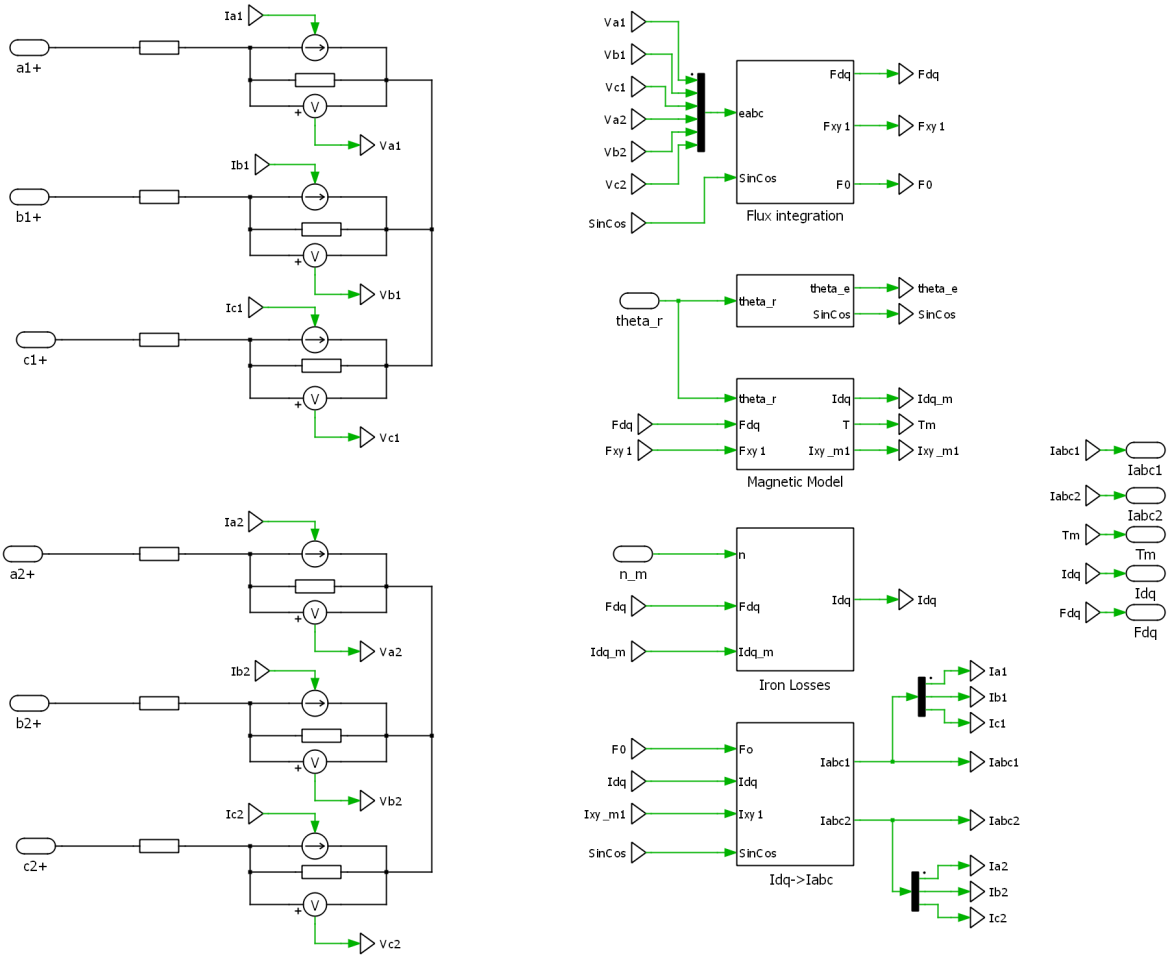

Figure 3.3: Six-phase machine motor model with 3CCG

In this section, the calculations performed in 1.4.1 are integrated within the simulation environment using various subsystems.

Flux integration

In the first step, the voltages measured across the terminals of the controlled current generators provides the values of fluxes.

The six back-EMFs are transformed from the six-phase domain to the VSD one: a main $\alpha\beta$ subsystem, $(n_{set} - 1)$ harmonic spaces, and n_{set} zero sequence spaces. Than, these are integrated to obtain the fluxes. Note that since zero sequence do not contribute to the electro-mechanical conversion or harmonic behaviour F_0 this quantity is not object of integration and further studies.

Finally, through the Park transformation (1.34), the main subspace is reported in the rotating dq axes.

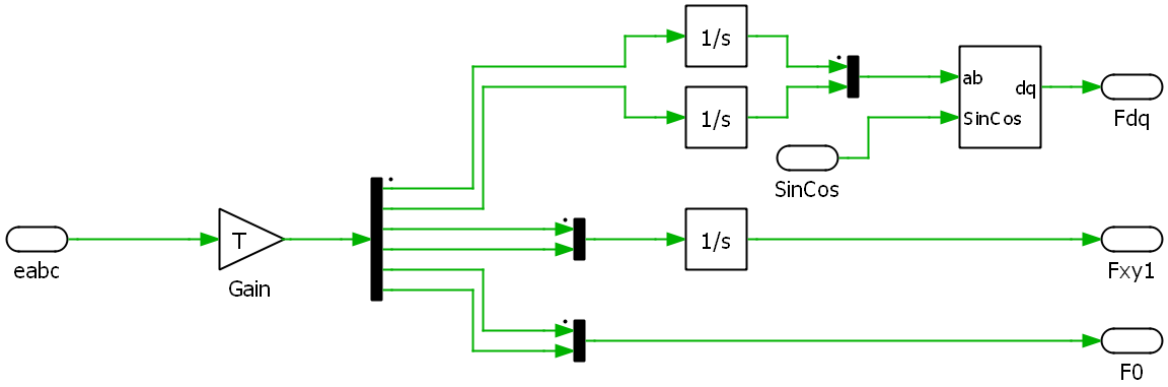

Figure 3.4: Flux integration block scheme

Inverse flux maps

Within the Inverse Flux Maps block, three subsystems can be identified. Two of these subsystems are dedicated to signal management with the goal of exploiting the symmetry of the motor type (SyR, PMSM, IPM) relative to the inverse flux maps. This approach allows for loading reduced-size maps with finer interpolation into the program. As a result, both memory usage and computational resources are optimized, while achieving a qualitatively superior magnetic behavior representation.

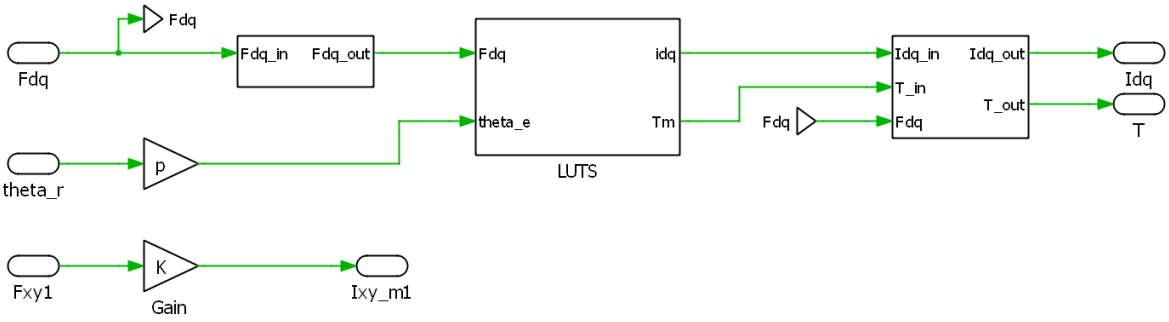

Figure 3.5: Magnetic Model block

The first subsystem (on the left in Figure 3.5) shows different signal treatments depending on the motor type:

- **SyR**: Synchronous Reluctance motors exhibit double symmetry along the dq rotor axes, also due to the absence of magnets. This allows the inverse flux maps to cover only 1/4 of the total definition area, significantly reducing computational demands and enhancing the representation of their magnetic behavior.
- **PMSM**: Surface-mounted Permanent Magnet machines demonstrate magnetic isotropy along the q-axis, perpendicular to the magnet flux direction. This feature enables the halving of the inverse flux maps' definition area.
- **IPM, PM-SyR**: Permanent Magnet machines controlled under the dq convention typical of Synchronous Reluctance motors exhibit magnetic symmetry relative to the d-axis.

These characteristics are succinctly summarized in Figure 3.6.

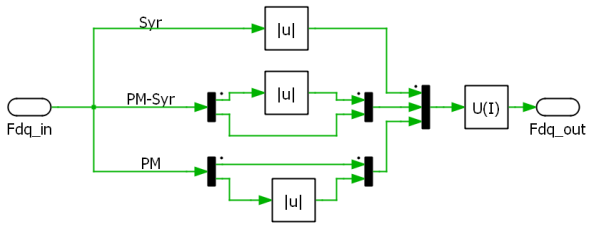

Figure 3.6: Axis configuration choice

Moving forward, within the 'LUTS' subsystem, *dq*-axis currents are calculated based on the flux information as detailed in 1.4.1.

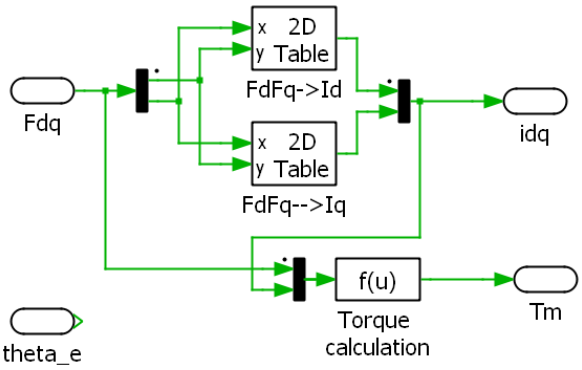

Figure 3.7: 2-D inverse-Flux look up tables

Subsequent steps involve further signal conditioning to ensure correct torque and currents readings for the specific motor type under study. This adjustment is crucial to correct for potential sign errors introduced by the flux map reduction resulting from the symmetrization procedure.

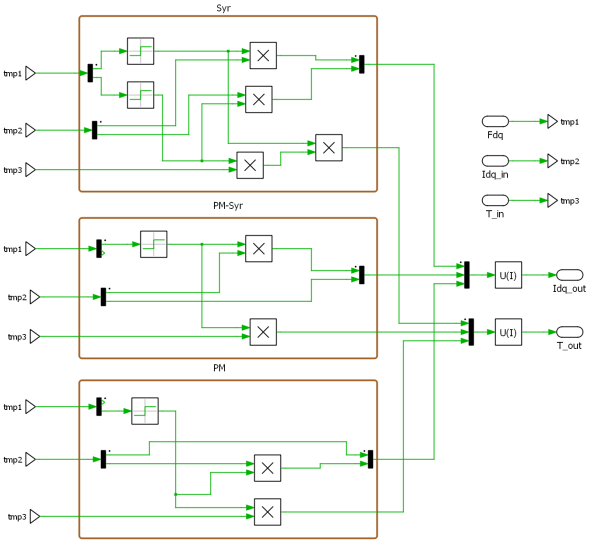

Figure 3.8: Axis reconfiguration after 2D-LUTS

Iron Losses

The calculation of iron losses is based on the approach described in [4].
This loss component can be analogized with an equivalent electrical circuit in the dq axes:

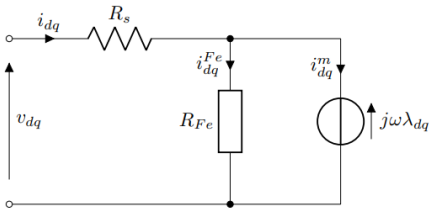

Figure 3.9: Steady state Iron losses model

Hence, iron losses can be likened to a resistance in parallel with the electromotive force of the phases. This non-linear resistance draws part of the current available for electro-mechanical conversion, thereby reducing the system's efficiency.
Within the PLECS model, the Iron Losses block comprises:

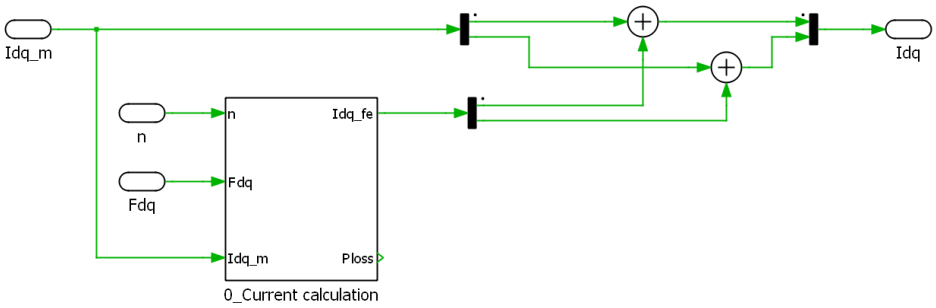

Figure 3.10: Iron losses block

The computation of the current unavailable for electro-mechanical conversion occurs in specific steps within the '0 Current Calculation' subsystem. As a starting point, it is assumed that iron losses can be modeled using Steinmetz's equation:

$$p_{Fe} = k_h \cdot f^{\alpha} \cdot B^{\beta} + k_e \cdot (f \cdot B)^2 \tag{3.1}$$

Here, B represents the flux density within the iron sections of the motor. The coefficients in this equation are determined through a fitting process, which utilizes loss data directly sourced from the manufacturer's datasheet.

The subsequent step for a straightforward iteration of the current calculation is based on the use of 2D iron loss maps. These maps are quantified across the (i_{md}, i_{mq}) domain, akin to flux maps, at a predetermined speed n_0, resulting in two maps, $P_{h,0}$ and $P_{e,0}$, for hysteresis and eddy-current losses, respectively.
These maps functionally describe the losses as dependent on (i_{mdq}) currents, calculated for a specific frequency $f_0 = \frac{n_0 \times 60}{p}$, where p denotes the number of pole pairs. Thus, to calculate losses at a different operating point, values are simply scaled as follows:

$$P_{Fe} = P_{h,0} \cdot \left(\frac{f}{f_0}\right)^{\alpha} + P_{e,0} \cdot \left(\frac{f}{f_0}\right)^{2} \tag{3.2}$$

Another term of losses encapsulated in this study are PM losses. These are caused by the circulation of eddy currents in the permanent magnet blocks. A simplified model is adopted for their calculation, since these are smaller than other losses and the PMs are axially segmented to reduce eddy currents. Here, the effect of eddy currents on the flux distribution is assumed to be negligible, a conservative assumption but relevant for surface magnet machines.
Furthermore, for a conservative and easy approach, PM losses are assumed proportional to ω^2, using this method they may be overestimated.
With these assumptions, this losses are calculated at fixed speed $P_{PM,0}$ and scaled on a grid (ω_m, i_{sqm}). Then, as for Iron losses, for actual computation they are rescaled as:

$$P_{PM} = P_{PM,0} \cdot \left(\frac{f}{f_0}\right)^2 \tag{3.3}$$

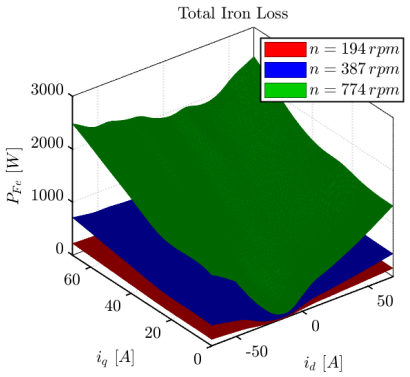

Figure 3.11: Six-phase IL maps

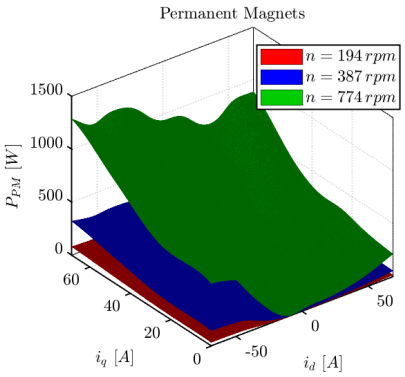

Figure 3.12: Six-phase PM loss maps

Finally, the $I_{dq,fe}$ currents are computed in a C-script function as:

$$\overline{I}_{dq,fe} = \frac{2}{3} \cdot (P_{Fe} + P_{PM}) \cdot j\omega\overline{\lambda}_{dq} \tag{3.4}$$

$$\begin{cases} I_{d,fe} = \frac{P}{\omega \cdot (\lambda_d^2 + \lambda_q^2)} \cdot (-\lambda_q) \\ I_{q,fe} = \frac{P}{\omega \cdot (\lambda_d^2 + \lambda_q^2)} \cdot \lambda_d \end{cases} \tag{3.5}$$

This calculations are performed in the '0 current calculation' subsystem present in Figure 3.13

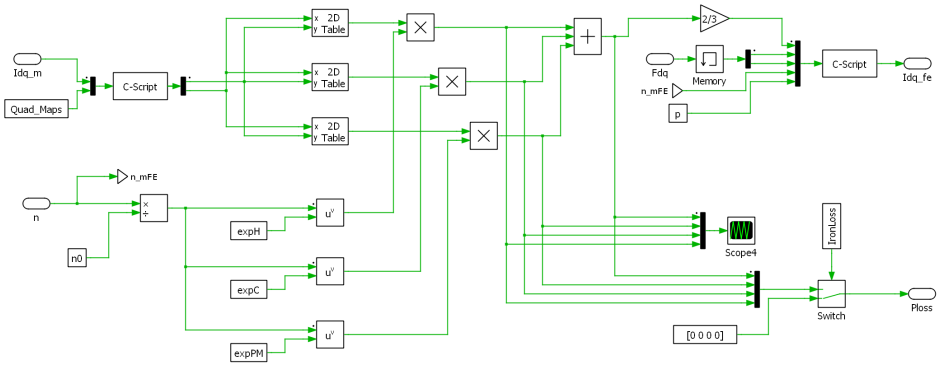

Figure 3.13: Iron Losses current calculation

3.2 Multi-Triphase Vectorized Model

The concept of vectorization within the context of multi-three-phase motors introduces a revolutionary approach to the modeling and simulation of these systems in PLECS. This technique, illustrated in Figure 3.14, embodies a streamlined and efficient representation of complex multiphase machines, facilitating their analysis and control strategy development.

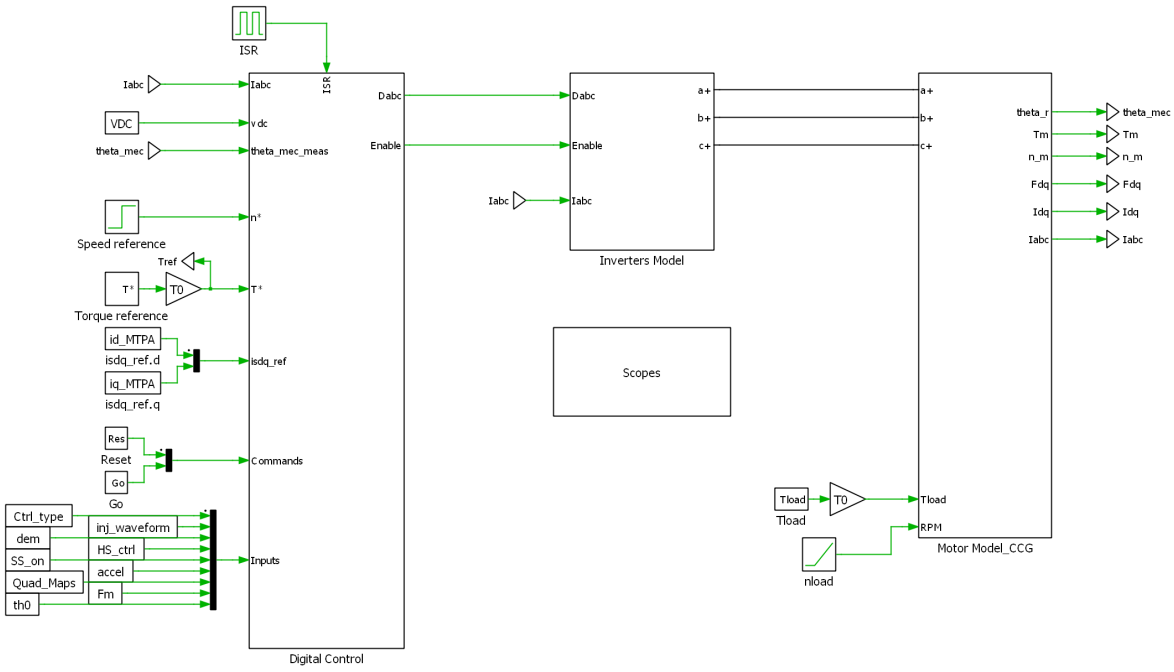

Figure 3.14: Generalized multi-three-phase machine schematic

At a glance, the vectorized model retains the visual simplicity of the single three-phase model previously introduced (1.4.1), yet it is capable of controlling motors with a number of phases that is $3 \cdot n$, times greater, with $n \in N$.
This approach stems from the insights gained through documents such as [5] and [6]. In the former, the unlocking of vectorization element potential is discussed, while the latter presents a practical example of modeling a multiphase synchronous buck converter with a user-variable number of phases.

The vectorized model encapsulates the essence of multiphase systems by abstracting the physical phase connections and control strategies into a unified framework.
This abstraction not only reduces the graphical complexity inherent in modeling multiphase systems but also significantly enhances the computational efficiency of simulations. By employing a single inverter model to manage multiple sets of three-phase systems, the model adeptly mirrors the parallel processing capabilities of modern simulation environments, thereby providing a clear and concise representation of the motor's operational dynamics.

A pivotal advantage of this vectorized approach lies in its inherent scalability and adaptability to various motor configurations and control algorithms. The model's flexibility allows for seamless integration into the SyreDrive environment, enabling the auto-generation of motor models with varying phase configurations, from traditional three-phase to more complex twelve-phase systems. This integration not only streamlines the simulation setup process but also opens new avenues for the exploration of fault tolerance and resilience in multiphase motors.

Moreover, the vectorized model's structure is designed to accommodate the simulation of fault conditions, a critical aspect for the development of robust motor control strategies. By providing a clear framework for the insertion and analysis of faults within the motor system, the model serves as an invaluable tool for the investigation of motor behavior under adverse conditions, thereby contributing to the advancement of fault-tolerant motor design and control methodologies.

The forthcoming sections will delve into the intricacies of developing this vectorized model, shedding light on the technical nuances and the strategic considerations on its implementation. This discussion aims to provide a comprehensive understanding of the vectorized model's architecture and its significant contributions to this thesis.

3.2.1 Inverter Model

At first glance, the inverter model might appear essentially unchanged compared to a standard three-phase, two-level inverter, except for the addition of two pivotal elements. These elements, which are the cornerstone of generating parallel functioning three-phase inverters, are 'Wire Selector' blocks from the PLECS component library.

Vectorization Element

Indeed, this component is capable of performing multiple functions depending on its configuration. Specifically, it can:

- Perform signal conditioning, i.e., swap the positions of input and output variables.
- Select only certain input quantities by specifying their position within the vector (this was utilized, for example, in the signal conditioning in 3.1.2).
- Multiply outputs from a single input signal to create multiple parallel paths, effectively acting as an electrical node to which several parallel branches are connected. This feature has enabled the development of the models discussed herein.

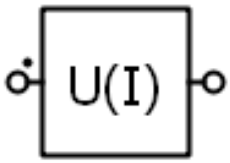

Figure 3.15: Wire Selector Block

Inverter

The result is a structure graphically equivalent to a three-phase 2-Level inverter, as can be seen in Figure 3.16, with the significant difference that each switch shown is receiving a number of signals equal to the number of sets that the motor under analysis possesses.

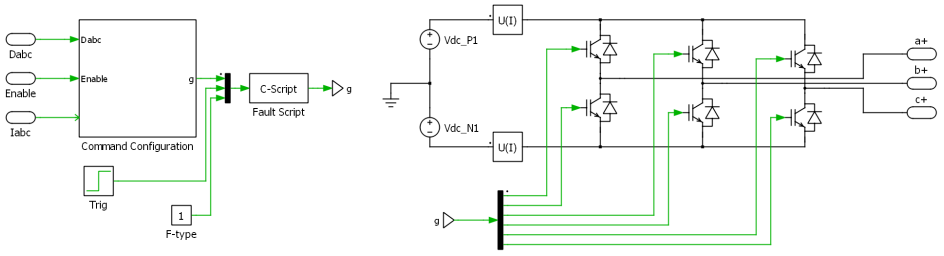

Figure 3.16: Generalized multi-three-phase inverter

Command Signals

Within the inverter subsystem, we find the 'Command Configuration' Block. Inside this block, a comparison is made between the duty cycles and the triangular carrier of the PWM modulation, as shown in Figure 3.17, which results in the generation of the switch gate commands.

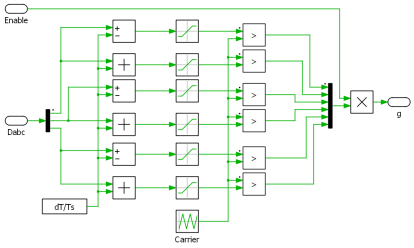

Figure 3.17: PWM generation

The C-Script shown in Figure 3.16 is prepared for the generation of faults that can be simulated on the inverter side. Further developments regarding this component is carried out in the course of this discussion.

3.2.2 Motor Model

As expected the motor model 3.18, here in the version with controlled current generators, appears exactly identical to the one presented in 1.4.1 of the three-phase motor.
Indeed, from the integration of the electromotive forces in the dq axis inside the 'Flux Integration' block, until the controlled currents are reiterated in the phase domains $a_k b_k c_k$, the calculations are almost identical to what was seen for the extended hexaphase motor. Said that, reference can be made to what was presented in 3.1.2.

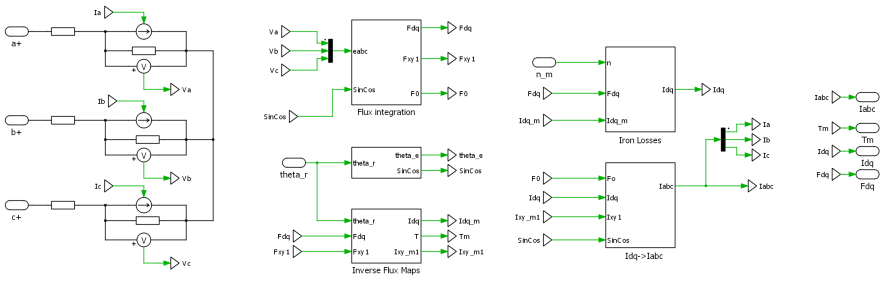

Figure 3.18: Vectorized circuital model 3CCG

Clearly, the model includes a single set of Controlled Current Generators, in which the n_{set} are solved in parallel.

3.3 Control Strategy

As already exposed in 1.6.2, the control of three-phase and multi-three-phase machines is conducted through the Multi Stator approach to maintain the modularity of these drive systems and ensure efficient response to various simulated faults. To address the cross-coupling issue between the three-phase systems composing the stator windings, decoupling algorithms is implemented. This operation make more easy understanding of the machine, simplifying the analysis under fault conditions. After appropriate transformations, the machine is controlled in the common-mode axes dq, simplifying fault analysis. A voltage feedforward is employed to support proportional-integral controllers, offering better compensation and quicker response to torque references. Clarke transformations from $a_k b_k c_k$ axes to $\alpha_k \beta_k$ axes is utilized, exploiting the asymmetry of the studied electric motors. Additionally, there is no significant difference in the definition of the Park transformation from $\alpha_k \beta_k$ to d_k, q_k.

3.3.1 Digital Control

Digital control is presented in a way similar to three-phase control, but with the difference that the input currents and duty cycles, along with their Enable signals, have a multiplicity equal to the number of sets of the studied e-motor.

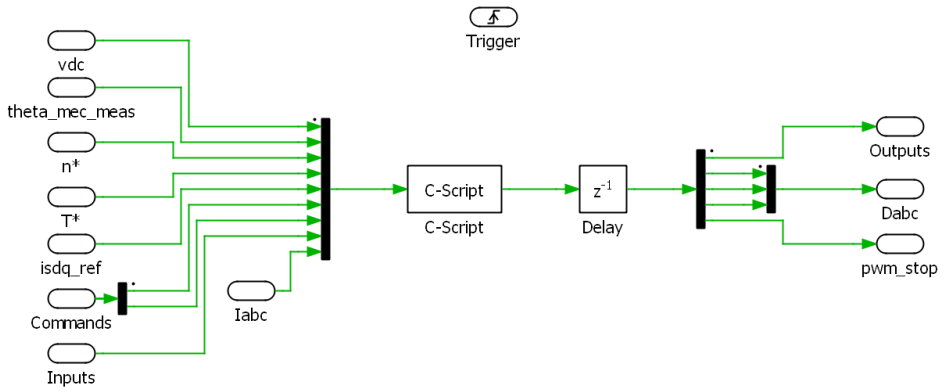

Figure 3.19: Digital control block

Placing these quantities in the last positions of the 'multiplexer' (or demultiplexer) component, as shown in Figure 3.19, ensures better signal management. This is because all the magnitudes preceding them do not have multiplicity and are therefore invariant in relative position to the number of three-phase sets.
Choosing otherwise would means to necessary modifying the positions of these magnitudes depending on the sets present, making the discussion unnecessarily complicated.

3.3.2 Decoupling Algorithm

This study use the approach suggested in [9], focusing on the computation of both common and differential modes of the machine.
The main purpose of this study is to isolate the energy conversion process into a single common-mode subspace. Meanwhile, any imbalances among three-phase sets concerning flux and torque production

are identified and addressed within designated differential-mode subspaces.
Starting from the coupled model 1.5, the final representation of the decoupled machine should be like:

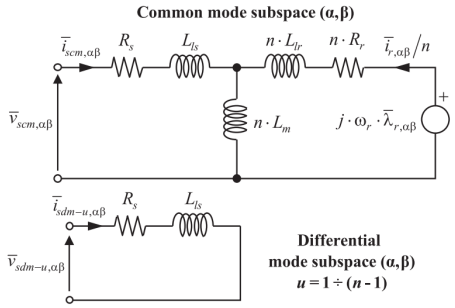

Figure 3.20: MS equivalent circuit after decoupling [9]

To achieve this goal, the following assumption are made:

- For a generic stator variable of the machine z_s exist one common mode vector $z_{s,cm}$ and $u = [1 : (n-1)]$ differential modes $z_{s,dm-u}$
- It is possible to demonstrate that the z_s variable can be expressed as a linear combination of common and differetial mode as:

$$\bar{z}_{sk,xy} = \bar{z}_{scm,xy} + w_k \cdot \bar{z}_{sdm-k,xy} + \sum_{u=1}^{u=k} (q_u \cdot \bar{z}_{sdm-u,xy}) \tag{3.6}$$

  Where the q_k and w_k coefficients are

$$w_k = \sqrt{\frac{n \cdot (n-k)}{(n-k+1)}}, \quad q_u = -\sqrt{\frac{n}{(n-u) \cdot (n-u+1)}} \tag{3.7}$$

It is remarkable that the calculation of the decoupling transformation remains independent of the angular displacements among three-phase winding sets. This characteristic circumvents the constraints associated to the Vector Space Decomposition method in the needs of spatial distrubution.

This strategy enables effective decoupling actions within any arbitrary rotating frame, thereby facilitating its integration into various MS-based control schemes. By segregating the energy conversion and addressing imbalances separately, our method enhances control precision and flexibility across different operational scenarios.

Although the resulting equation systems of the decoupled multi-stator model and the VSD approach may appear similar, their underlying mathematical and physical interpretations diverge significantly. The VSD methodology employs a time-harmonic dissection of the machine's spatial configuration, representing energy conversion through an averaged representation within the principal subspace, signifying the fundamental temporal model of the machine. Conversely, in the Multi Stator (MS) approach, the collective and differential models are derived through linear combinations of MS variables, thereby preserving the modularity of energy conversion within the time-fundamental subspaces of the machine.

3.4 Integration into SyreDrive

The integration process into SyreDrive begins by creating a repository containing the PLECS model of the generalized multiphase machine, for which the number of motor phases is not yet defined. Besides the PLECS model, the following should be present:

- The 'SimMatFiles' folder, initially empty, where motor flow maps, machine parameters, and user settings (user settings) is uploaded in Matlab table format.
- The 'MotorControl0' file, a file containing parts of the control code that are independent of the e-motor under examination and that are reported for all, which is better addressed in **??**.

The integration into SyreDrive essentially passes through the definition of the '.m' file that is called when, once the motor model and the characteristics that the drive must possess (i.e., instantaneous/average model, switching frequency, on-resistance of the switches, threshold voltage, etc.) are selected, the user presses the 'create PLECS model' button on the SyreDrive graphical interface, shown in figure 1.14. This file performs specific tasks aimed at the final realization of the project.

This script initiates by ensuring that the motor model has all the necessary maps for control trajectories, inverse models for dq and $dq-t$ domains, and incremental inductance maps. If any of these are missing, they are evaluated and generated.

Subsequently, a new project folder is created, and the base PLECS model is copied into this folder, customized with the specific motor name. The motor model data is saved within this folder, and motor data are printed into a header file for the control script.

The script then considers the motor type and axis convention to determine the quadrant maps for control strategies. Following this, several scripts are executed to generate the control script, decoupling function, Clarke transform, Vector Space Decomposition Matrix, and fault simulation script. Finally, the SyreDrive simulation path is updated, and the simulation initialization script is run, completing the process of integrating the motor model into SyreDrive for simulation.

```
%% Copyright 2022
%%
%%   Licensed under the Apache License, Version 2.0 (the "License");
%%    you may not use this file except in compliance with the License.
%%   You may obtain a copy of the License at
%%
%%       http://www.apache.org/licenses/LICENSE-2.0
%%
%%   Unless required by applicable law or agreed to in writing, dx
%%  distributed under the License is distributed on an "AS IS" BASIS,
%%    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
%%    See the License for the specific language governing permissions and
%%    limitations under the License.



function motorModel = MMM_createPLECSmodel(motorModel)

%% Check if motormodel already has maps and if it doest't have them, are evaluate
if isempty(motorModel.controlTrajectories)
    motorModel.controlTrajectories = MMM_eval_AOA(motorModel);
end
if isempty(motorModel.FluxMapInv_dq)
```

```
    motorModel.FluxMapInv_dq = MMM_eval_inverseModel_dq(motorModel);
end
if isempty(motorModel.FluxMapInv_dqt)
    motorModel.FluxMapInv_dqt = MMM_eval_inverse_dqtMap(motorModel);
end
if isempty(motorModel.IncInductanceMap_dq)
    motorModel.IncInductanceMap_dq = MMM_eval_inductanceMap(motorModel);
end


%%Create folder for new project
ctrlFolder_path = [motorModel.data.pathname motorModel.data.motorName '_ctrl_PLECS'];
%%Syre folder
syrePath = fileparts(which('GUI_Syre.mlapp'));
%%Copy and migration of the base model in the new project folder
copyfile([syrePath '\syreDrive\PLECSmodel_mf'], ctrlFolder_path);
movefile([ctrlFolder_path '\Generalized_multiphase_machine.plecs'],[ctrlFolder_path '\
    ' motorModel.data.motorName '_Motor_ctrl.plecs']);
%%Save of the selected motor model on the project folder
save([ctrlFolder_path '\motorModel.mat'],'motorModel');
%%Print of the motor data on h file
MMM_print_MotorDataH_PLECS(motorModel);

%%n_set and axis style information
n_set=motorModel.data.n3phase;
if strcmp(motorModel.data.axisType,'SR') && strcmp(motorModel.data.motorType,'SR')
    Quad_Maps = 0; %SyR Convention - 1st quadrant maps
elseif strcmp(motorModel.data.axisType,'SR') && strcmp(motorModel.data.motorType,'PM')
    Quad_Maps = 1; %PM-SyR - 1st and 4th quadrant maps
elseif strcmp(motorModel.data.axisType,'PM') && strcmp(motorModel.data.motorType,'PM')
    Quad_Maps = 2; %IPM - 1st and 2st quadrantsmaps
end
%%Print of the control script
print_PLECS_control_script_2(ctrlFolder_path,n_set,Quad_Maps);
%%Print of decoupling funcion
ComputeDecouplingMatrix(ctrlFolder_path,n_set);
%%Print of Clarcke transform
PrintClarke(ctrlFolder_path,n_set);
%%Print of Vector Space Decomposition Matrix
Compute_TVSD(ctrlFolder_path,n_set);
%%Create C-Script to simulate faults
print_PLECS_fault_script(ctrlFolder_path,n_set);
%%ending process
motorModel.SyreDrive.SIM_path = [ctrlFolder_path '\' motorModel.data.motorName '_Model
    .plecs'];
run([ctrlFolder_path '\init_sim_PLECS.m']);
end
```

3.4.1 Print Control Script

This subsection explores the methodology for generating the `Motor_Ctrl_Script.c` for any three-phase or multiphase motor, illustrating a sophisticated framework that combines a MATLAB script with a generic C template (`Motor_Ctrl_0.c`). This approach is instrumental in automating the creation of a motor control script tailored to specific motor configurations, significantly streamlining the simulation setup in PLECS.

The foundament of this automated script generation lies in the $Print_Control_Script.m$ file, which orchestrates the control logic assembly. This script intelligently integrates modular code segments from $Motor\_Ctrl\_0.c$, adjusting these snippets to match the unique specifications and operational parameters of the target motor.

This process is ensuring that the control script accurately reflects the dynamics and control requirements of various motor designs, from standard three-phase to more complex multiphase configurations.

The Matlab script initiates by reading the `Motor_Ctrl_0.c` file, which contains generic motor control algorithms and state machine logic. It then dynamically modifies and expands this code base, adding specific control parameters, input signal processing, and state transition logic relevant to the motor under consideration. This customization includes adapting the script to handle different numbers of phases and incorporating specific control strategies such as current, torque, speed, and flux control.

The generated `Motor_Ctrl_Script.c` is seamlessly integrated into the PLECS simulation environment, serving as the digital control core for the simulated motor. This integration enables the simulation to mirror real-world motor control scenarios accurately, providing valuable insights into the motor's performance, operational efficiency, and response to various control strategies.

This automated script generation technique offers significant advantages, including reduced development time, increased flexibility in motor control strategy exploration, and enhanced accuracy in simulating motor behavior. It is particularly beneficial in educational and research settings, where rapid prototyping and testing of different motor control strategies are essential.

The *Print Control Script* process exemplifies a practical and efficient approach to motor control script generation in PLECS. By leveraging the power of MATLAB scripting and generic C code templates, it facilitates the detailed study and simulation of a wide range of motor types, advancing the field of motor control simulation and design.

3.4.2 Compute Vector Space Decomposition Matrix

The algorithm for computing the Vector Space Decomposition (VSD) matrix is a cornerstone for analyzing and controlling asymmetrical multiphase machines. This process, inspired by a foundational paper on the topic [14], translates complex theoretical concepts into a practical MATLAB implementation.

The VSD matrix plays a crucial role in dissecting the spatial orientation and harmonic content within multiphase electric machine systems. Each row of the VSD matrix delineates a distinct two-dimensional subspace, constructed using the phase propagation angle vector $[\theta]$ and a specific subspace constant C. These subspaces are initially represented in a complex format, which can be equivalently expressed by real values, simplifying the construction of the generalized multiphase Clarke's transformation matrix.

In particular, for asymmetrical multiphase machines, the zero-sequence components (ZS) are predominantly represented as matrices filled with zeros. This simplification is applicable for each motor presented in this lecture except for symmetric three-phase machine, where the conventional Clarke's transformation is adequately sufficient due to the non-existence of subspaces. The creation of the VSD matrix involves determining the subspace constants C, which play a pivotal role in mapping the harmonics into their respective subspaces.

The script, fully reported in 4.5.2, calculates the $[T_{VSD}]$ matrix, leveraging the defined subspace constants and phase positions to create a comprehensive mapping of the electrical system's spatial and harmonic characteristics. This matrix forms the foundation for the generalized multiphase Clarke's transformation, facilitating a deeper understanding and more effective control of multiphase electric

machines.

This MATLAB implementation is not just a computational tool but a bridge between theoretical exploration and practical application, enabling engineers to navigate the complexities of asymmetrical multiphase systems with greater precision and insight. The generated $[T_{VSD}]$ matrix is subsequently saved, ready for integration into simulation and control algorithms that drive the next generation of electric machine analysis and design.

$$e^{jC(\theta)} = \begin{bmatrix} Re[e^{jC(\theta)}] \\ Im[e^{jC(\theta)}] \end{bmatrix} = \begin{bmatrix} \cos(C(\theta)) \\ \sin(C(\theta)) \end{bmatrix} \tag{3.8}$$

The subspace harmonic mapping is determined by the subspace constant C. For positive integers $i = 1, 2, 3, \ldots$, C can be defined as follows:

1. $C \neq i \cdot k$ – Non-zero-sequence harmonics are mapped into these subspaces $[T_{nZS}]$ (e.g., $k = 3$, $C = 1, 2, 4, 5, \ldots$).
2. $C = i \cdot k$ – Zero-sequence harmonics are mapped into these subspaces $[T_{ZS}]$ (e.g., $k = 3$, $C = 3, 6, 9, \ldots$).
3. $C = n/2$ or $C = n$ – A set of real values is produced, the imaginary part is zero, thus forming a homopolar zero-sequence component $[ZS]$ (e.g., $n = 9$, $C = 9$).

The rows are systematically arranged such that the initial subspaces are non-zero-sequence subspaces, followed by zero-sequence subspaces, and concluding with zero-sequence homopolar components.

The definition of subspace constants ensures that all odd-order harmonics are uniquely mapped into their respective x-y planes, crucial for the analysis of asymmetrical machines.

Although the discussion here is tailored towards asymmetrical machines, the underlying algorithm can accommodate symmetrical machines, underscoring the algorithm's versatility and comprehensive approach to analyzing multiphase systems.

The MATLAB code snippet provided is instrumental in computing the VSD matrix, encapsulating the theoretical concepts into a practical computational tool. This code effectively automates the process of generating the VSD matrix for any given multiphase system, streamlining the analysis and enhancing the understanding of complex multiphase electric machines.

The developed code can be integrally found in Appendix D to fully address this topic.

3.4.3 ComputeDecouplingMatrix

This section presents the Matlab function code, reported in the appendix E, incorporated into SyreDrive for printing the decoupling algorithm and the Clarke transformation into the 'User_Macros.h' file.

- **Initialization:** The function starts by determining the path to the 'User_Macros.h' file within the control folder. It then sets up the number of phase sets (n) which directly influences the complexity and size of the decoupling matrix.
- **Decoupling Matrix Calculation:** A decoupling matrix $[TD]$ is computed based on the number of phase sets. This matrix is crucial for transforming the d-q parameters from individual

phase sets into a common-mode and differential-mode subspace, facilitating the analysis and control of multiphase systems.

- **Macro Definition for Direct Decoupling:** The function dynamically generates a macro within the 'User_Macros.h' file. This macro, _Decoupling, abstracts the mathematical transformation of d-q parameters from individual phase sets to a common and differential mode, easing the implementation of control algorithms.
- **Common Mode Subspace Calculation:** It calculates the common mode subspace parameters, which represent the average behavior of the system, ignoring the differential effects between phases.
- **Differential Mode Subspace Calculation:** For each phase set, it computes the differential mode subspace parameters, capturing the unique characteristics and discrepancies among the phases.
- **Inverse Decoupling Algorithm:** This part of the function reverses the decoupling process, transforming parameters from the common and differential mode back to the original d-q parameters of each phase set. This is essential for applying the calculated control actions back to the physical system.
- **Macro Definition for Inverse Decoupling:** Similarly to the direct decoupling, a macro for inverse decoupling (_InvDecoupling) is defined, facilitating the reverse transformation process in the control code.
- **Finalization:** The computed decoupling and inverse decoupling macros are written into the 'User_Macros.h' file, updating the file with the necessary algorithms for handling the electric motor's multiphase control system effectively.

By automating the generation of these critical algorithms and incorporating them directly into the control system's codebase, the ComputeDecouplingMatrix function significantly simplifies the development and implementation of advanced control strategies for multiphase electric motors within the SyreDrive framework.

3.4.4 PrintClarcke

The `PrintClarke` function within SyreDrive is designed to generate and insert the generalized Clarke transformation for asymmetrical multi-three-phase motors and its inverse into the `User_Macros.h` file.

This operation is fundamental for handling the conversion between phase currents and the alpha-beta reference frame and vice versa, a critical aspect in the control of multiphase electric motors. Here's a breakdown of the process, explained step by step:

This function automates the inclusion of the Clarke transformation and its inverse into the `User_Macros.h` file, facilitating the control system's interaction with multiphase electric motors.

- **Initialization:** The function determines the path to the `User_Macros.h` file. It then calculates the total number of phases (n) based on the number of phase sets (l) and the phases per set (k).
- **Theta Matrix Calculation:** It computes the matrix θ_n that contains the electrical angles of each phase, crucial for determining the transformation coefficients.
- **Clarke Transformation Calculation:** For each phase set, the function calculates the direct Clarke transformation matrix. This matrix transforms the three-phase currents (a, b, c) into the two-dimensional alpha-beta reference frame.

- **Inverse Clarke Transformation Calculation:** Similarly, it computes the inverse Clarke transformation for each phase set, enabling the reverse conversion from the alpha-beta reference frame back to the three-phase currents.
- **Macro Definitions for Transformation:** The function dynamically generates macros for both the direct and inverse Clarke transformations. These macros are designed to simplify the application of the transformations within the control code.
- **Update to `User_Macros.h`:** The newly created macros are added to the `User_Macros.h` file. This includes both the direct transformation, which converts phase currents to the alpha-beta frame, and the inverse transformation, which does the opposite.
- **Finalization:** After updating the `User_Macros.h` file with the necessary macros, the function closes the file. This completes the integration of the Clarke transformation into the control system's codebase.

By embedding these transformations directly into the control system's macros, the `PrintClarke` function significantly streamlines the implementation of control algorithms for multiphase motors. It ensures that the system can efficiently handle the conversion between different reference frames, a critical operation for accurately controlling motor behavior and analyzing its performance.

The developed code can be integrally found in Appendix F to fully address this topic.

3.5 Hardware-in-the-loop

For the HIL implementation of the developed model, it was decided to create an ad hoc project dedicated exclusively to the three-phase motor THOR_1x3ph. This choice was determined by the need to simplify the process by using a single microcontroller to generate the inverter's switching signals. Even when considering only the three-phase motor, the configuration of the advanced TIMER1 timer and the definition of six output channels for signal generation and five input channels for quantity measurements is required.

The Hardware-In-The-Loop modelling of the vectorised model similarly follows what was seen for the offline simulations setup: the parallelisation of the resolution of multiple three-phase systems takes place via wire selector blocks. The main differences lies in the import/export of input/output quantities to the control system implemented in the microcontroller.

In order to create correct analogue-to-digital conversion of signals, the voltage scale ranges must be configured for the microcontroller used in RT-BoX and the NUCLEOG474RE microboard in the same way,in order for them to match. In particular, the adopted microboard accepts and sets signals on a scale from 0 to $3.3V$. The following table shows the signal scaling operations performed in the HIL project of the THOR_1x3ph motor:

Table 3.1: Singals Scaling Operations for Output HIL Testing

Signal Type	Full Range	Scaling Factor	Offset
Peak to peak current (I)	150 A	$I_{\text{scl}} = \frac{3.3}{I_{\text{FR}}}$	$I_{\text{off}} = \frac{3.3}{2}$
DC Voltage (V_{dc})	310 V	$V_{\text{scl}} = \frac{3.3}{V_{\text{FR}}}$	$V_{\text{off}} = 0$
Angular Position (θ)	7 (2π ceil rounded)	$\theta_{\text{scl}} = \frac{3.3}{\theta_{\text{FR}}}$	$\theta_{\text{off}} = \frac{3.3}{2}$

After having correctly set the scales for the measured quantities, it is necessary to define the pins that connect the printed circuit board of the Rt-Box with the pins configured in the nuceoboard.

As far as the output quantities from RT-BOX are concerned, the defined pins are:

Table 3.2: RTBox Output Pin Configuration

Signal	**Channel Number**
Current I_a	0
Current I_b	5
Current I_c	4
Angular Position θ	6
DC Voltage V_{dc}	1

With regard to the input quantities to RT-BOX, i.e. the switching signals of the inverter legs, the following pins are defined:

Table 3.3: RTBox Input Pin Configuration

PWM Signal	**Commanded Switch**	**Switch in Channels**
TIM1 Channel 1	Upper A phase	0
TIM1 Channel 1N	Lower A phase	1
TIM1 Channel 2	Upper B phase	2
TIM1 Channel 2N	Lower B phase	3
TIM1 Channel 3	Upper C phase	4
TIM1 Channel 3N	Lower C phase	5

3.5.1 Cube IDE Project

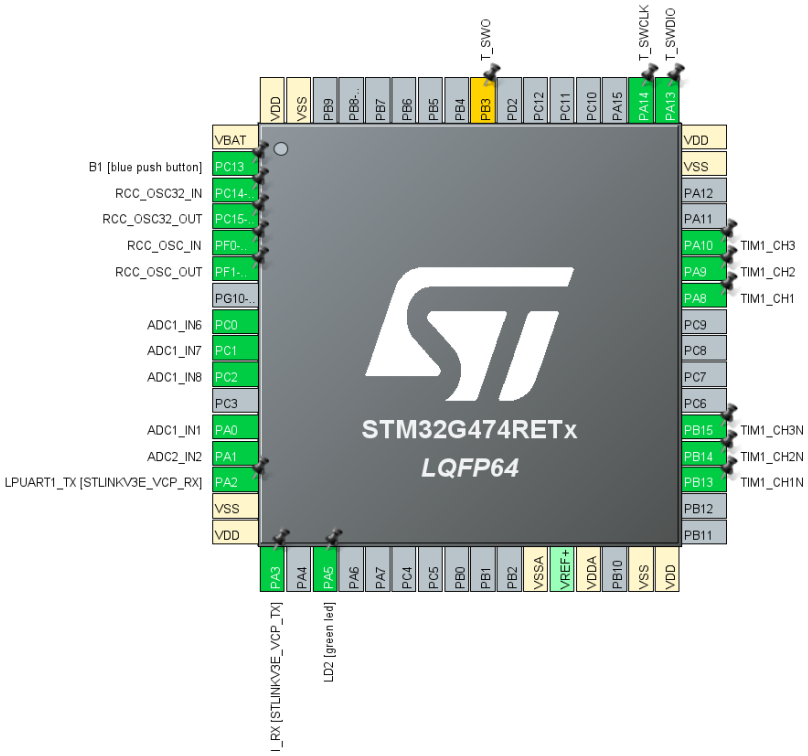

Figure 3.21: Cube Project - Pinout wiew

The project is based on the advanced configuration of an STM32G4 microcontroller, oriented towards the precision control of a three-phase motor at a frequency of $10kHz$.
ation, which is examined in more detail in this section. The realisation of the project was carried out in CubeIDE, an integrated development environment in which it is possible to have total project management right from the hardware setup. The file `NUCLEOG474RE_THOR.ioc` contains the design of the microcontroller configur In figure 3.21 is presented the pinout wiew of the project.From this wiew it is clear witch pins have been configured and what is their purpose. For this project is made use of two ADC (ADC1 and ADC2) for digital signal conversion of analogical quantities. Furthermore, the command signal are realized using accurate configuration of TIMER 1.

Configuration of TIMER 1

The **TIMER 1** plays a crucial role in generating PWM control signals for the three-phase motor, with a detailed configuration that supports the modulation of switch-switch signals on three channels (CH1, CH2, CH3) and uses the fourth channel (CH4) to generate a trigger for the **ADC** that allows the sampling of currents in readiness for the execution of the ISR, thus synchronising the sampling of measurements with the motor control routine execution.

This approach ensures that the readout of critical motor control variables, such as phase currents and mechanical angle, is closely aligned with current motor states, optimising the response of the control system.

The configuration of the **TIMER 1** registers is carried out setting up TIM1 registers: the`Prescaler` and `AutoReload Register (ARR)`, which determine the frequency of the PWM signals, and the `Repetition Counter (RCR)`, which further refines the accuracy of the PWM timing and trigger for the ADC.

Its main configurations are as follows:

- **Counting Mode:** Set to TIM_COUNTERMODE_CENTERALIGNED3, this mode allows finer time management, critical for generating symmetrical PWM signals with a triangular up/down carrier.
- **Preload of Auto-Reload Register (ARR):** Enabled, improving PWM frequency accuracy by dynamically changing the running period.
- **Prescaler:** Set to 0, TIMER1 uses the input clock directly, maximising the resolution of the timer.
- **Auto-Reload Register (ARR):** Valued at 8500, this defines the period of TIMER1 and consequently the frequency of $10kHz$ of PWM signals.
- **Repeat Counter (RCR):** With a value of 17, specifies the number of timer cycles before generating an update, thus refining event handling.
- **Dead Time:** Set to 149, this parameter is crucial in preventing overlaps between complementary PWM signals, protecting the system.
- **PWM Channels:** CH1 CH2, e CH3 are configured for the generation of switch signals, while CH4 is reserved to produce a syncronized trigger event for ADC conversion, so it does not produce any output.

ADC configuration

For monitoring phase currents, mechanical angle and voltage at the DC bus, the system uses two specifically configured ADCs.

ADC1 is configured to sample phase currents and mechanical angle, using injected conversion to ensure maximum accuracy and timeliness of the reading, while **ADC2** is reserved for measuring voltage at the DC bus.

The configuration of **ADC1** provides four injected conversions, allowing the simultaneous measurement of phase currents (i_{sa}, i_{sb}, i_{sc}) and mechanical angle (θ_{mec}), with channel four configured to sample θ_{mec} with improved accuracy, as indicated by the `InjectedChannel` and `InjectedSamplingTime` settings. The **ADC2**, with an injected conversion, is optimised for reading the voltage at the DC bus (V_{dc}), highlighting the importance of this measurement for inverter control and energy management.

The Analog to Digital Converter (ADC) plays a critical role in collecting accurate data for motor control. The configuration specifications of the ADC are as follows:

- **Injected Conversions Enabled:** ADC1 enables injected conversions, allowing prioritisation of conversion of specific channels for fast and accurate readout.
- **Regular Conversions Disabled:** For ADC1, regular conversions are disabled, focusing exclusively on injected conversions for critical data.

- **External Trigger for Injected Conversions:** Uses ADC_EXTERNALTRIGINJEC_T1_TRGO, indicating that injected conversions are synchronised with an external trigger from TIMER1, for perfect alignment with motor control events.
- **Number of Injected Conversions:** Set to 4, allowing simultaneous measurement of several critical parameters.
- **Configured Channels for Injected Conversions:** Channels 1, 6, 7, and 8 are selected for injected conversions, optimising the system for collecting specific data without offsets.
- **Sampling Times for Injected Conversions:** Each channel has a detailed sampling time, ensuring measurement accuracy.

The choice of using channel four of **TIMER 1** to trigger the **ADC** was driven by the need to closely synchronise the sampling of measurements with the exact phase of the motor control cycle. This configuration allows for the acquisition of accurate current, mechanical angle and voltage data at the DC bus at critical times, facilitating the implementation of advanced control algorithms that can dynamically react to real-time motor and inverter conditions.

In conclusion, the detailed configuration of the STM32G4 microcontroller, as defined in the `NUCLEOG474RE_THOR.ioc` file and enriched by the provided design specifications, demonstrates a holistic approach to three-phase motor control. Through intelligent hardware configuration and careful synchronisation between critical components, the project aims to realise an efficient and responsive control system for advanced industrial applications.

3.6 Simulation Results

This section presents the outcomes of simulations conducted on multi-three phase motors under healthy conditions, highlighting the capability of the developed framework for auto-generating machine models within the Syre environment. The primary focus is on demonstrating the accuracy, efficiency, and reliability of the simulation models, which are essential for validating the design and control strategies of multi-three phase electrical drives.

Through the implementation of the Syre-driven automated model generation process, a range of multi-three phase motors was subjected to a comprehensive set of simulations. These simulations were aimed at verifying the operational characteristics of the motors, such as torque production, power efficiency, and electromagnetic behavior, under various loading conditions and operational modes.

The results obtained from these simulations underscore the robustness of the auto-generated models in replicating the expected performance metrics of the multi-three phase motors. Notably, the torque-speed curves, the current and duty measurements derived from the simulations exhibited excellent agreement with theoretical predictions, thereby validating the effectiveness of the Syre integration for motor design and analysis.

Moreover, the process of model auto-generation facilitated by Syre significantly streamlined the simulation setup, reducing the time and complexity involved in modeling sophisticated multi-three phase drive systems. This efficiency not only accelerates the design cycle but also enables a more iterative approach to optimizing motor configurations and control algorithms.

In conclusion, the simulation results affirm the capability of the proposed framework to accurately model and analyze the performance of multi-three phase motors. The successful demonstration of auto-generated models operating under healthy conditions sets a solid foundation for further investigations into fault conditions and resilience strategies, which are crucial for advancing the reliability and durability of these motors in real-world applications.

For each simulation in this section, the same procedure is followed:

- At time 0,03 s the Go button is pushed for the first time and the status of the state machine changes from ERROR to WAKE UP.
- After $4 \cdot T_s$ the status goes from WAKE UP to READY.
- At time 0.1 s the Go button is pushed again and the drive is already controlling the e-motor actively in the START mode.
- At time 0.15 s the torque reference switches from 0 to the nominal value for each e-motor.
- At time 0.17 s a speed ramp is commanded to an acceptable final speed value that depends on the motor type.
- At time 0.8 s the torq reference switches from $+T_{nom}$ to $-T_{nom}$.

THOR_1x3ph Waveforms

The THOR electric motor, a three-phase machine, is subjected to detailed simulation to observe its performance under nominal operating conditions. The simulation results showcase the motor's torque response, current waveforms, duty cycles, flux linkages in d and q axes, and the direct and quadrature currents under a nominal torque setting.

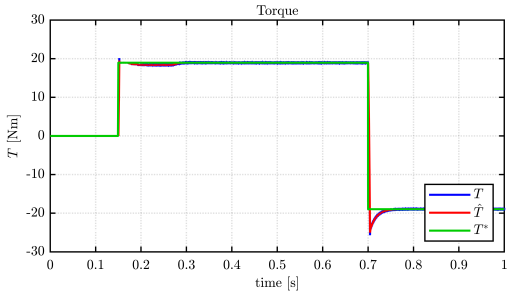

Figure 3.22: Threephase motor Torque graph

This graph demonstrates the motor's accurate torque tracking to the nominal value, indicating optimal performance.

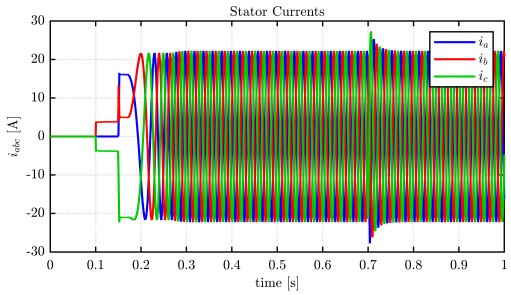

Figure 3.23: Threephase motor - Currents at 1500 rpm, Nominal Torque

The current waveforms remain sinusoidal and at the nominal value, confirming the motor's efficiency in torque delivery.

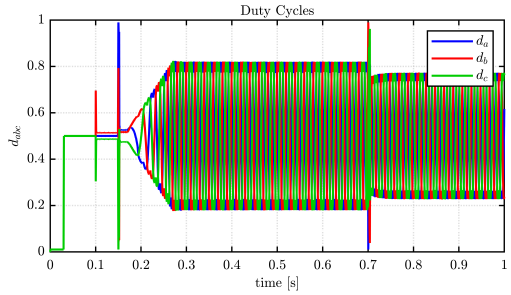

Figure 3.24: Threephase motor - Duty Cycles at 1500 rpm, Nominal Torque

Duty cycles are not saturated, revealing correct motor operation for this workload.

These zoomed-in views provide a detailed examination of the motor's response, highlighting the precision of current management and duty cycle adjustments.

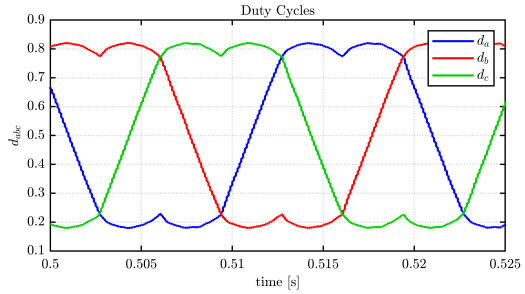

Figure 3.25: Zoom on Duty Cycles

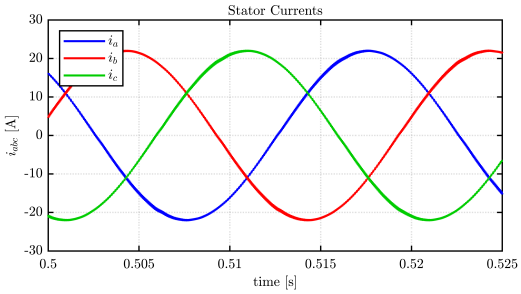

Figure 3.26: Zoom on Currents

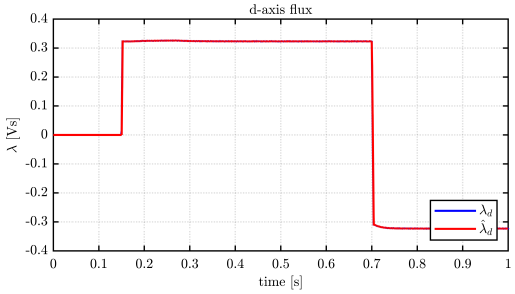

Figure 3.27: Threephase motor - λ_d

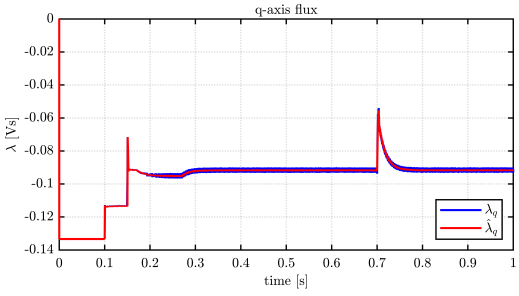

Figure 3.28: Threephase motor - λ_q

Flux linkages in d and q axes are calculated accurately by the control algorithm, matching the observed simulation values.

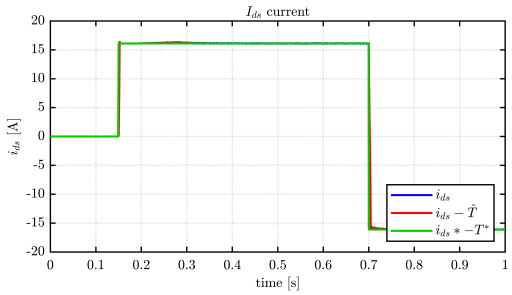

Figure 3.29: Threephase motor - I_d

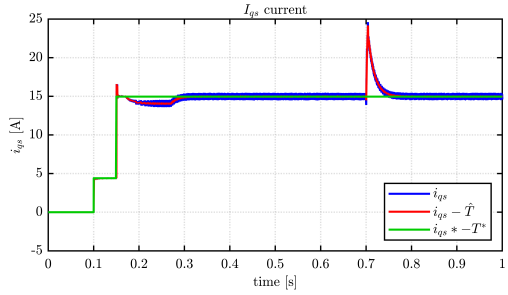

Figure 3.30: Threephase motor - I_q

The direct and quadrature currents further confirm the motor's ability to adhere to the control demands, ensuring efficient torque production and motor performance.

THOR_3x3ph Waveforms

The PMSM_2x3ph, a six-phase electric motor, is analyzed under nominal conditions to assess its performance. The following figures depict the motor's behavior under a nominal torque condition, demonstrating its adherence to the torque reference, the sinusoidal nature of the current waveforms, the optimal duty cycle operation, and the accurate calculation of flux linkages and direct/quadrature currents.

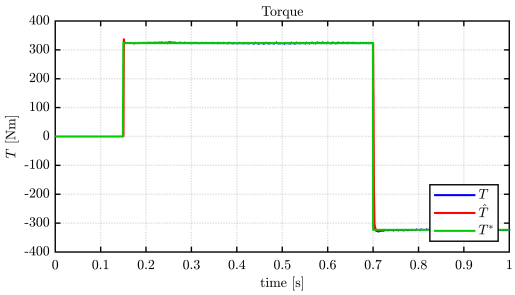

Figure 3.31: Six phase motor Torque graph

The torque graph indicates the motor's precise response to the torque reference, showcasing effective torque management.

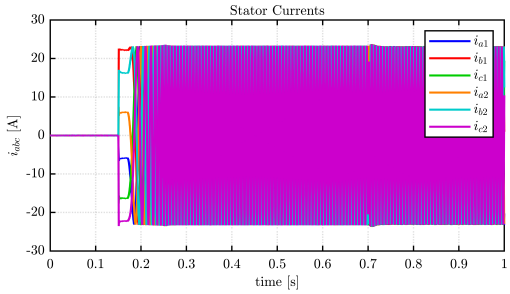

Figure 3.32: Six phase motor - Currents at 100 rpm, Nominal Torque

Current waveforms are sinusoidal, maintaining nominal value, which is expected given the motor's operation under nominal torque.

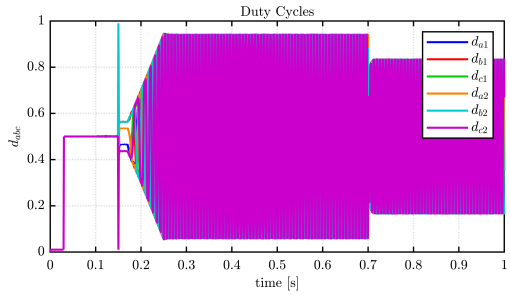

Figure 3.33: Six phase motor - Duty Cycles at 100 rpm, Nominal Torque

The duty cycles, which are not saturated, illustrate the motor's correct operation for this workload, avoiding overloading conditions.

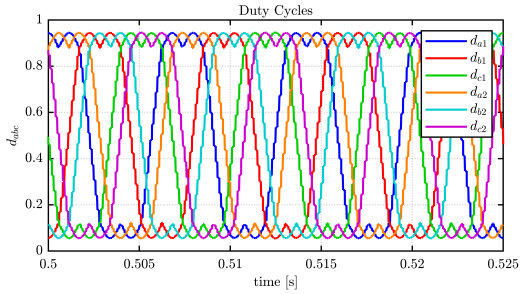

Figure 3.34: Zoom on Duty Cycles

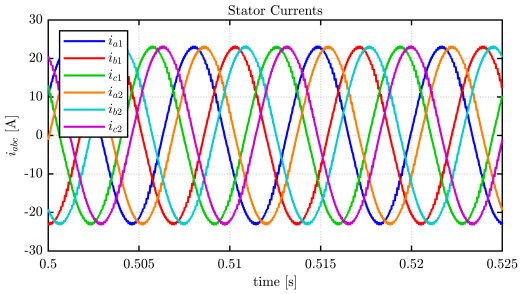

Figure 3.35: Zoom on Currents

These detailed views offer deeper insight into the motor's performance, underlining the effectiveness of current control and modulation strategies.

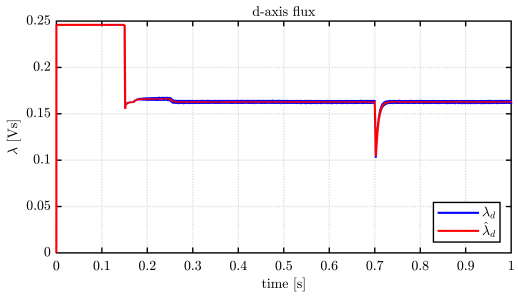

Figure 3.36: Six phase motor - λ_d

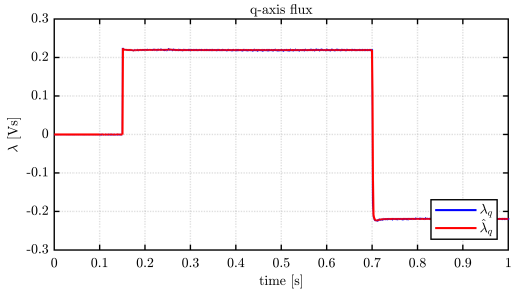

Figure 3.37: Six phase motor - λ_q

Flux linkages in d and q axes are precisely computed, validating the control algorithm's effectiveness.

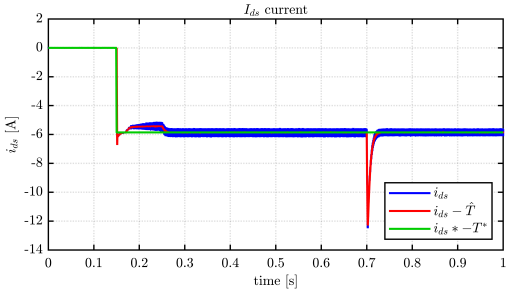

Figure 3.38: Six phase motor - I_d

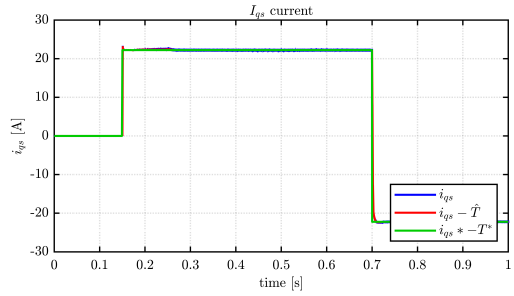

Figure 3.39: Six phase motor - I_q

The analysis of direct and quadrature currents further confirms the motor's capacity to meet control targets, showcasing its proficiency in torque generation and overall performance efficiency.

PM_2x3ph Waveforms

The THOR_3x3ph, a nine-phase electric motor, presents a unique configuration aimed at enhancing the motor's robustness and operational flexibility.
The subsequent figures display the motor's responses to a nominal torque request, showcasing its capability to follow the torque reference accurately, generate sinusoidal current waveforms, maintain optimal duty cycle operations, and accurately compute flux linkages as well as direct/quadrature currents.

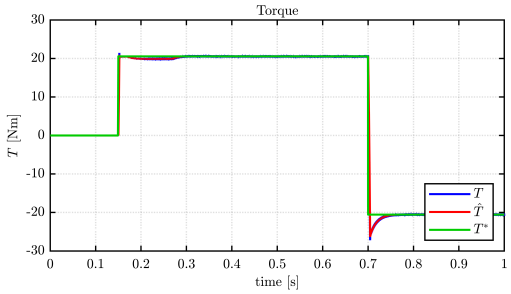

Figure 3.40: Nine phase motor Torque graph

The torque graph validates the motor's precision in tracking the torque reference across the operational range.

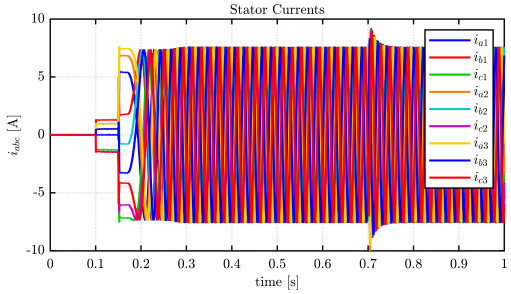

Figure 3.41: Nine phase motor - Currents at 1500 rpm, Nominal Torque

Currents are sinusoidal and at nominal value, reflecting the motor's compliance with the torque demand.

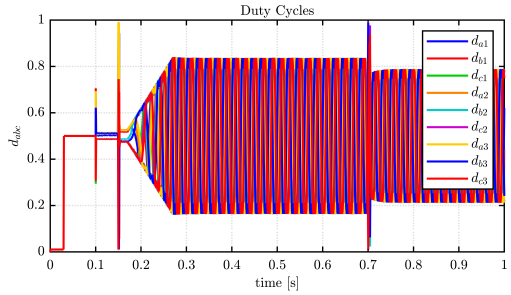

Figure 3.42: Nine phase motor - Duty Cycles at 1500 rpm, Nominal Torque

Duty cycles remain within optimal ranges, ensuring the motor operates within safe operational limits.

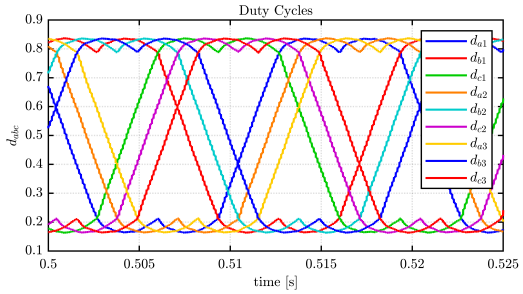

Figure 3.43: Zoom on Duty Cycles

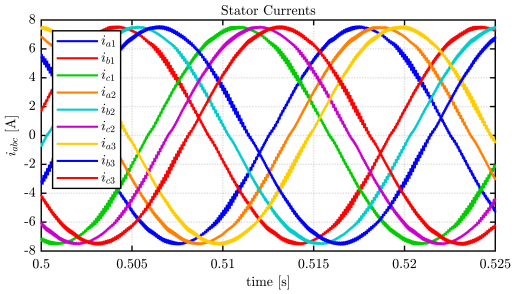

Figure 3.44: Zoom on Currents

These zoomed views further detail the motor's performance, highlighting the efficiency of its current modulation and the fidelity of its control mechanisms.

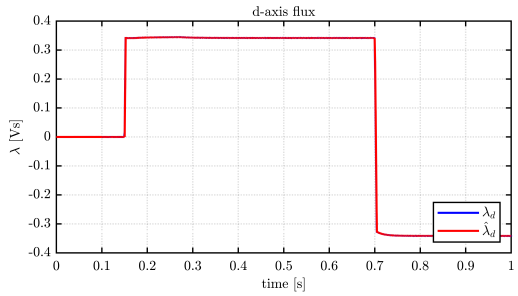

Figure 3.45: Nine phase motor - λ_d

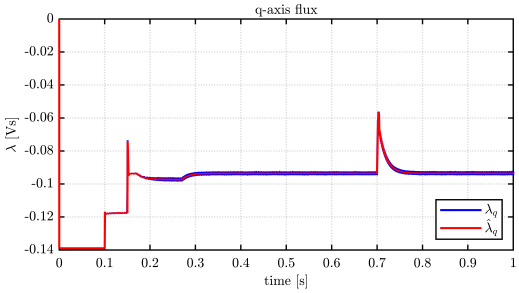

Figure 3.46: Nine phase motor - λ_q

Flux linkages in d and q axes are computed accurately, showcasing the control strategy's success in maintaining optimal magnetic conditions.

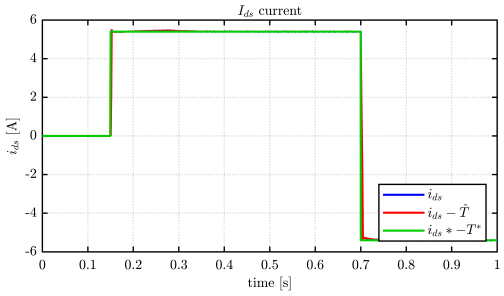

Figure 3.47: Nine phase motor - I_d

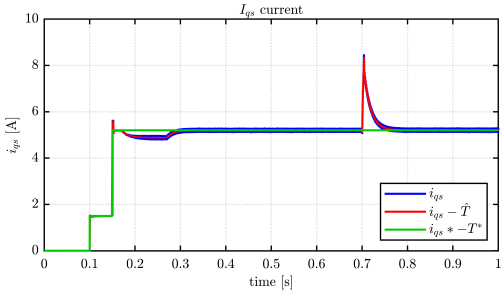

Figure 3.48: Nine phase motor - I_q

The analysis of direct and quadrature currents further confirms the motor's proficiency in meeting its performance targets, thereby affirming the effectiveness of the simulation and control strategy.

PM_4x3ph Waveforms

The PMSM_4x3ph represents an advanced twelve-phase electric motor designed for high torque and precise control.
The series of figures that follow illustrate the motor's response to the applied nominal torque, showing its capability to adhere closely to the torque reference, produce sinusoidal current waveforms at the specified nominal values, and ensure duty cycles are optimally managed for the given operational point.

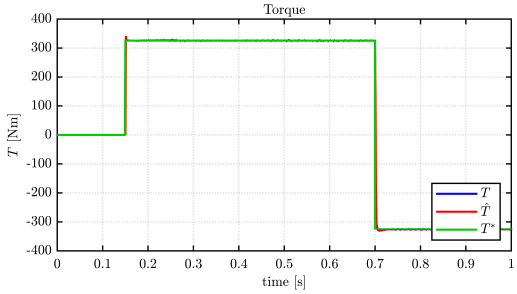

Figure 3.49: Twelve phase motor Torque graph

This torque graph validates the twelve-phase motor's precision in following the set torque reference, showcasing its robust control strategy.

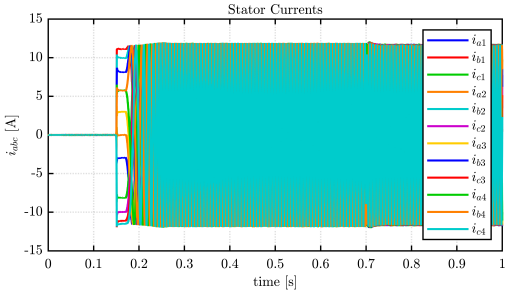

Figure 3.50: Twelve phase motor - Currents at 100 rpm, Nominal Torque

The sinusoidal currents at nominal value confirm the motor's performance is in line with the expected torque demand, illustrating efficient energy conversion.

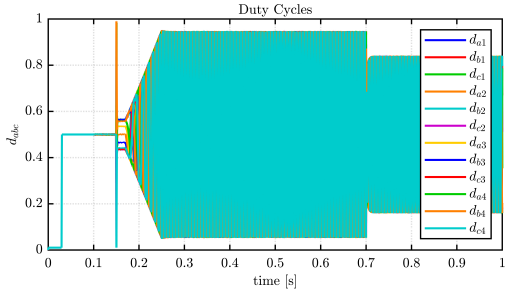

Figure 3.51: Twelve phase motor - Duty Cycles at 100 rpm, Nominal Torque

Optimal duty cycles ensure the motor operates within safe parameters, highlighting the effectiveness of the control system in maintaining stability.

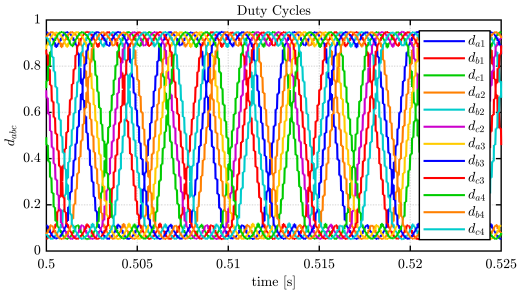

Figure 3.52: Zoom on Duty Cycles

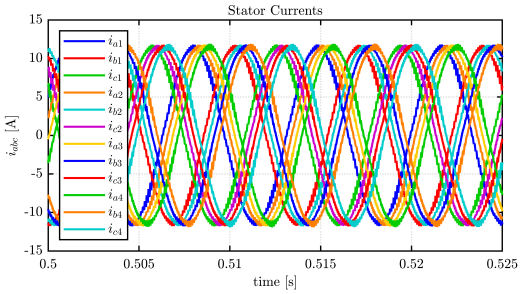

Figure 3.53: Zoom on Currents

These zoomed views provide a closer look at the motor's current modulation and duty cycle precision, further evidencing the motor's reliability and the control strategy's efficacy.

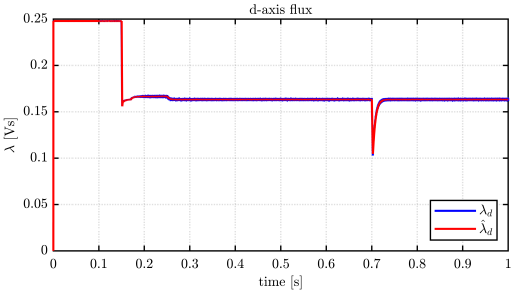

Figure 3.54: Twelve phase motor - λ_d

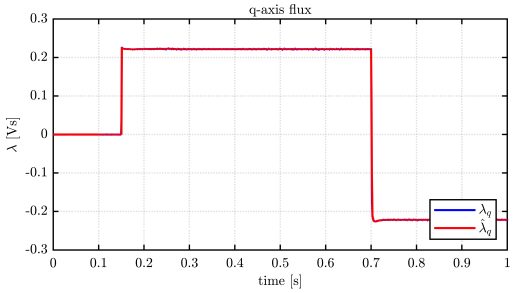

Figure 3.55: Twelve phase motor - λ_q

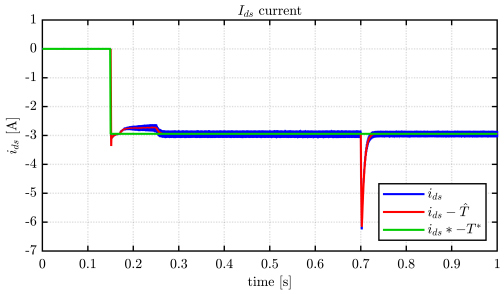

Figure 3.56: Twelve phase motor - I_d

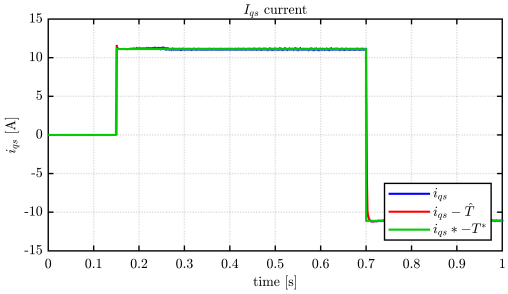

Figure 3.57: Twelve phase motor - I_q

THOR_1x3ph HIL Waveforms

The simulation of the THOR three-phase motor in a Hardware in the Loop (HIL) environment, unlike offline simulations, presents steady-state waveforms at the nominal torque of $19Nm$ and at a speed of $1500rpm$.
It can be observed that the real-time control developed with the NUCLEOboard aligns with the behavior obtained in the offline simulation environment presented in 3.6. Indeed, since the torque references and imposed speeds are the same, it is expected that the observed quantities would also align across the two environments.
Indeed, as can be seen from the graphs below, this happens for all the major observed quantities. Starting with the delivered torque, in Figure 3.58, the control replicates the reference torque well, on which a slightly higher ripple component can be observed, naturally caused by uncertainties in measurements and analog/digital conversions that this type of unoptimized system must face. The result can nevertheless be considered largely satisfactory.

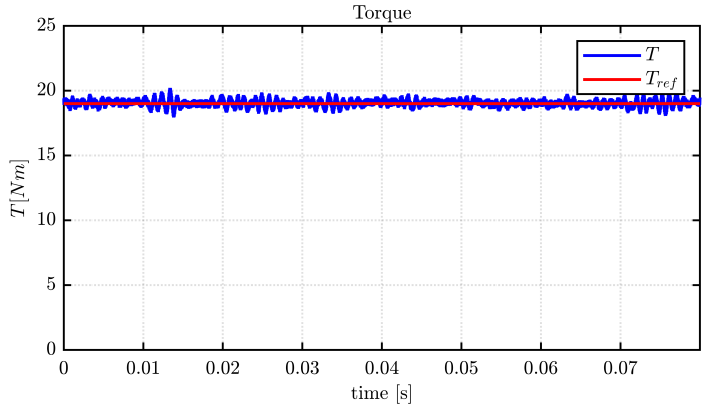

Figure 3.58: Threephase HIL Torque graph

The observations for the torque hold exactly true for the current graph 3.59, in which the peak values are in line with the offline simulation and there is a greater presence of background noise in the measurement.

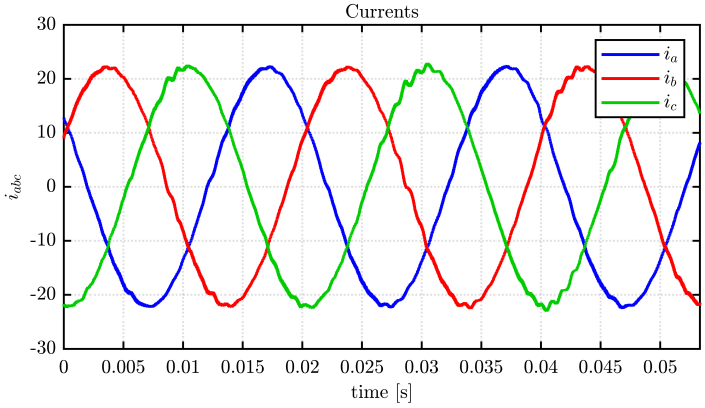

Figure 3.59: Threephase HIL - Currents at 1500 rpm, Nominal Torque

The estimated machine fluxes are also in line with the values obtained through offline simulations, with a d-axis flux of about $0.3[Vs]$ and a q-axis flux equal to $-0.1[Vs]$.

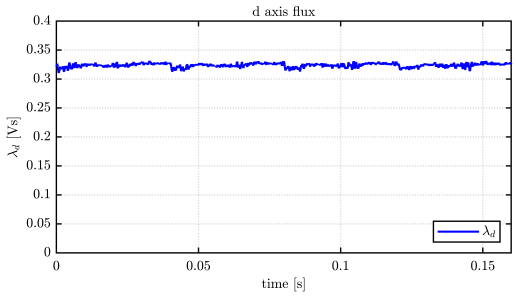

Figure 3.60: Threephase HIL - λ_d

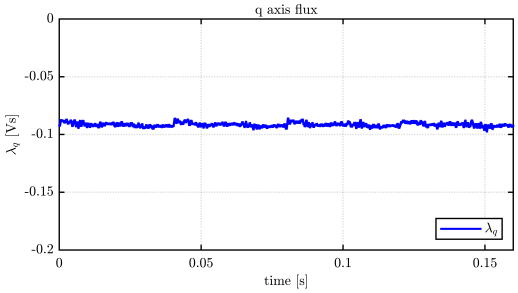

Figure 3.61: Threephase HIL - λ_q

Finally, the stator currents in dq axes also follow the assigned references, with a i_d current of about $16.5[A]$ and a i_q current of about $15.5[A]$.

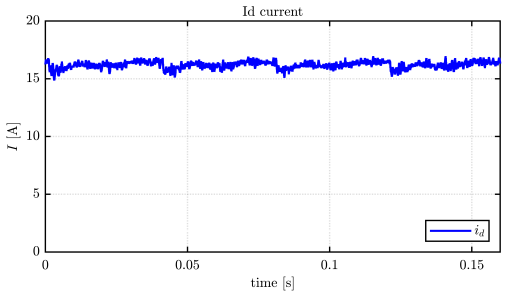

Figure 3.62: Threephase HIL - I_d

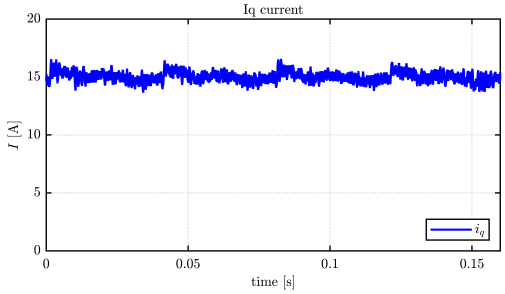

Figure 3.63: Threephase HIL - I_q

Chapter 4

Fault Modeling and Simulation

This chapter delves into the simulation of various fault scenarios in multi-three-phase electrical drives, aiming to assess their resilience under different fault conditions. Through these simulations, the effectiveness of designed fault mitigation strategies is highlighted, providing a robust framework for understanding and managing faults in electrical drives.

With the exception of the ITSC fault, for all faults, the implementation is done automatically when the models are generated in SyreDrive, thanks to the support of the Matlab `print_PLECS_fault_script`, given in Appendix G.
This automatically generates the correct switching signals for each simulated fault case based on knowledge of the number of three-phase sets present in the motor under study.
The integration of this file with a '.c' extension takes place within the C-script shown in figure 3.16.

This code is designed for the automatic generation of fault combinations in multi-three-phase motors within a PLECS model, using a C-script block where the '.c' generated files are reposed. The procedure initiates with the setup of output signals to match input signals under normal conditions. Then, it proceeds to define various fault conditions, such as active short circuits (ASC) and open phases for each three-phase set, by systematically altering the output signals to simulate these faults. For each fault type, the script ensures that only the affected set is modified while keeping the others in their normal state. Additionally, it includes cases for open and shorted legs within the A phase, as well as faults in upper and lower switches the A leg of the fist three phase system, by manipulating the gate signals accordingly. The choice to use just one leg of the first three phase system is made to reduce the cases to a reasonable number, and also because the choice of the faulted leg or switch is arbitrary. Finally, the script concludes by assigning the modified output signals back to the PLECS model and writing the entire script to a .c file, facilitating integration into the simulation environment for fault analysis.

4.1 Three-phase Short Circuit / Active Short Circuit

Three Phase Short Circuit, is a phenomenon that occurs when the terminals of a three phase electrical system in a circuit that normally have different potentials are connected to each other across a very low or zero resistance, causing a very high current flow. This event can occur for various reasons, such as material failure, maintenance error, mechanical damage or overvoltage, and can lead to serious consequences for the electrical and electronic components involved, including physical damage, loss of functionality, or even fire.

In the context of electric motors and drive systems, Three Phase Short Circuit can refer to specific failure scenarios or test methodologies aimed at assessing the resilience of such systems. The rapid identification of this fault is crucial to minimise damage to the system. Modern drive systems are often equipped with advanced fault detection and protection features, capable of quickly shutting down the power supply in the event of an Three Phase Short Circuit. These include overcurrent detection, analysis of current and voltage signatures, and sophisticated software algorithms that differentiate faults from normal transients.

The consequences of an Three Phase Short Circuit vary depending on the severity of the short circuit, the type of system and its ability to withstand overcurrents. In electric motors, an Three Phase Short Circuit can lead to excessive overheating, insulation damage, component deformation and, in the most extreme cases, destruction of the motor. To assess the robustness of electric drive systems against short circuits, specific tests are often conducted. These tests help to identify weaknesses in the design and organisation of the system, allowing engineers to make changes to improve fault resistance.

To mitigate the impact of Three Phase Short Circuits in drive systems, several strategies have been developed. These include fuse-based protection systems, circuit breakers, current limiters and intelligent control systems that can quickly disconnect power or modify system operation to reduce short-circuit current. Simulation and modelling play an important role in understanding how drive systems react to Three Phase Short Circuits. Using advanced simulation software, engineers can predict the impact of various short-circuit scenarios and develop effective mitigation strategies, thus avoiding exposing real systems to potential damage. This approach not only contributes to improving the safety and reliability of electrical drive systems, but also helps optimise design and fault management during development, ensuring efficient and safe operation in the long term.

In recent years, some applications intentionally use three phase short circuit events (under the name of Active Short Circuit) to safely stop electric motors during operation, since in some conditions, other braking techniques are more critical to manage. For example, in very high speed operating points with flux-weakening operation, the inverter shut down would cause Uncontrolled Generator Operation. To implement ASC, this systems switch-on all the upper (or lower) devices of the 2-L inverter, turning off the counterpart.

In summary, Active Short Circuit is a critical event that requires attention in the design, testing and operation of electrical and electronic systems. Strategies to identify, prevent and mitigate ASCs are essential to ensure the safety, reliability and longevity of electrical devices and systems.

In this section, we explores this occurrence intentionally in safe simulation environment for all the Motors Under test presented.

THOR_1x3ph Waveforms

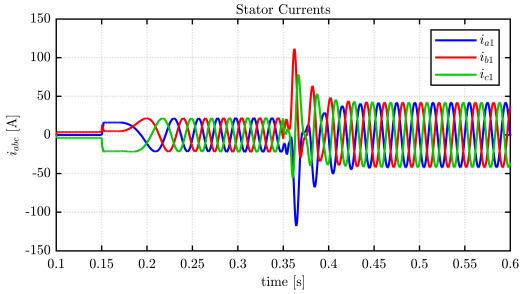

Figure 4.1: Threephase ASC - I_{abc}

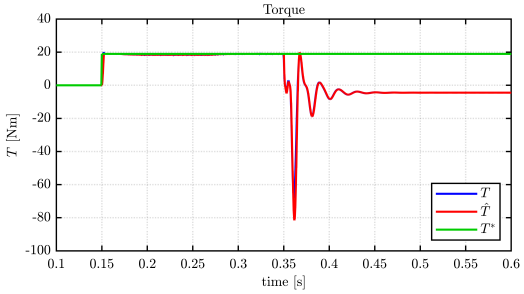

Figure 4.2: Threephase ASC - T_m

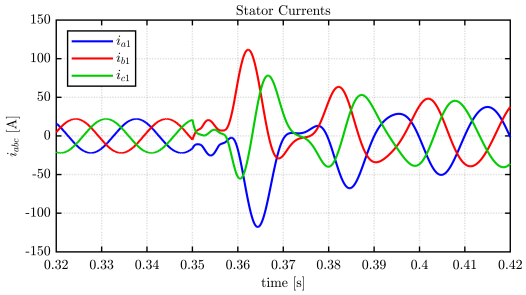

Figure 4.3: Threephase ASC - Zoom I_{abc}

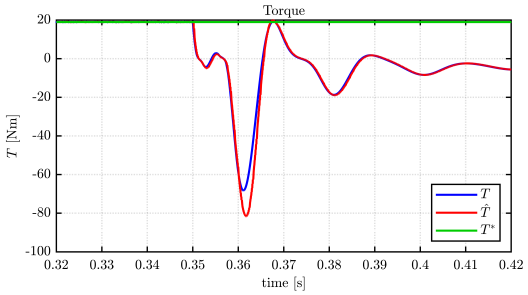

Figure 4.4: Threephase ASC - Zoom T_m

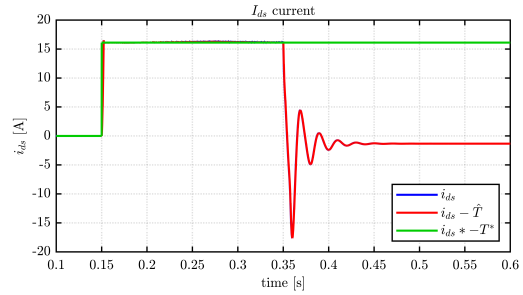

Figure 4.5: Threephase ASC - I_d

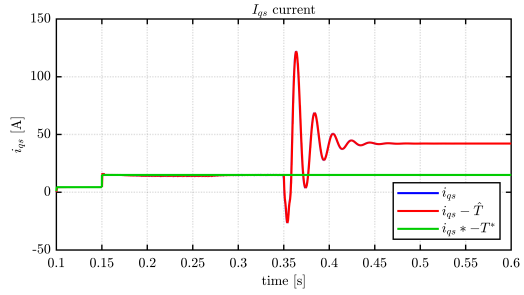

Figure 4.6: Threephase ASC - I_q

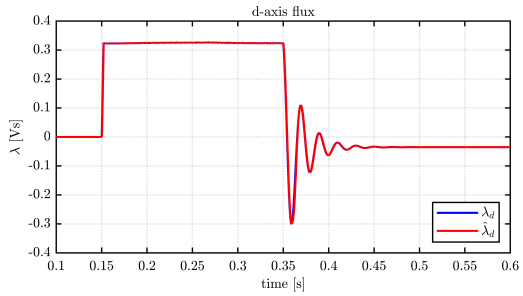

Figure 4.7: Threephase ASC - λ_d

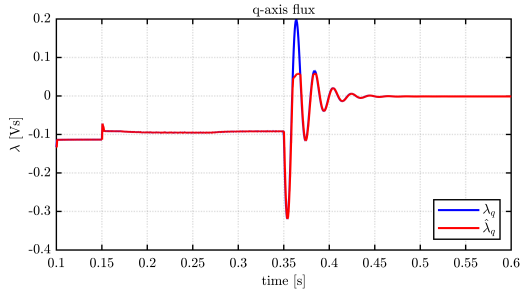

Figure 4.8: Threephase ASC - λ_q

THOR_3x3ph Waveforms

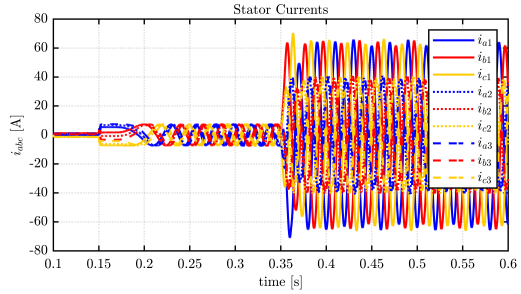

Figure 4.9: Nine-Phase ASC - I_{abc}

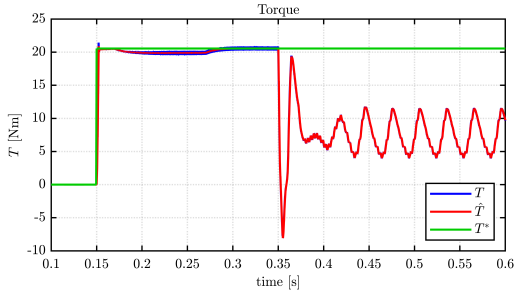

Figure 4.10: Nine-Phase ASC - T_m

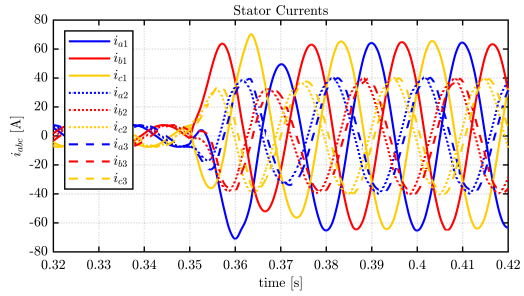

Figure 4.11: Nine-Phase ASC - Zoom I_{abc}

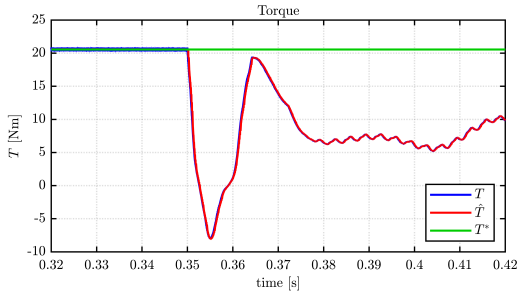

Figure 4.12: Nine-Phase ASC - Zoom T_m

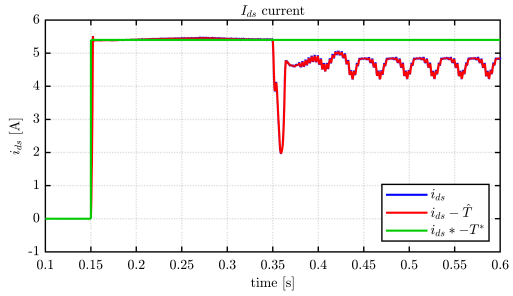

Figure 4.13: Nine-Phase ASC - I_d

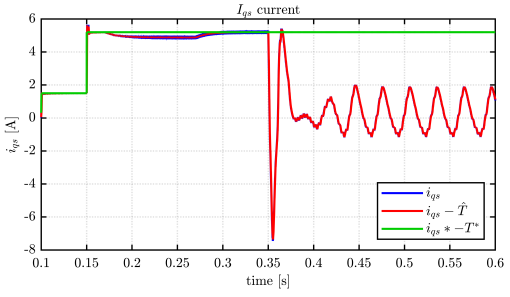

Figure 4.14: Nine-Phase ASC - I_q

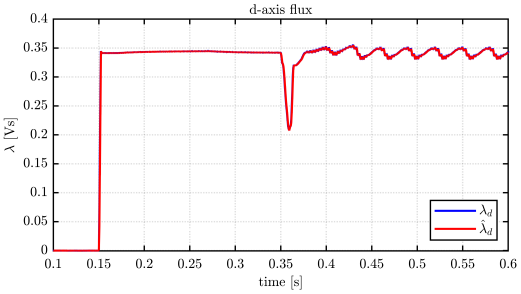

Figure 4.15: Nine-Phase ASC - λ_d

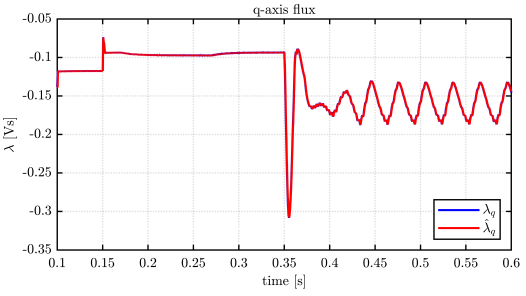

Figure 4.16: Nine-Phase ASC - λ_q

PMSM_2x3ph Waveforms

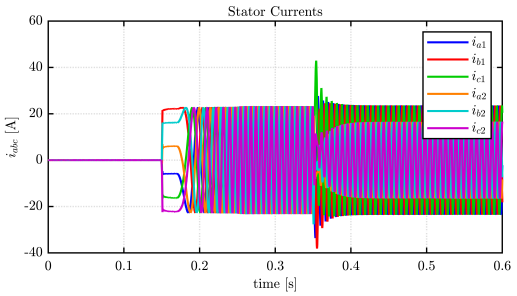

Figure 4.17: Six-Phases ASC - I_{abc}

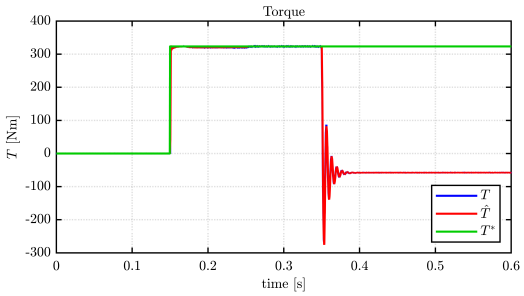

Figure 4.18: Six-Phases ASC - T_m

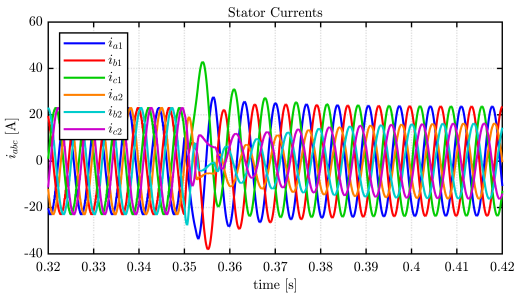

Figure 4.19: Six-Phases ASC - Zoom I_{abc}

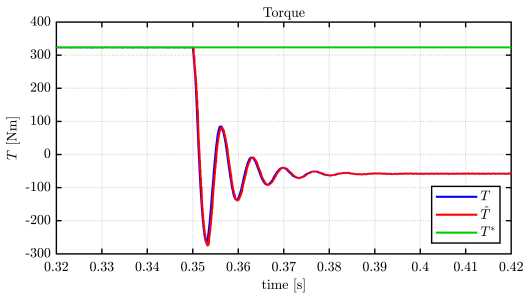

Figure 4.20: Six-Phases ASC - Zoom T_m

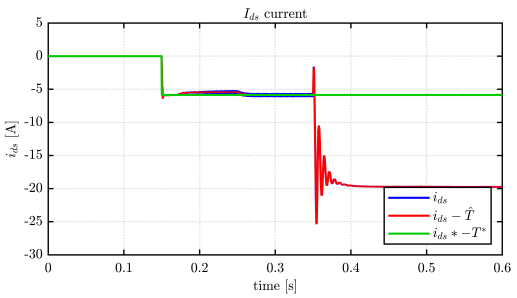

Figure 4.21: Six-Phases ASC - I_d

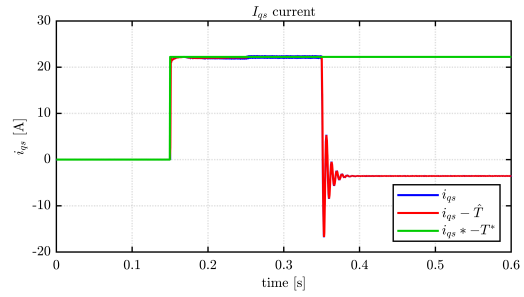

Figure 4.22: Six-Phases ASC - I_q

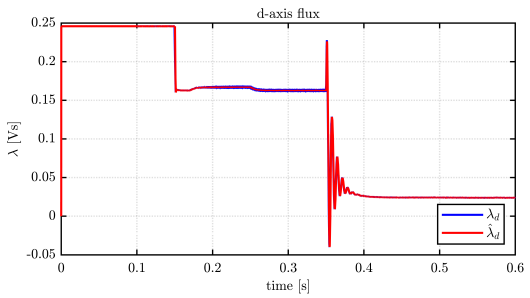

Figure 4.23: Six-Phases ASC - λ_d

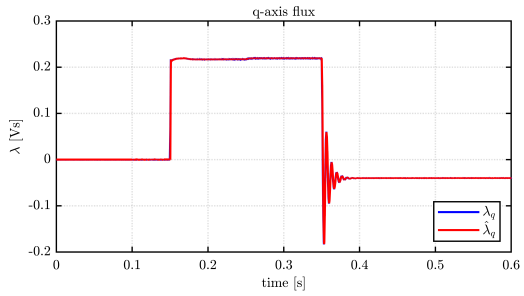

Figure 4.24: Six-Phases ASC - λ_q

PMSM_4x3ph Waveforms

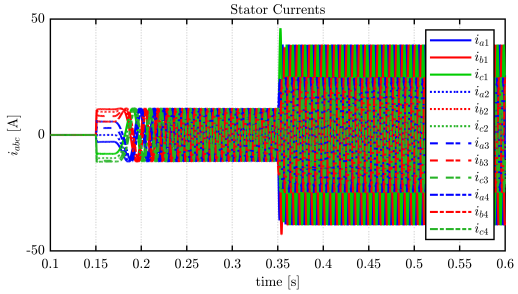

Figure 4.25: Twelve-Phases ASC - I_{abc}

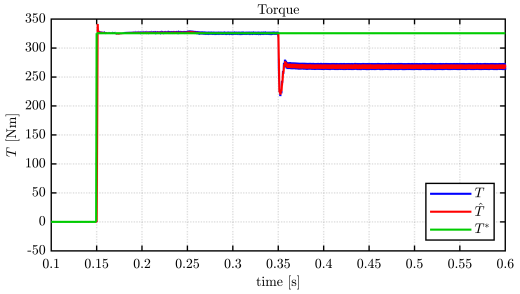

Figure 4.26: Twelve-Phases ASC - T_m

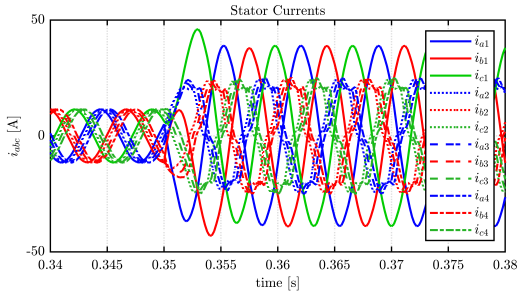

Figure 4.27: Twelve-Phases ASC - Zoom I_{abc}

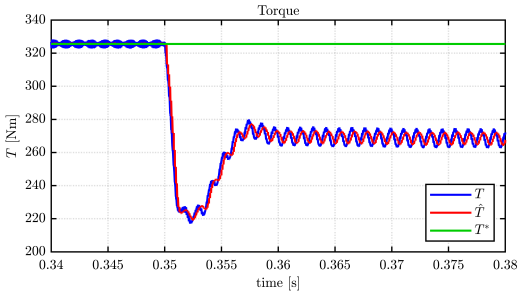

Figure 4.28: Twelve-Phases ASC - Zoom T_m

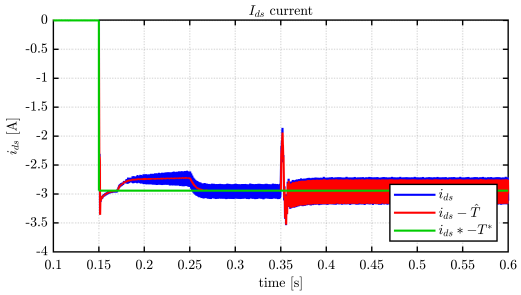

Figure 4.29: Twelve-Phases ASC - I_d

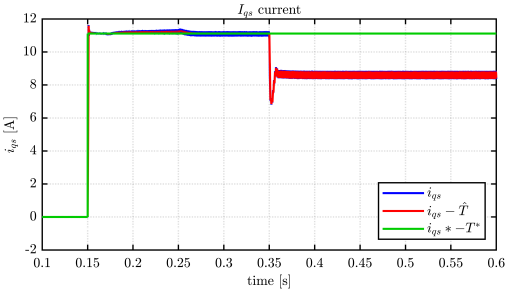

Figure 4.30: Twelve-Phases ASC - I_q

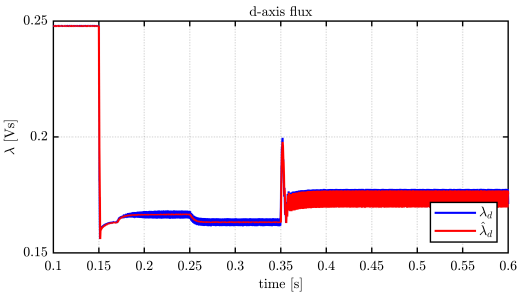

Figure 4.31: Twelve-Phases ASC - λ_d

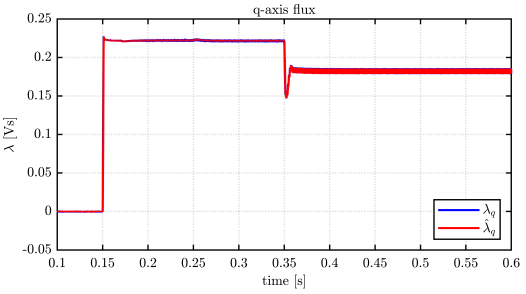

Figure 4.32: Twelve-Phases ASC - λ_q

THOR_1x3ph Real-time Waveforms

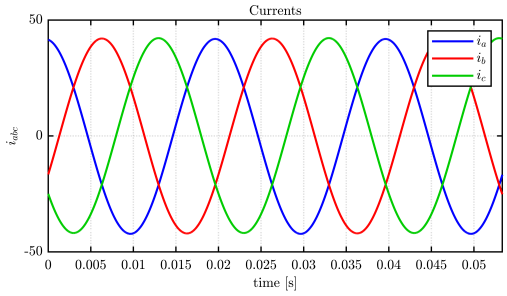

Figure 4.33: Threephase HIL ASC - I_{abc}

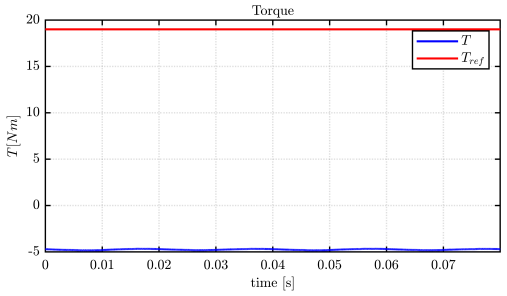

Figure 4.34: Threephase HIL ASC - T_m

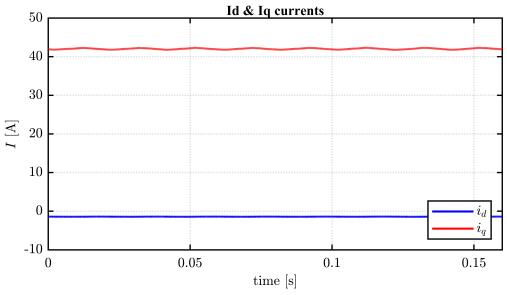

Figure 4.35: Threephase HIL ASC - I_{dq}

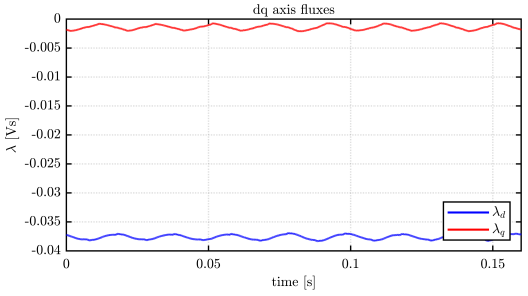

Figure 4.36: Threephase HIL ASC - λ_{dq}

4.1.1 Simulation Results

A substantial difference can be seen in the different versions of the THOR motor: in fact, the active short-circuit behaviour is different for the three-phase motor and the nine-phase.

In particular, we can observe that the three-phase motor transiently has a much greater peak current (equal to about 110[A]) than that found for its nine-phase "brother", which transiently has a peak negative current on phase a of about70[A].

In contrast, under ASC conditions, the three-phase motor delivers a peak current of about 50[A], to which a slight braking torque corresponds.

The nine-phase motor, on the other hand, on the failed backhoe delivers a current under steady state conditions that is higher than 60[A] at the peak; this happens because the phases that have remained healthy are trying to make up for the lack and braking of the first backhoe, managing to deliver a positive average torque of about one third of that required.

Almost similar considerations can be made for the Six-phase PM_2x3ph and Twelve-phase PM_4x3ph motors: the first of these two does not manage to make up for the lack of the first three phases and results in an average negative torque of$50N \cdot m$, with maximum currents on the faulty phases of approximately 25[A].

On the other hand, the PM_4x3ph motor, thanks to its modularity, similarly to what has been seen for the nine-phase motor, succeeds in delivering part of the required torque $270N \cdot m$ on $325N \cdot m$, with an increase of the steady-state currents on the failed phases to 40[A] at the peak, which exceed the maximum current limits for this motor 35[A].

All of these simulations were conducted with the motors close to the nominal operating conditions ($T_m^* = 19N \cdot m$,$n = 1500rpm$ for three-phase and nine-phase motors, $T_m^* = 320N \cdot m$,$n = 400rpm$ for hexaphase and twelve-phase motors). It is observed that in the case where the multiphase motors THOR_3x3ph and PM_4x12ph the ASC is carried out at a reduced rotational speed (about one third of that used in the simulations presented), they are able to restore the full required torque by the control, demonstrating a particular resilience to failure.

With regard to the HIL model, it can be seen that the current, torque and machine flow waveforms closely mirror those obtained through offline simulation, validating the control developed.

4.2 Inverter Shut Down

Inverter shutdown, known as inverter shutdown, is a critical mechanism in the operation of these devices, which play a key role in the conversion of direct current (DC) to alternating current (AC) in a wide variety of applications, from photovoltaic systems to electric vehicle traction systems to complex industrial drive systems. This switch-off process can take place either as part of normal operation, e.g. for maintenance or scheduled shutdown, or as a protective measure in response to specific faults or hazardous conditions, ensuring safety and reliability for the entire system.

The main protective functions that can trigger an inverter shutdown include overload, thermal, short-circuit and critical supply voltage variation protection. These safety mechanisms are designed to prevent damage to the inverter's internal components and the connected load by promptly intervening if potentially damaging conditions are detected, such as an excessive current that could indicate an overload or short circuit, or a dangerous rise in internal temperature.

Modern inverters are equipped with sophisticated monitoring and control systems that, through advanced sensors and algorithms, are able to quickly detect these abnormal conditions. This detection capability allows timely implementation of the shutdown function to mitigate risks and prevent potential damage. Some systems require manual intervention to restart after a shutdown, while others can be programmed to attempt an automatic restart once the abnormal conditions are resolved or after a certain time interval, always respecting safety criteria.

This precautionary approach not only prevents physical damage and reduces the risk of fire, but also ensures that connected equipment is protected from harmful current and voltage fluctuations. In this way, controlled inverter shutdown contributes significantly to the reliability and durability of electronic systems, emphasising the importance of such protection mechanisms in maintaining operational integrity and safety in a wide range of applications. Through continuous innovation and improved monitoring and control technologies, inverters continue to evolve, becoming safer and safer devices capable of effectively handling the most diverse operating conditions and possible anomalies.

THOR_1x3ph Waveforms

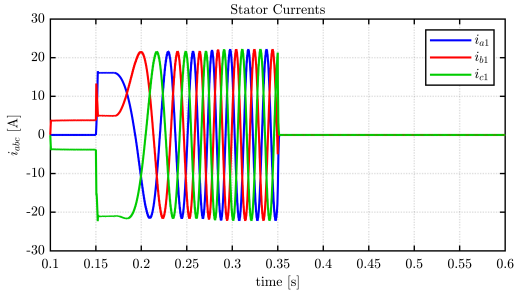

Figure 4.37: Threephase ISD - I_{abc}

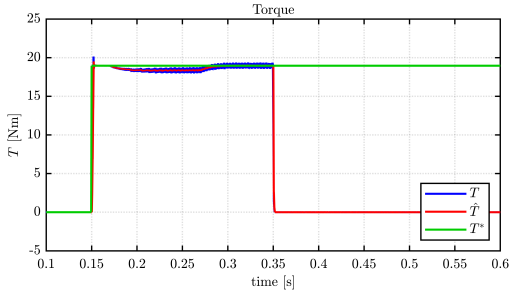

Figure 4.38: Threephase ISD - T_m

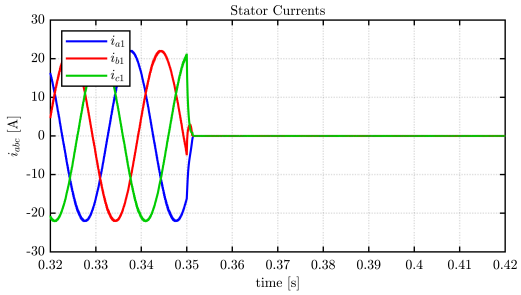

Figure 4.39: Threephase ISD - Zoom I_{abc}

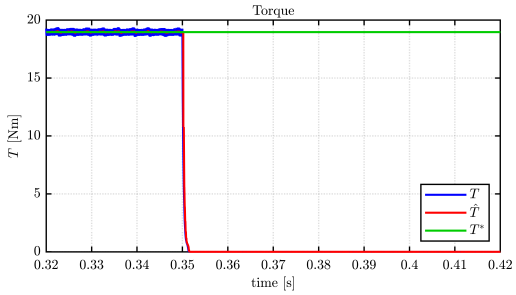

Figure 4.40: Threephase ISD - Zoom T_m

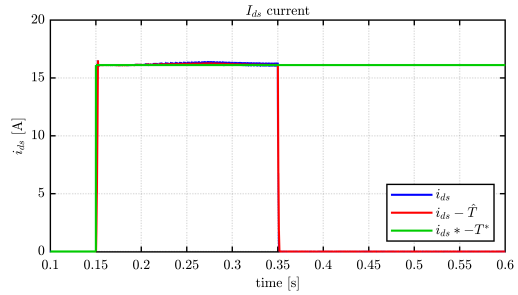

Figure 4.41: Threephase ISD - I_d

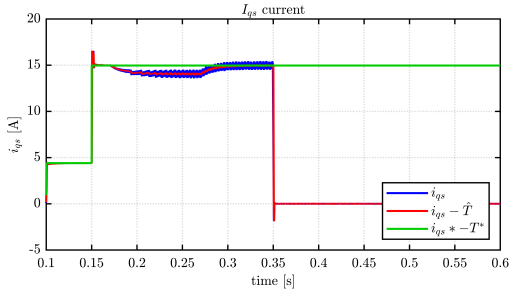

Figure 4.42: Threephase ISD - I_q

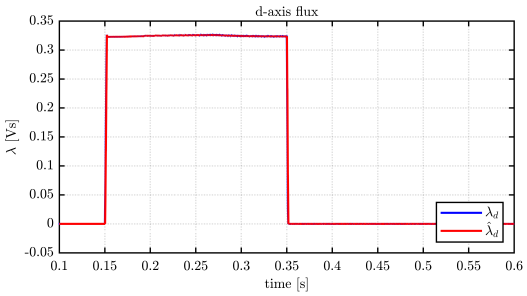

Figure 4.43: Threephase ISD - λ_d

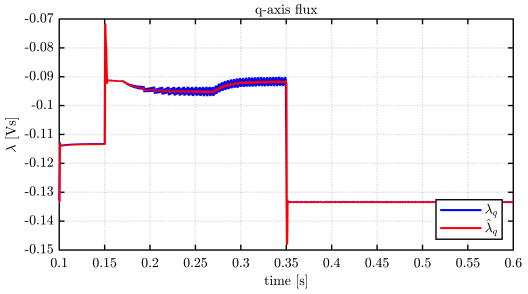

Figure 4.44: Threephase ISD - λ_q

THOR_3x3ph Waveforms

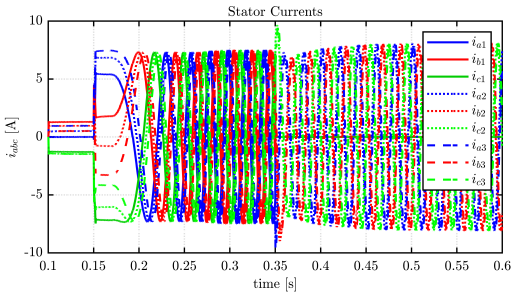

Figure 4.45: Nine-Phase ISD - I_{abc}

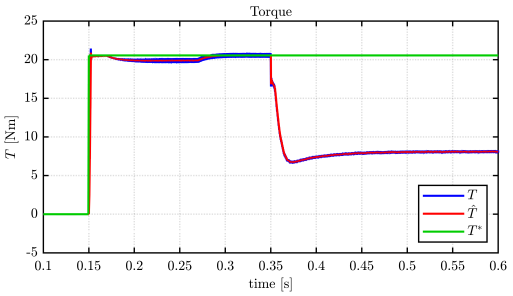

Figure 4.46: Nine-Phase ISD - T_m

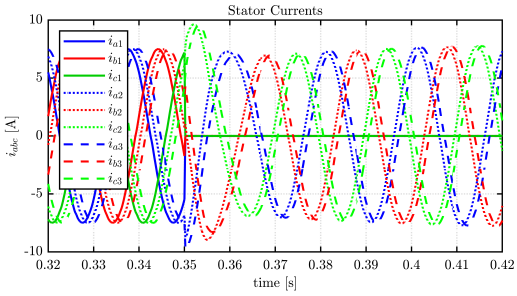

Figure 4.47: Nine-Phase ISD - Zoom I_{abc}

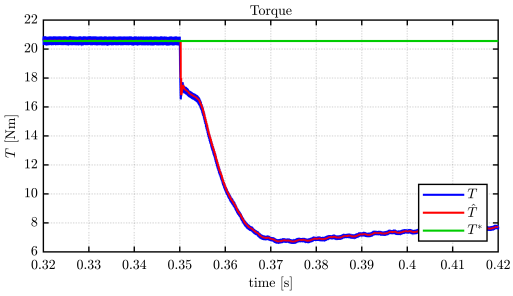

Figure 4.48: Nine-Phase ISD - Zoom T_m

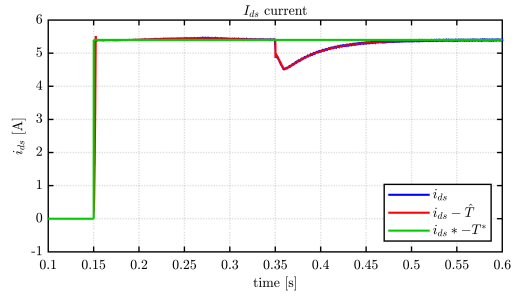

Figure 4.49: Nine-Phase ISD - I_d

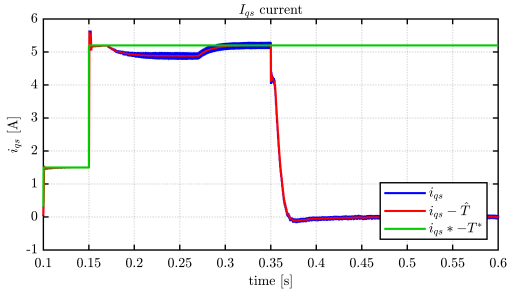

Figure 4.50: Nine-Phase ISD - I_q

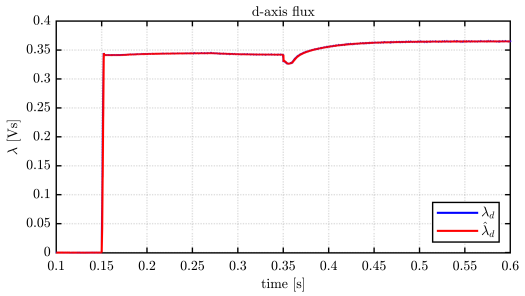

Figure 4.51: Nine-Phase ISD - λ_d

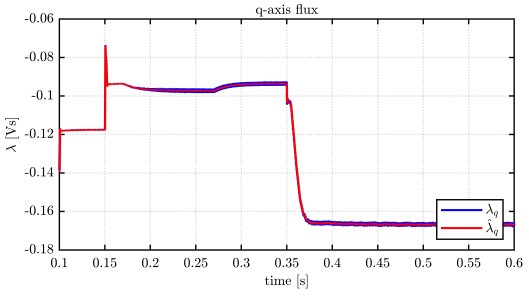

Figure 4.52: Nine-Phase ISD - λ_q

PMSM_2x3ph Waveforms

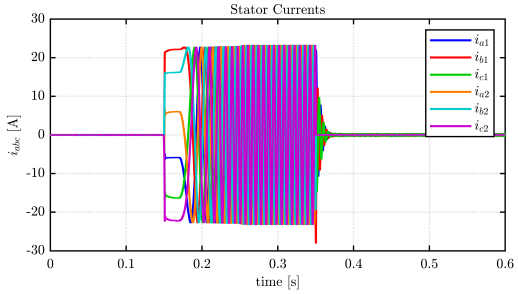

Figure 4.53: Six-Phases ISD - I_{abc}

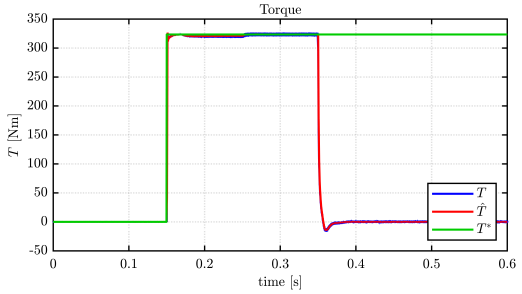

Figure 4.54: Six-Phases ISD - T_m

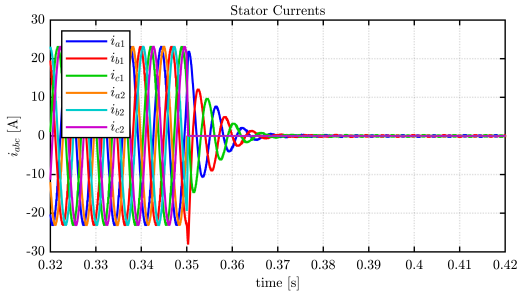

Figure 4.55: Six-Phases ISD - Zoom I_{abc}

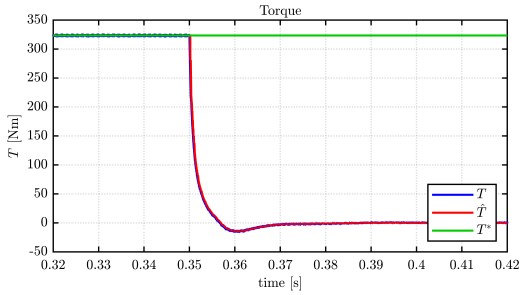

Figure 4.56: Six-Phases ISD - Zoom T_m

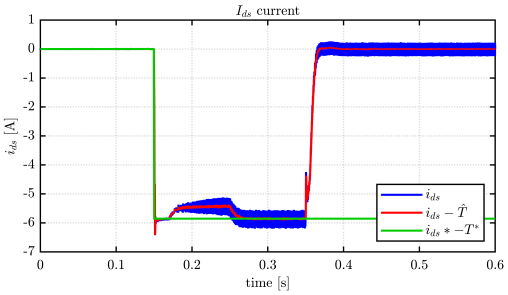

Figure 4.57: Six-Phases ISD - I_d

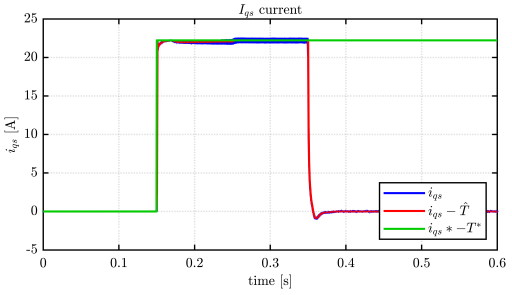

Figure 4.58: Six-Phases ISD - I_q

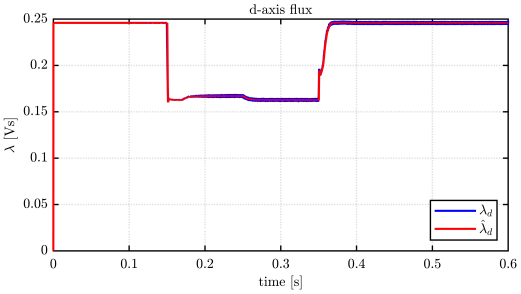

Figure 4.59: Six-Phases ISD - λ_d

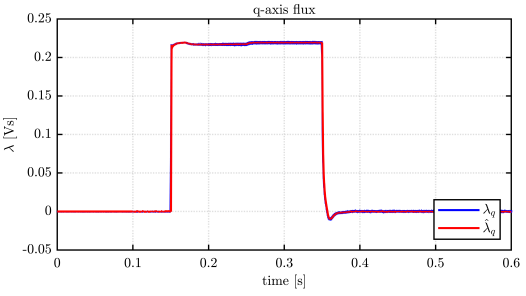

Figure 4.60: Six-Phases ISD - λ_q

PMSM_4x3ph Waveforms

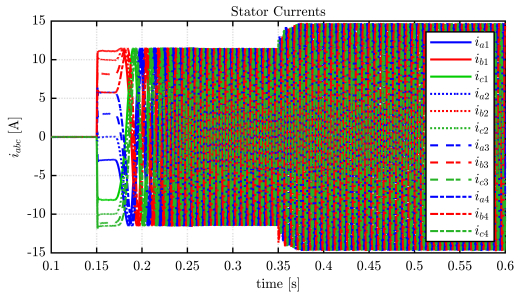

Figure 4.61: Twelve-Phases ISD - I_{abc}

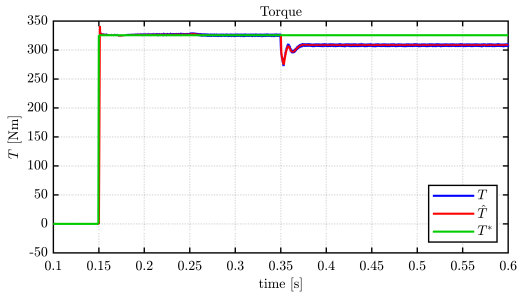

Figure 4.62: Twelve-Phases ISD - T_m

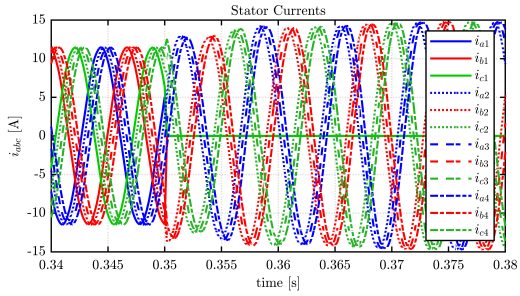

Figure 4.63: Twelve-Phases ISD - Zoom I_{abc}

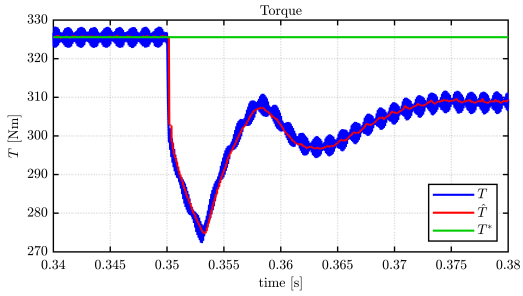

Figure 4.64: Twelve-Phases ISD - Zoom T_m

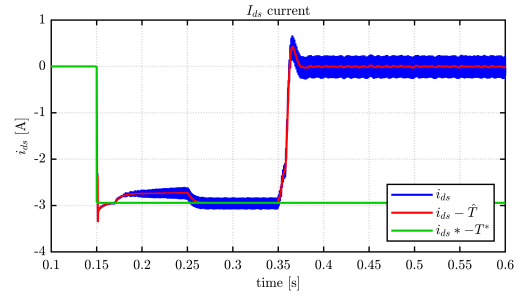

Figure 4.65: Twelve-Phases ISD - I_d

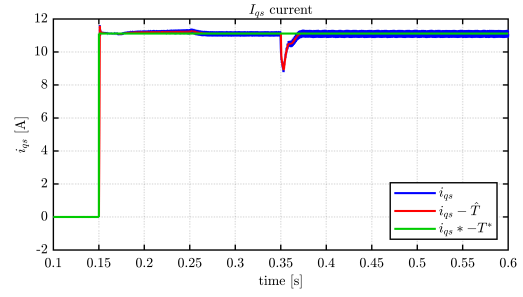

Figure 4.66: Twelve-Phases ISD - I_q

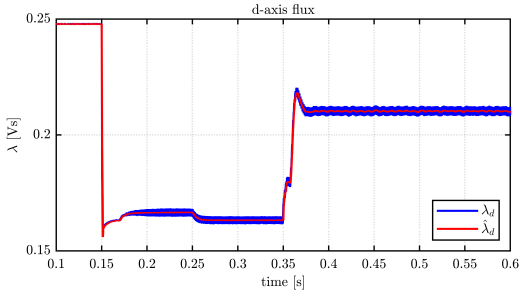

Figure 4.67: Twelve-Phases ISD - λ_d

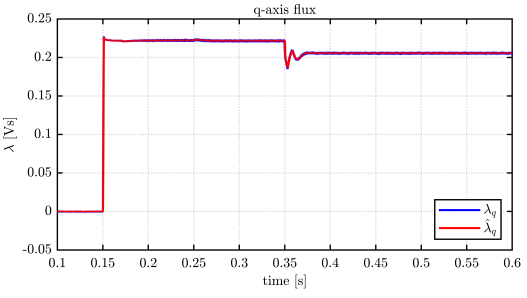

Figure 4.68: Twelve-Phases ISD - λ_q

THOR_1x3ph Real-time Waveforms

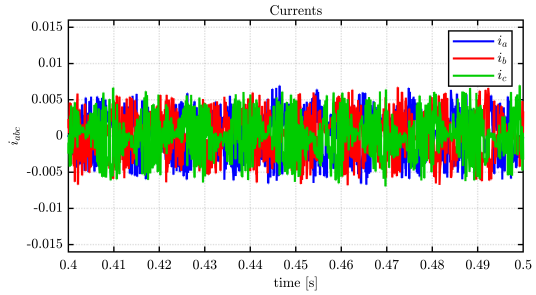

Figure 4.69: Threephase HIL ISD - I_{abc}

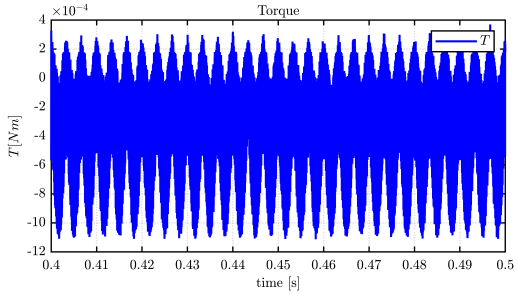

Figure 4.70: Threephase HIL ISD - T_m

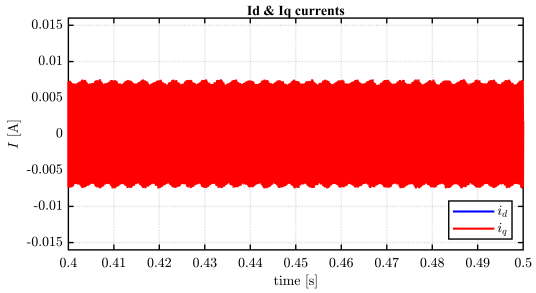

Figure 4.71: Threephase HIL ISD - I_{dq}

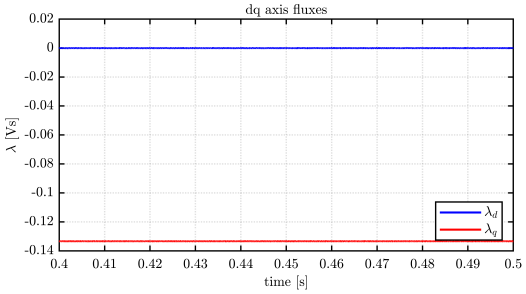

Figure 4.72: Threephase HIL ISD - λ_{dq}

4.2.1 Simulation Results

The inverter shut down, considered as an accidental event, generally causes a significant loss of torque for most of the observed cases. Moreover, compared to the other failures, it is the one that generally has the lowest post-fault torque ripple.

Clearly,for both offline and HIL environments, as far as the three-phase THOR motor is concerned the loss of its only power source results in the cancellation of all quantities typical of electromechanical conversion, such as voltage and current. Furthermore, under these operating conditions, the UGO condition is not observed.

On the other hand, its nine-phase namesake, despite the loss of a third of its power supply, THOR_3x3ph still manages to deliver a positive average torque of just under half of its nominal torque, with a slight overload of its remaining healthy phases.

The six-phase motor, despite the presence of half of its phases still in a healthy state, is unable to make up for the lack of a faulty three phase system and autonomously goes into an inert state, cancelling the current in both directions dq.

On the contrary, the PM_4x3ph motor at the disconnection of a three-phase inverter reacts autonomously in an optimal way and, after a transient drop in motor torque, succeeds in almost completely restoring the motor torque, settling it at a value of $310N \cdot m$.

4.3 Open Phase

When we speak of a 'one-phase open' fault in an electric motor, we are referring to a situation in which one of the three supply phases of the motor is interrupted or loses connection. This type of failure is particularly significant in three-phase motors, which are widely used in industrial and commercial applications due to their efficiency and reliability. A fault of this type can have several causes, such as a damaged cable, a faulty contact, or a problem in the electrical distribution system.

The opening of a phase significantly alters the operation of the motor, causing it to operate under sub-optimal conditions. Normally, a three-phase motor requires a balanced supply from all three phases to function properly. When one phase opens, the motor loses part of its power supply, causing an imbalance that can drastically reduce its efficiency and its ability to generate torque.

Furthermore, this imbalance induces irregular currents in the motor that can cause overheating and vibration, compromising motor life and potentially leading to premature failure.

Recognising and responding quickly to a phase-opening fault is crucial to minimise damage to the motor and related equipment. Early detection of this type of fault can be facilitated by the use of current and voltage monitoring systems that alert operators to abnormal conditions. Once the fault has been identified, it is essential to interrupt the power supply to the motor and investigate the cause of the phase opening, restoring balanced three-phase power before restarting the motor.

To prevent phase-opening faults, it is important to maintain an electrical installation in good condition and to periodically check the integrity of connections and protection systems. The use of specific protections, such as phase protection relays, can help to quickly identify and isolate problems before they can cause significant damage.

In conclusion, a phase-opening fault is a serious problem for three-phase electric motors, negatively affecting their performance and reliability. A thorough understanding of the causes, consequences and strategies for detecting and preventing this type of failure is crucial to ensure the safe and efficient operation of electric drive systems in industrial applications.

THOR_1x3ph Waveforms

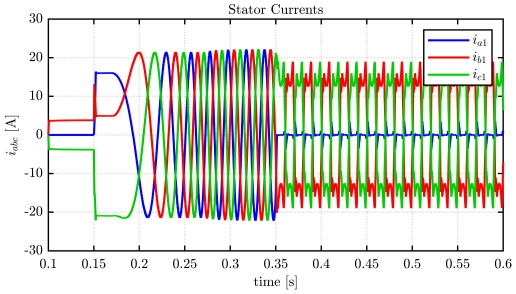

Figure 4.73: Threephase OLF - I_{abc}

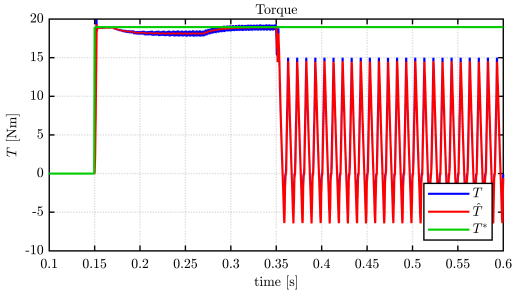

Figure 4.74: Threephase OLF - T_m

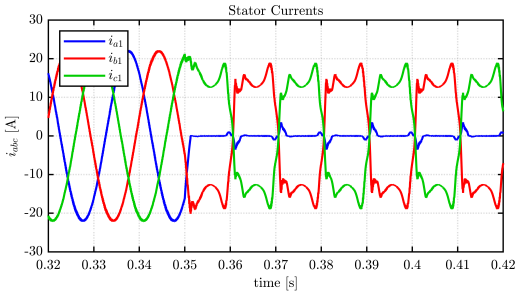

Figure 4.75: Threephase OLF - Zoom I_{abc}

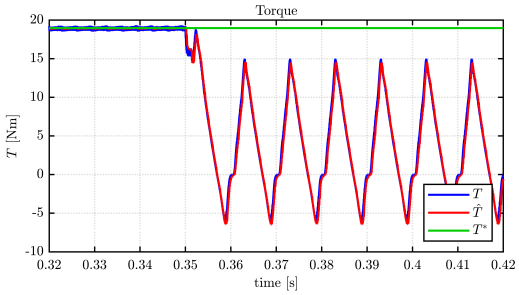

Figure 4.76: Threephase OLF - Zoom T_m

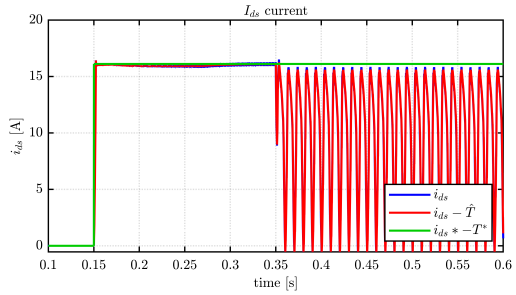

Figure 4.77: Threephase OLF - I_d

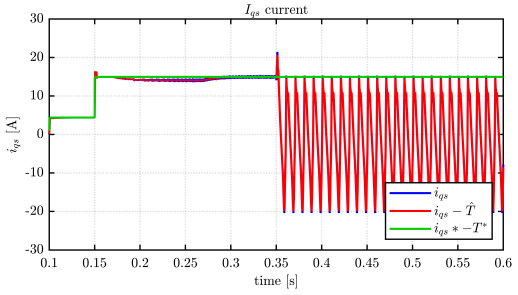

Figure 4.78: Threephase OLF - I_q

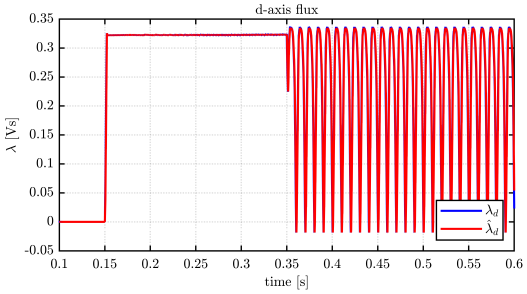

Figure 4.79: Threephase OLF - λ_d

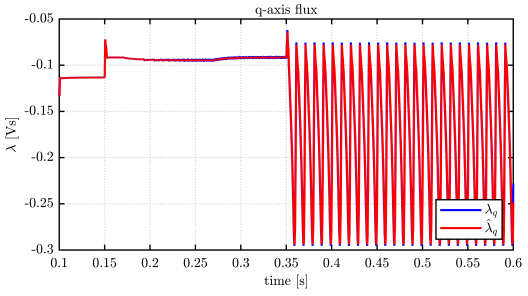

Figure 4.80: Threephase OLF - λ_q

THOR_3x3ph Waveforms

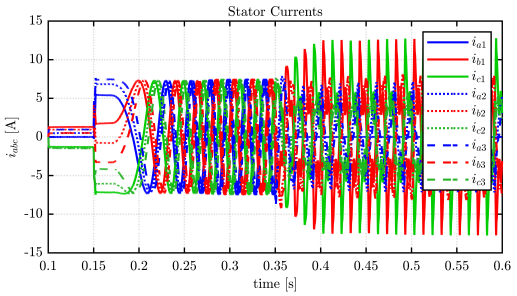

Figure 4.81: Nine-Phase OLF - I_{abc}

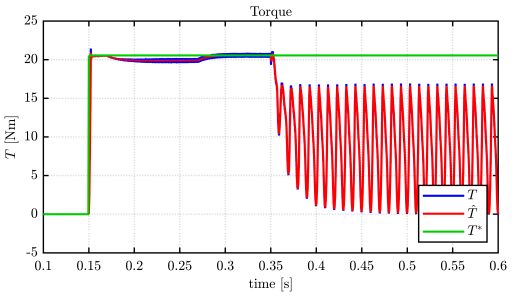

Figure 4.82: Nine-Phase OLF - T_m

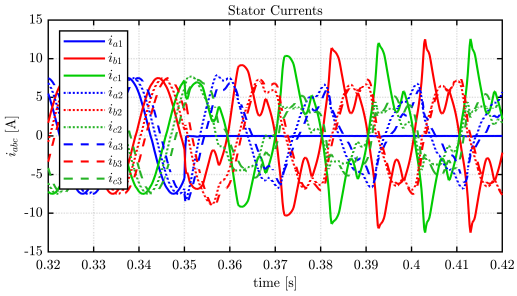

Figure 4.83: Nine-Phase OLF - Zoom I_{abc}

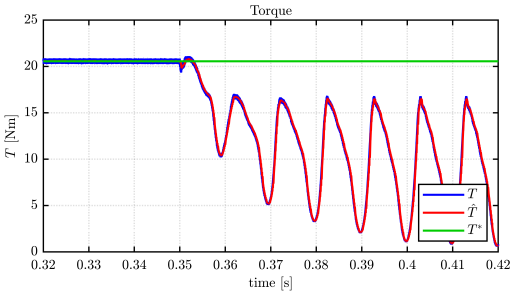

Figure 4.84: Nine-Phase OLF - Zoom T_m

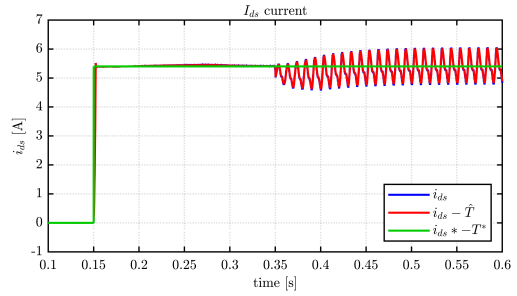

Figure 4.85: Nine-Phase OLF - I_d

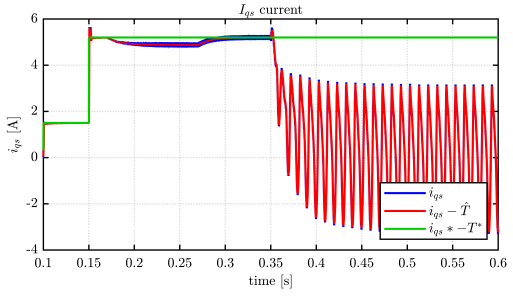

Figure 4.86: Nine-Phase OLF - I_q

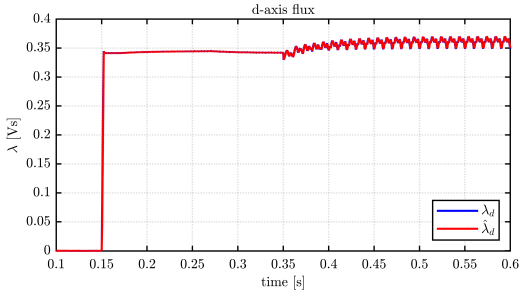

Figure 4.87: Nine-Phase OLF - λ_d

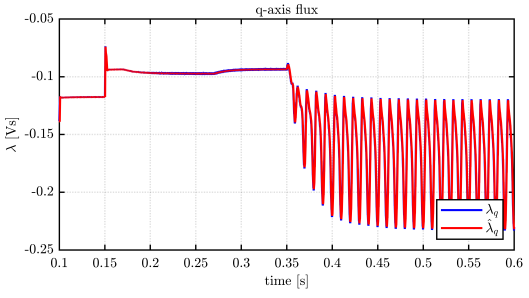

Figure 4.88: Nine-Phase OLF - λ_q

PMSM_2x3ph Waveforms

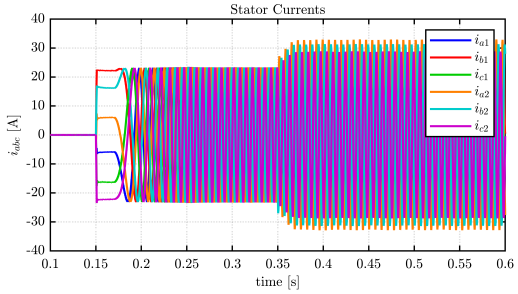

Figure 4.89: Six-Phases OLF - I_{abc}

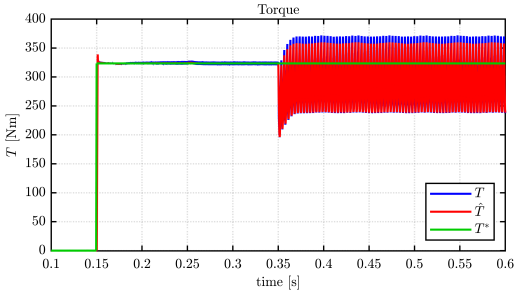

Figure 4.90: Six-Phases OLF - T_m

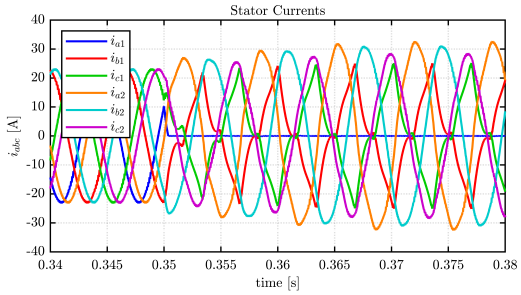

Figure 4.91: Six-Phases OLF - Zoom I_{abc}

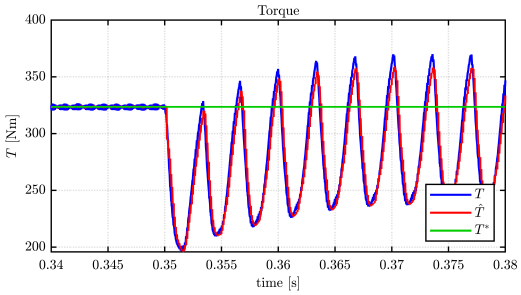

Figure 4.92: Six-Phases OLF - Zoom T_m

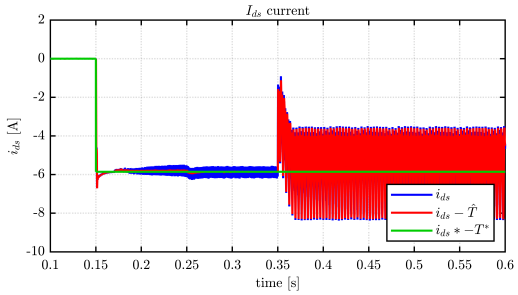

Figure 4.93: Six-Phases OLF - I_d

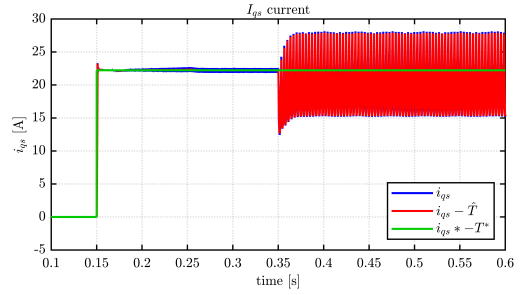

Figure 4.94: Six-Phases OLF - I_q

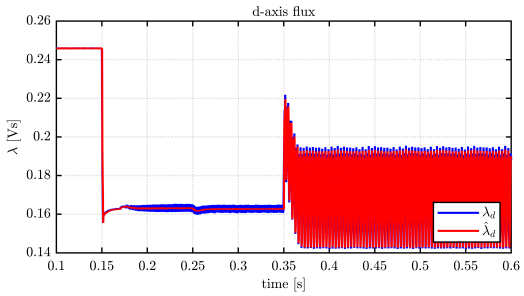

Figure 4.95: Six-Phases OLF - λ_d

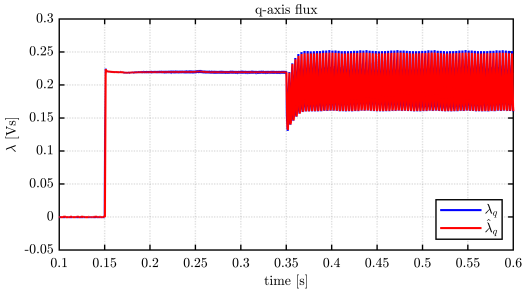

Figure 4.96: Six-Phases OLF - λ_q

PMSM_4x3ph Waveforms

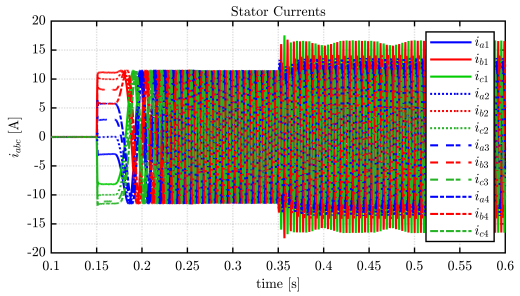

Figure 4.97: Twelve-Phases OLF - I_{abc}

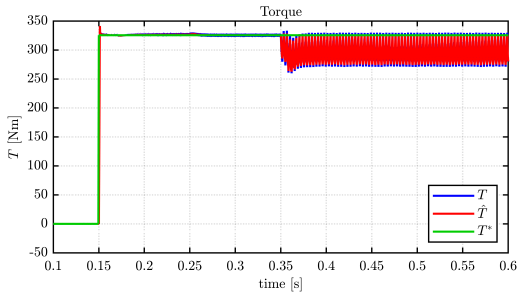

Figure 4.98: Twelve-Phases OLF - T_m

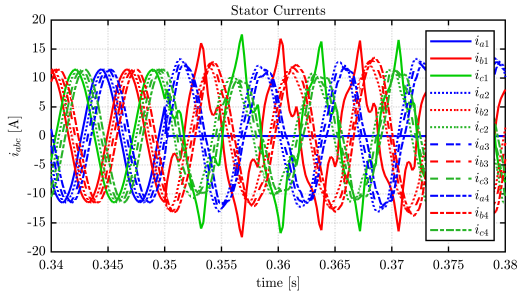

Figure 4.99: Twelve-Phases OLF - Zoom I_{abc}

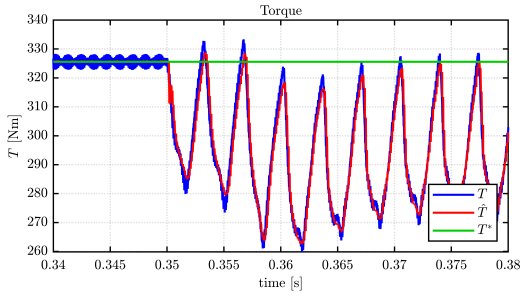

Figure 4.100: Twelve-Phases OLF - Zoom T_m

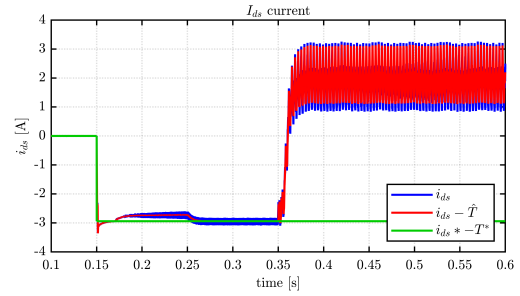

Figure 4.101: Twelve-Phases OLF - I_d

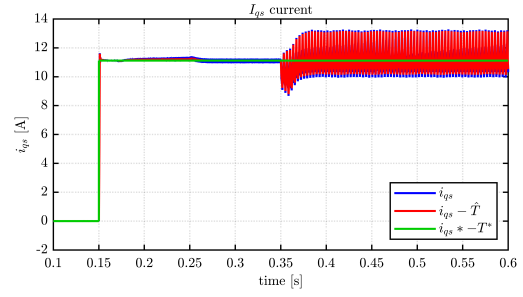

Figure 4.102: Twelve-Phases OLF - I_q

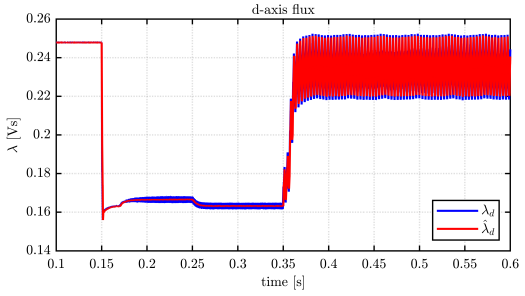

Figure 4.103: Twelve-Phases OLF - λ_d

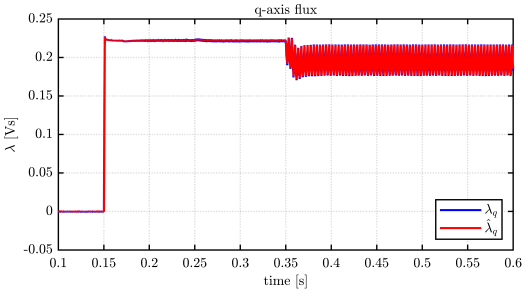

Figure 4.104: Twelve-Phases OLF - λ_q

THOR_1x3ph Real-time Waveforms

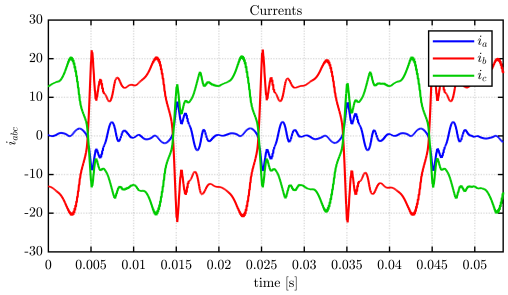

Figure 4.105: Threephase HIL OLF - I_{abc}

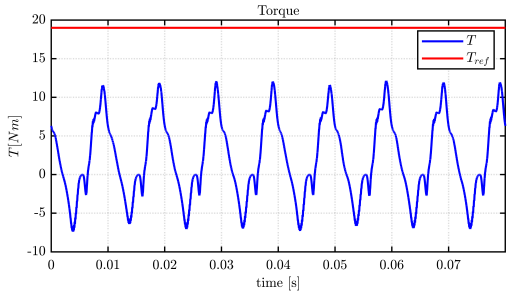

Figure 4.106: Threephase HIL OLF - T_m

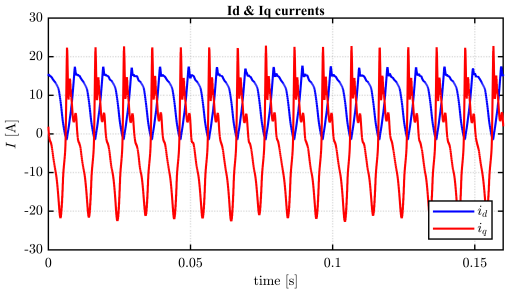

Figure 4.107: Threephase HIL OLF - I_{dq}

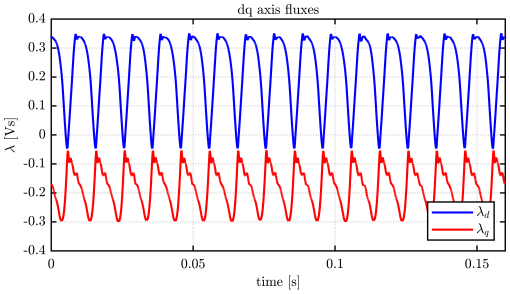

Figure 4.108: Threephase HIL OLF - λ_{dq}

4.3.1 Simulation Results

The open phase fault is the first of the so-called 'dissymmetrical' faults in three-phase applications exploited in this chapter. For this type of fault, the three phases of the system are not balanced or have significant differences between them, in this conditions one three phase system can be decomposed into three symmetrical systems mado of direct, inverse and homopolar sequences. In particular, the last two of these components generate highly time-varying torques that cannot be controlled by our control system. For this reason, in all the cases analysed, we find the presence of high torque ripples with very high peak-to-peak oscillations.

In addition, for the three-phase motor, it can be seen that the presence of the voltage feedforward, which uses the estimated machine fluxes for calculation, introduces further calculation errors into the control, reflected in an increase in peak transient currents in the healthy phases, as can be seen in the figure 4.109.

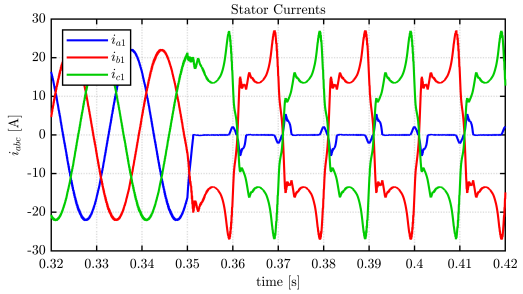

Figure 4.109: OLF - With feedforward

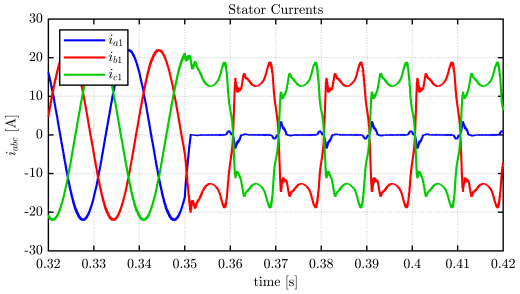

Figure 4.110: OLF - Without feedforward

The peak current obtained by maintaining the feedforward of the PI controllers is almost 30[A], whereas when no feedforward was used, this value drops below 20[A]. Therefore, for the rest of this discussion, which includes only dissymmetrical faults, the control of the three-phase motor in the images presented will be without feedforward.

In contrast to three-phase motors, in multi-phase motors (although even in this case the machine flows are estimated with larger errors than in the case of a healthy machine), the feedforward of the common-mode and differential components remains a valuable tool for preserving part of the machine performance under fault conditions.

It can also be observed that, for three-phase and nine-phase motors, the torque ripple is similar, the difference being that nine-phase has a greater positive average component than three-phase.

In contrast, for SPM motors, the torque ripple decreases significantly as the number of phases increases. In any case, both six-phase and twelve-phase motors manage to maintain an average torque slightly below that required.

4.4 Open Switch

An open switch failure in a two-level inverter is a type of critical malfunction that directly affects the performance and reliability of the entire system in which the inverter is used. This problem occurs when one or more semiconductor switches, such as IGBTs or MOSFETs, which are critical for switching current within the inverter, fail to close or conduct as they should. In two-level inverters, the ability to generate two voltage levels enables the efficient conversion of direct current (DC) to alternating current (AC), and any failure in this process can have immediate and significant consequences.

The causes of such a failure are varied and range from physical defects or wear of the components themselves to errors in the drive signal that may not activate the switch correctly. Fluctuations or instabilities in the power supply to the driver circuits can also contribute to the problem, as can over-temperature conditions that alter operation or damage the switches.

The repercussions of an open switch failure are wide-ranging and can manifest themselves in a variety of ways, from loss of inverter efficiency, which results in higher energy losses, to interruptions or malfunctions of powered equipment. Erratic switching also introduces harmonics and electrical noise into the system, potentially damaging other devices. In addition, the remaining functioning switches may be subjected to additional thermal stress, accelerating their degradation and reducing the life of the inverter.

To effectively address the problem of open switches, modern inverters are equipped with advanced monitoring and diagnostic systems that detect switch malfunctions in a timely manner. These safety systems are designed to intervene quickly, stopping inverter operation or adjusting its operation to minimise damage and protect the connected load.

Dealing with an open switch failure requires the replacement or repair of affected components to restore normal inverter operation. Prevention through regular maintenance and careful design, including effective cooling systems and adequate safety margins, is key to minimising the risk of such failures. Ultimately, understanding the dynamics of an open-switch failure and implementing appropriate preventive and monitoring measures is essential to maintain the efficiency and reliability of systems that depend on two-level inverters, thus ensuring the business continuity and safety of the entire system.

THOR_1x3ph Waveforms

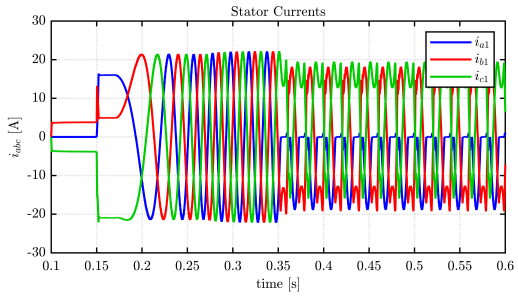

Figure 4.111: Threephase OSF - I_{abc}

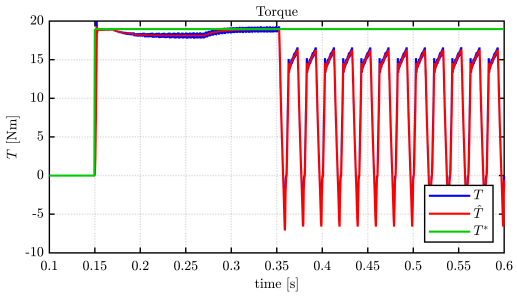

Figure 4.112: Threephase OSF - T_m

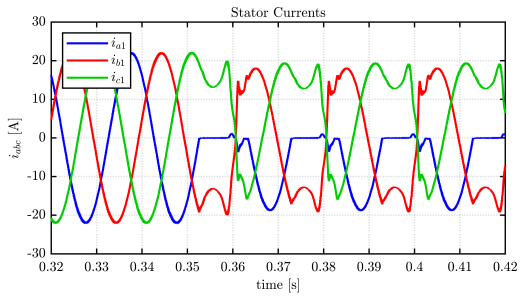

Figure 4.113: Threephase OSF - Zoom I_{abc}

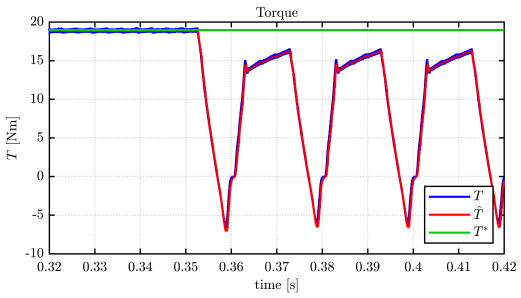

Figure 4.114: Threephase OSF - Zoom T_m

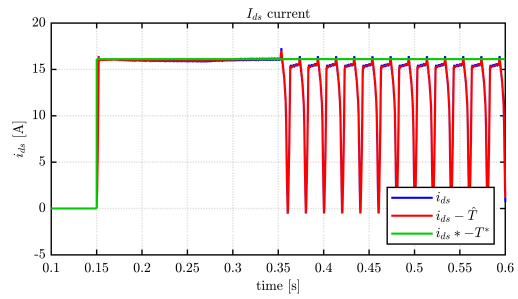

Figure 4.115: Threephase OSF - I_d

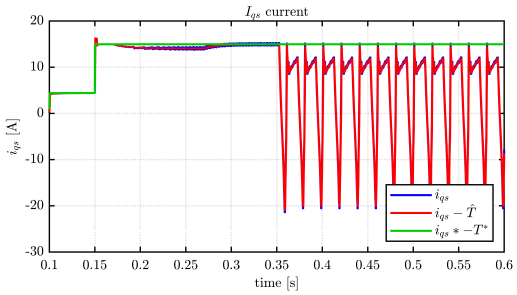

Figure 4.116: Threephase OSF - I_q

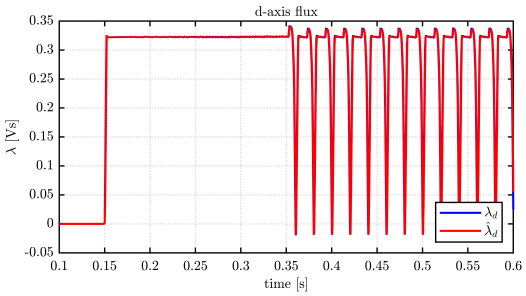

Figure 4.117: Threephase OSF - λ_d

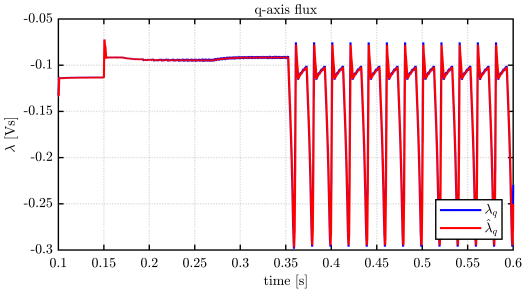

Figure 4.118: Threephase OSF - λ_q

THOR_3x3ph Waveforms

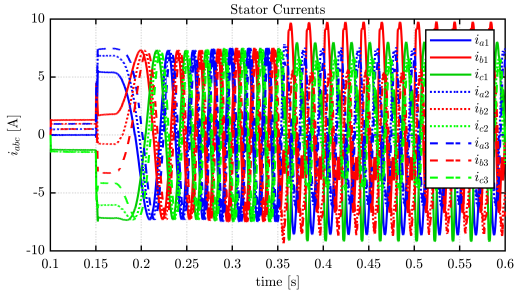

Figure 4.119: Nine-Phase OSF - I_{abc}

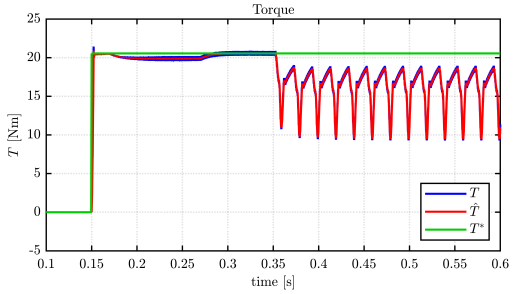

Figure 4.120: Nine-Phase OSF - T_m

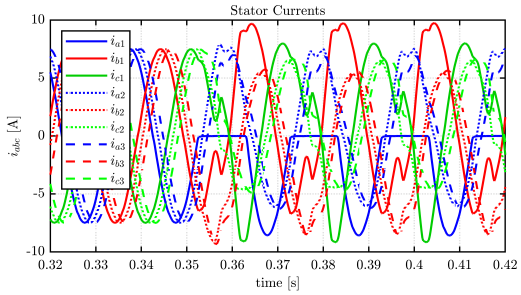

Figure 4.121: Nine-Phase OSF - Zoom I_{abc}

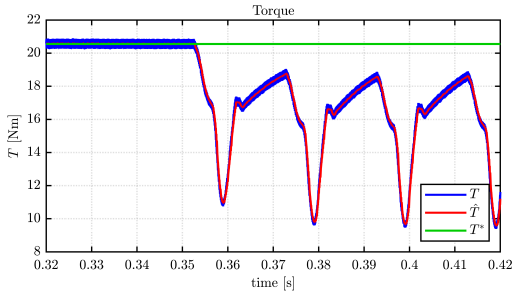

Figure 4.122: Nine-Phase OSF - Zoom T_m

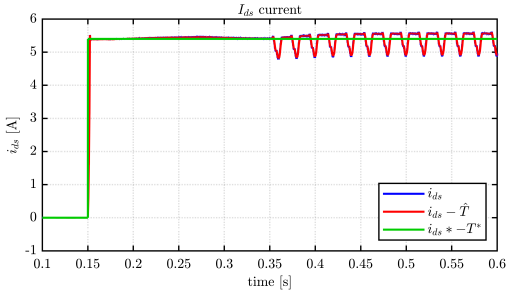

Figure 4.123: Nine-Phase OSF - I_d

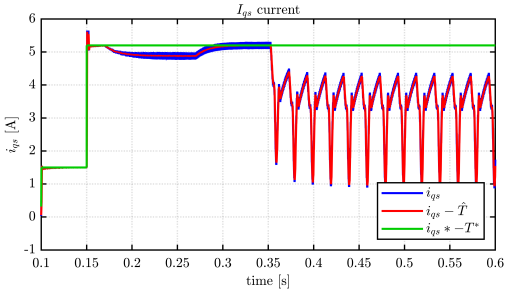

Figure 4.124: Nine-Phase OSF - I_q

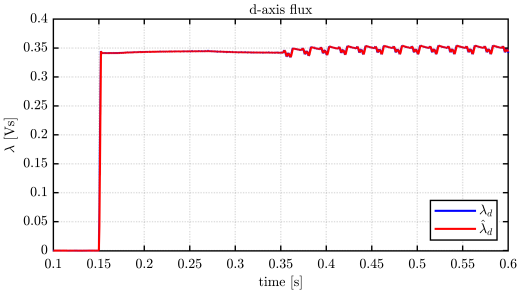

Figure 4.125: Nine-Phase OSF - λ_d

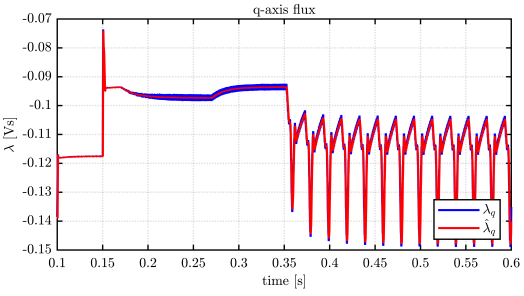

Figure 4.126: Nine-Phase OSF - λ_q

PMSM_2x3ph Waveforms

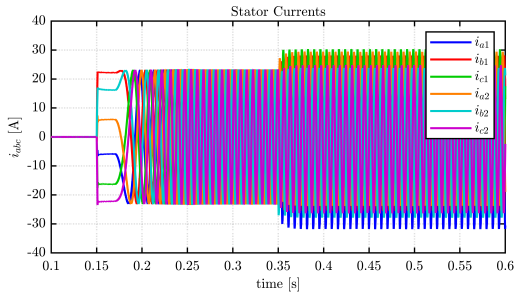

Figure 4.127: Six-Phases OSF - I_{abc}

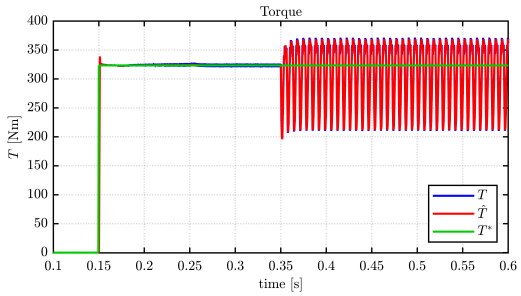

Figure 4.128: Six-Phases OSF - T_m

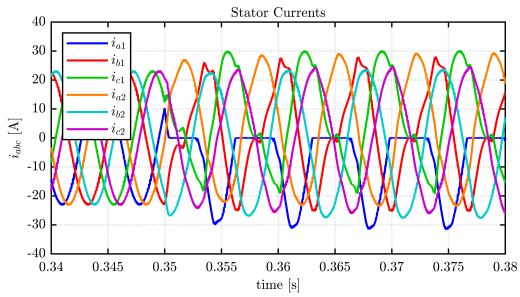

Figure 4.129: Six-Phases OSF - Zoom I_{abc}

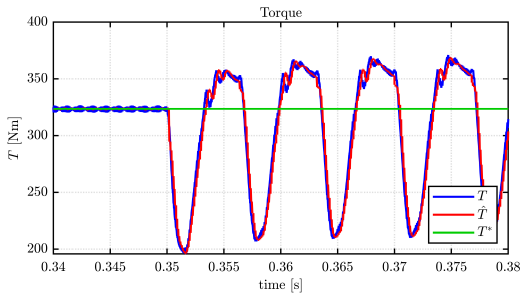

Figure 4.130: Six-Phases OSF - Zoom T_m

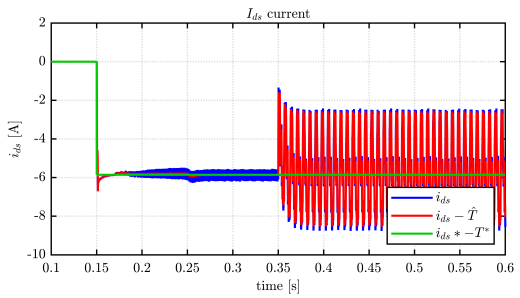

Figure 4.131: Six-Phases OSF - I_d

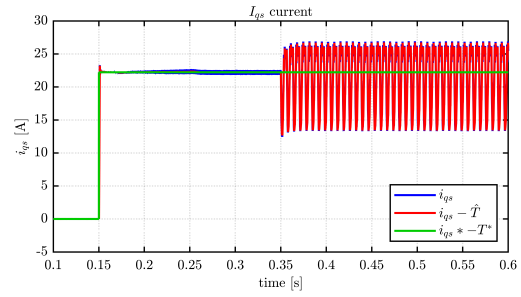

Figure 4.132: Six-Phases OSF - I_q

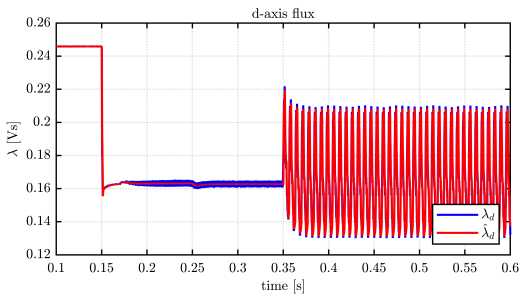

Figure 4.133: Six-Phases OSF - λ_d

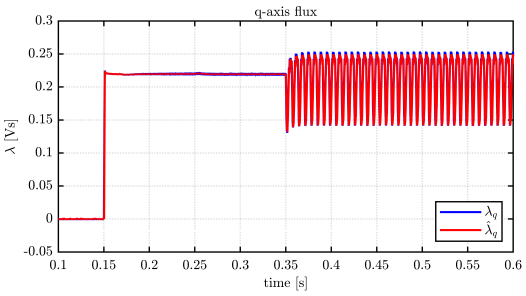

Figure 4.134: Six-Phases OSF - λ_q

PMSM_4x3ph Waveforms

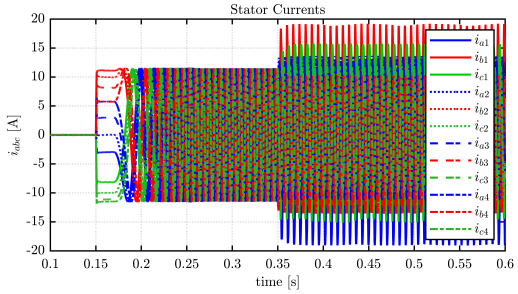

Figure 4.135: Twelve-Phases OSF - I_{abc}

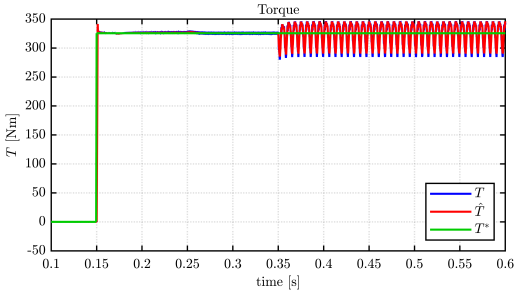

Figure 4.136: Twelve-Phases OSF - T_m

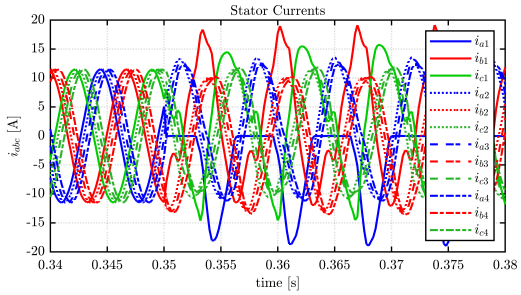

Figure 4.137: Twelve-Phases OSF - Zoom I_{abc}

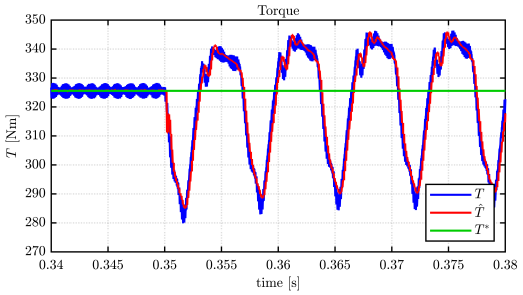

Figure 4.138: Twelve-Phases OSF - Zoom T_m

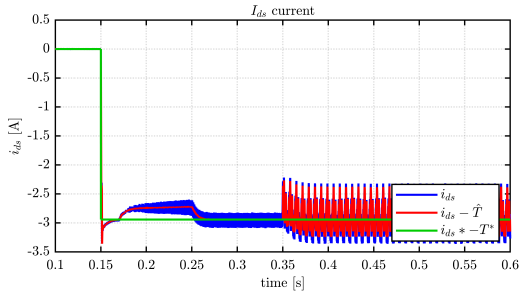

Figure 4.139: Twelve-Phases OSF - I_d

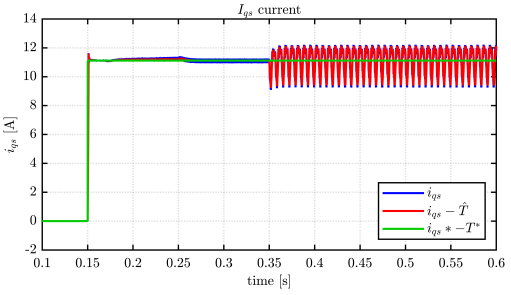

Figure 4.140: Twelve-Phases OSF - I_q

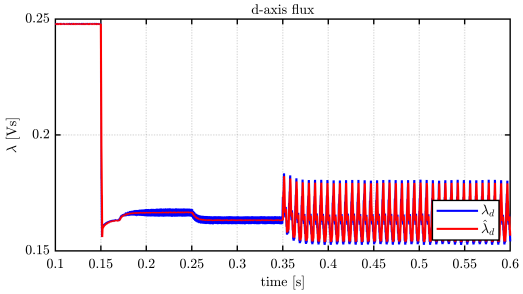

Figure 4.141: Twelve-Phases OSF - λ_d

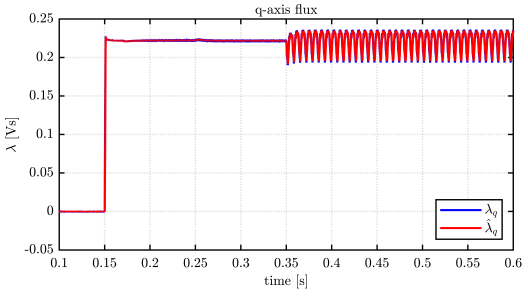

Figure 4.142: Twelve-Phases OSF - λ_q

THOR_1x3ph Real-time Waveforms

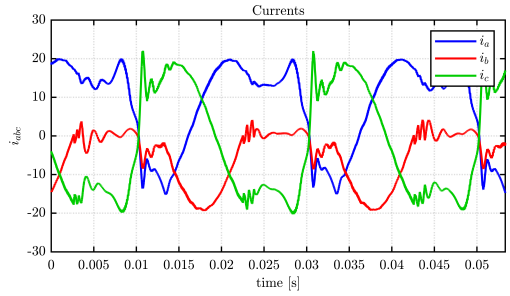

Figure 4.143: Threephase HIL OSF - I_{abc}

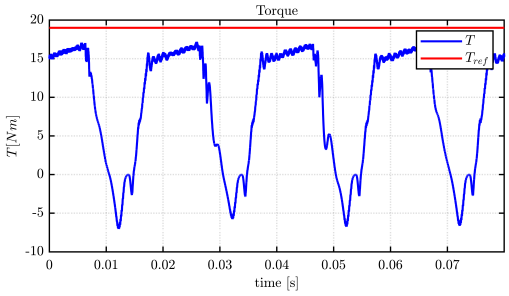

Figure 4.144: Threephase HIL OSF - T_m

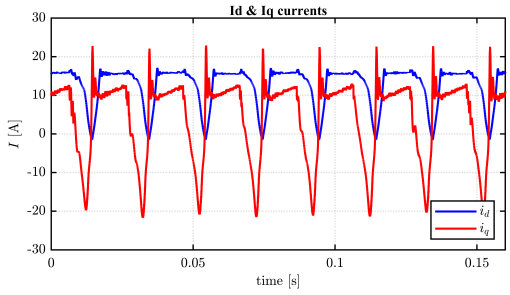

Figure 4.145: Threephase HIL OSF - I_{dq}

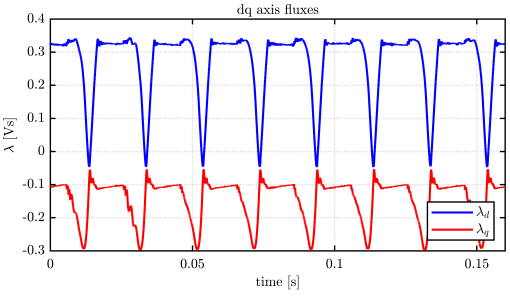

Figure 4.146: Threephase HIL OSF - λ_{dq}

4.4.1 Simulation Results

In fact, the open switch fault is an open phase fault that continues on the drive only for the time frame in which the switch would actually be closed that is half of the period of the desired voltage waveform imposed by the voltage source inverter.

For this reason, for the open switch fault many references are similar to those already made for the open phase, beginning, for example, with the presence of large ripples on the produced torques that the motors are able to deliver with the control set on the healthy machine, or the choice not to adopt the feedforward technique on the three-phase motor that reduces the peak stator currents on the phases that remain healthy.

Also, regarding the three-phase motor, it is noted that the open switch fault, compared to the open phase, produces slightly higher torque ripples. This happens because in the half-periods when all phases are operating properly, the control tries to raise the torque produced. Considering that the negative peak turns out to be the same in both cases (about -5 $N \cdot m$), this explains the slight increase in the torque ripple values. Lastly, for this motor the mean torque produced is higher for the same reasons that increased the ripple.

In contrast to the three-phase motor case, in the nine-phase motor not only the peak-to-peak torque ripple produced is about half as much as in the open phase fault case, but even the average torque produced is much more in line with the reference set by the control, demonstrating some resilience of the multistator approach with decoupling technique to dissymmetrical faults.

As for the six- and twelve-phase motors, slight differences are stock compared with the open phase case.For example, there is a slightly higher positive average torque output than in the previous case and, in addition, a ripple frequency that is halved compared with the previous case.

As far as the HIL simulation world is concerned, the recorded waveforms are masterfully in line with those reproduced in offline simulations, with very slight differences in terms of small current spikes.

4.5 Inter-turn Short Circuit

Introduced in 1.3.5, the inter-turn short circuit phenomenon occurs when the insulation between two or more turns of the same winding deteriorates to such an extent that a direct electrical connection can be made between them. The causes can be many, from ageing of the insulation due to thermal stress, to physical wear due to vibration, to overvoltages that exceed the insulation's resistance. Moisture and contaminants can also accelerate this degradation process, making the windings susceptible to such failures.

The consequences of an ITSC are not to be underestimated: the current flowing directly between the coils involved creates excessive heat points, which can further damage the insulation and extend the failure. This can lead to a number of problems, such as a decrease in device efficiency, increased energy losses, overheating and, in the worst case, total failure of the equipment.

Given the potential severity of an ITSC, it is crucial to be able to detect it early. Several diagnostic techniques exist, including the analysis of winding impedances and the use of thermography to identify areas of overheating. In addition, more sophisticated tests such as frequency response testing can reveal abnormalities in winding characteristics caused by short circuits between turns. The availability of real-time monitoring systems can speed up the identification of these faults, allowing rapid and targeted intervention.

When an ITSC is identified, it is crucial to act without delay to repair or replace the damaged winding. Depending on the extent of the fault, a local repair may be sufficient or, in more serious cases, a complete replacement of the winding or the entire device may be necessary.

Prevention plays a key role in minimising the risk of ITSC. Regular and careful maintenance, constant monitoring of operating conditions to avoid overloading, the use of quality insulation materials, and a design that minimises stress on the windings all help to reduce the likelihood of such failures. Implementing these preventive strategies not only extends the life of motors and transformers, but also ensures greater safety and reliability of the entire electrical system. In summary, dealing with ITSC requires a comprehensive approach that includes preventive measures, early detection capabilities and effective action to repair or replace damaged elements, thus ensuring the continuity and safety of operations.

Dealing back with the fault modeling purposes of this thesis, ITSC unlike other faults seen until here needs to be modeled into the motor model itself, emphasizing its distinct impact on motor performance.

To model inter-turn short circuits in the THOR three-phase machine, the voltage behind reactance approach is employed, since is widely used in other research applications such as [8], [15].

Using this method, each phase is represented as controlled voltage sources in series with an RL impeadance for each branch. The fault ratio μ is defined as the ratio of the number of shorted turns to the total number of turns in one phase ($\frac{N_f}{N}$).

Initially, there was challenges in modelling the inter-turn fault within the PLECS simulation environment, mainly due to the complexity of the pattern of ITSC and to make appropriate use of the 'Coupled Inductor' component in the PLECS library, witch requires clear input and output currents flowing in it. The representation of the fault layout, as shown in Figure 1.10, was not very intuitive,

as the arrangement of the inductors on a single plane appeared unclear. However, through subtle circuit manipulation, as shown in Figure 4.147,this issue was solved.

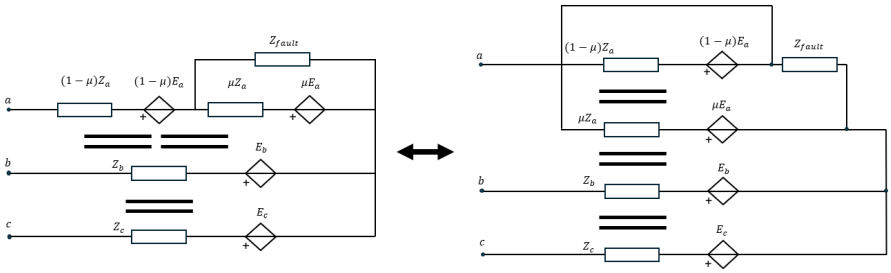

Figure 4.147: ITSC layout manupulation

4.5.1 VBR model for ITSC

The developed circuital VBR model in PLECS is shown in figure 4.148. As expected, it is composed of four controllable voltage sources (two of them for the faulted phase, and others two for the healty one).

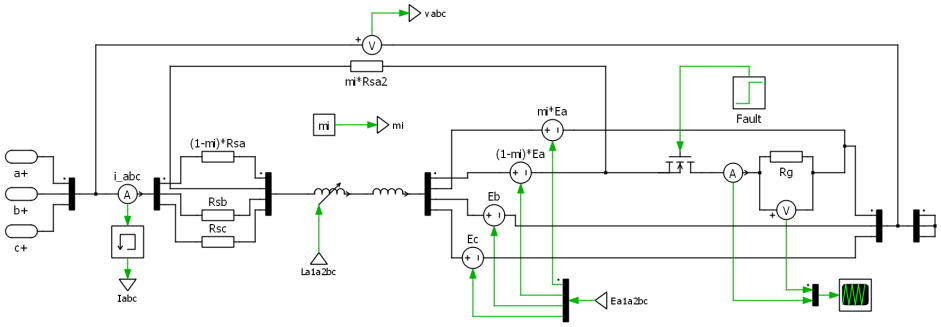

Figure 4.148: ITSC PLECS MotorModel

hereafter are reported some observations:

- Total Inductances and controlled voltage sources of the equivalent *abc* healty model are computed as suggested in 1.4.1.
- Inductances and controlled voltage sources of the faulted parth are computed as suggested in 1.3.5.
- The Coupled inductor block has a series strain inductor with a value of $10-6[H]$ and it is used in order to stabilize the simulation that had some convergence issues due to direct access of the coupled inductor block with the faulted path.
- The amperometric current measurement is delayed for the control in order to avoid the forming of algebrical loop involving other major components.
- The faulted path is modeled as a series of the following elements:

- An ideal Mosfet, with $0[S]$ conductance in the off state and $0[\Omega]$ resistance in the on state. This switch triggers the fault event.
- A strain inductor, with the value of $10^{-6}[H]$ that stabilizes the simulation.
- A fault resistance R_g equal to $0.05[\Omega]$, a realistic value for this kind of faults.
- Voltage and Current ideal measurements groups.

Comparison with CCG model under healty condition

Before proceeding with the simulation of the ITSC fault, in order to validate this proposed VBR model, it is necessary to make a comparison with the model previously developed using controlled current generators.

In the following, images of the trends in the magnitudes of torque, current and duty cycle are shown for both methods in order to show the fidelity of this representation introduced. The simulation methodology is the same as that presented in 3.6. As can be seen, the VBR model presented

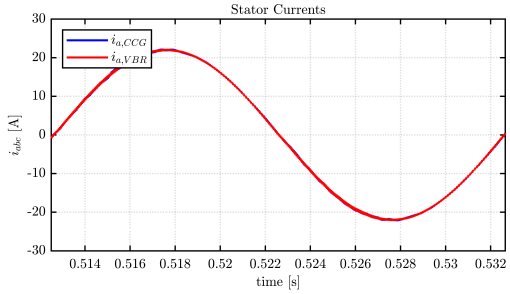

Figure 4.149: VBR vs CCG - I_a

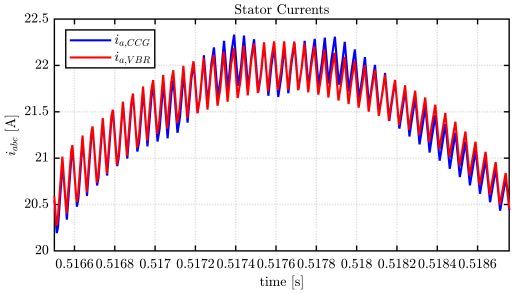

Figure 4.150: VBR vs CCG - Zoom I_a

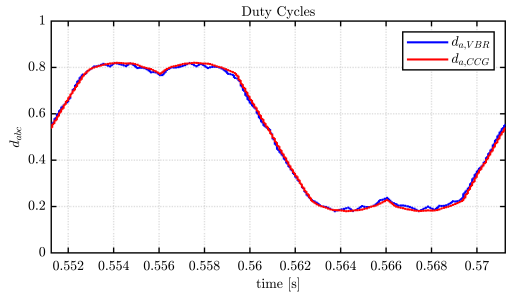

Figure 4.151: VBR vs CCG - Duty$_a$

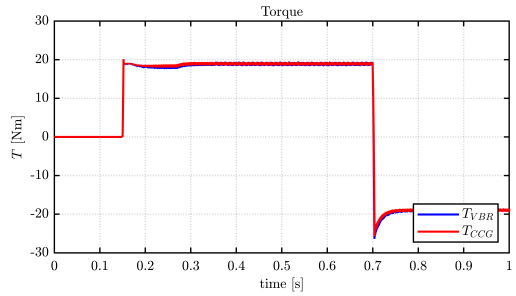

Figure 4.152: VBR vs CCG - T_m

faithfully replicates what was previously developed with very small discrepancies, allowing the simulation to be deepened in the case of ITSC failure.

4.5.2 Variable Fault Ratio Simulation

In order to evaluate the performance of the model and determine the threshold of percentage of shorted turns required for the ITSC fault to be considered severe, a parametric study is being conducted using the parameter $\mu = \frac{N_{faulted}}{N_tot}$.

This simulation was possible with few adjustments of the already developed model, introducing a variable fault ratio to feed the signals and variable resistor to interactively change the 'point' of the short circuit.

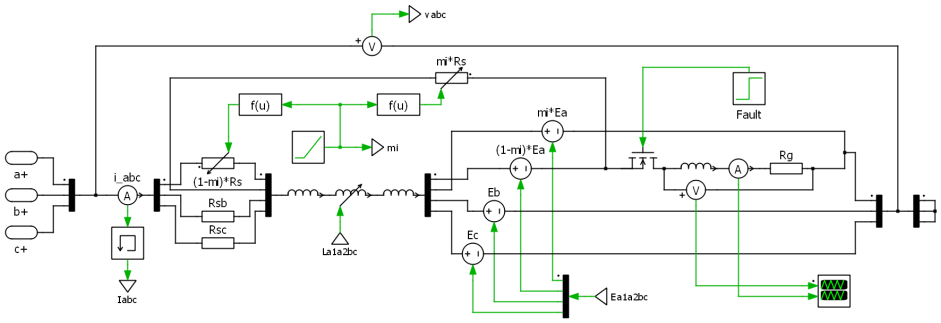

Figure 4.153: ITSC Variable Fault ratio

For this purpose, a simulation was configured in which the fault ratio varies over time as a ramp signal, with an angular coefficient equal to $\frac{25\% turns}{0.5[s]}$.

It was observed that, depending on the model configuration and the control parameters used, the failure is particularly severe even with a percentage of failed turns below 25% on the phase. Above this threshold, loss of motor control occurs. Below are images of waveforms with a failure ratio ranging from 0 al 25%.

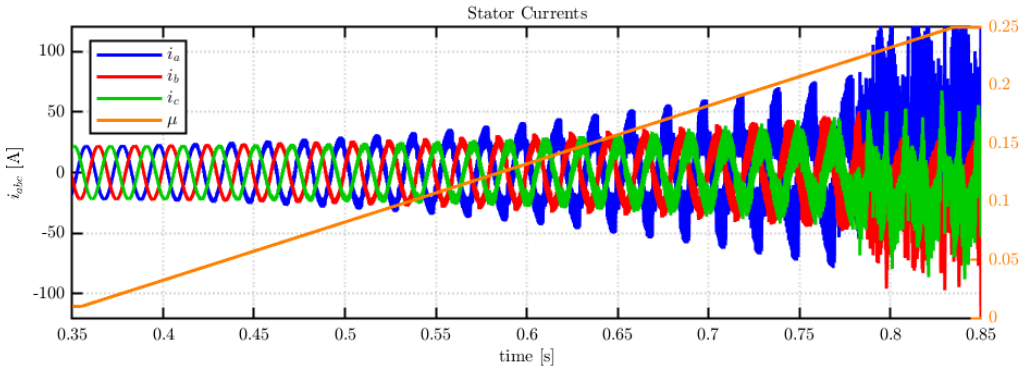

Figure 4.154: ITSC - Currents vs Fault ratio

THOR_1x3ph 10% Faulted Turns

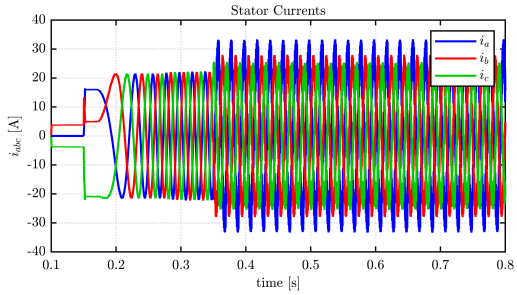

Figure 4.155: ITSC - I_{abc}

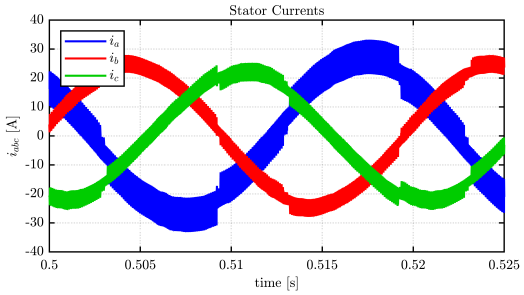

Figure 4.156: ITSC - Zoom I_{abc}

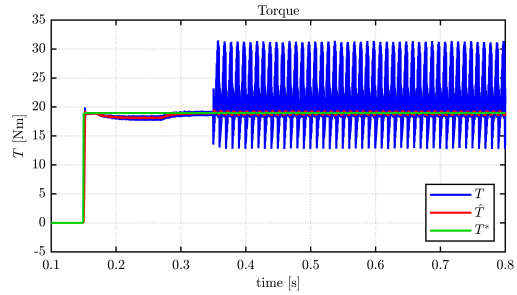

Figure 4.157: ITSC - T_m

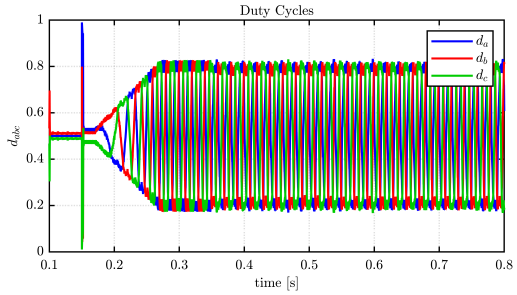

Figure 4.158: ITSC - Duty Cycles

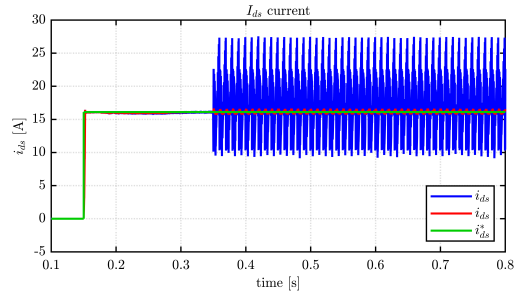

Figure 4.159: ITSC - I_d

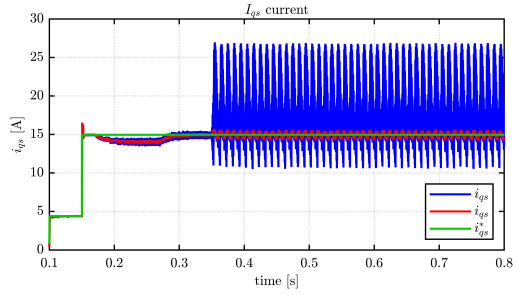

Figure 4.160: ITSC - I_q

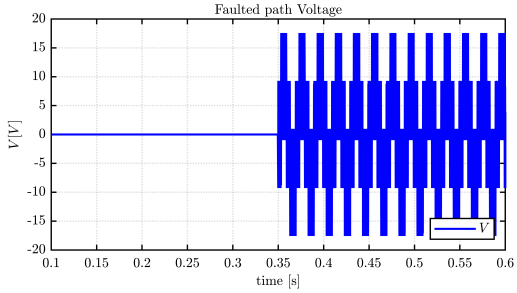

Figure 4.161: ITSC - Voltage on R_g

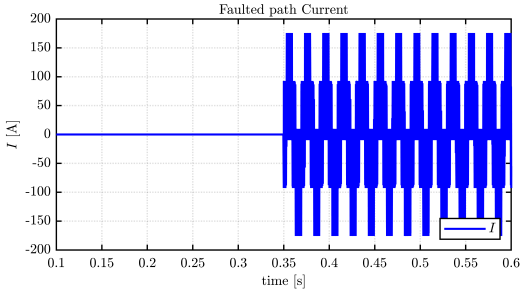

Figure 4.162: ITSC - Current on R_g

The simulation results on the ITSC with 10% of faulted turns on the first phase of the motors shows the negative effects on motor performance due to this continence.

Like other asymmetric faults, the interturn short circuit introduce very high torque ripples, that are comparable with the reference torque. This is due to the compresence of negative and zero sequence in both current and voltage quantities.

The maximum peak currents in the faulted path are about $175[A]$, while at the motor terminals the peak currents in the faulted phase are over $35[A]$.

In this concluding chapter, we have summarised key findings from our in-depth exploration of faults in multi-phase electrical drives and strategies for their detection and mitigation. We analysed several failure scenarios, including three-phase short circuits, inverter shutdowns, open phases, switch interruptions and short circuits between turns, highlighting how each affects the performance of drive systems. Through detailed simulations, we demonstrated the effectiveness of various mitigation methods, emphasising the importance of careful design and advanced control strategies to ensure the reliability and safety of electrical drives under fault conditions. This work not only contributes to the existing academic body of work, but also provides practical insights for industry, moving towards the development of more robust and resilient electrical drive systems.

Conclusions

This thesis has advanced the field of electric motor technology by developing and implementing unified circuit models in PLECS for multi-three-phase motors, integrating these models with SyreDrive, and focusing on fault simulation and validation through Hardware-in-the-Loop technologies. Through detailed chapters, it explored the motivation behind this research, the complexities of electric propulsion systems, the importance of reliability and fault tolerance, and the innovative approaches to modeling and simulation.

The work began by setting the stage with an in-depth look at the state of electric motors, emphasizing the shift towards multiphase systems for enhanced efficiency, reliability, and fault tolerance. It underscored the critical nature of robust modeling and simulation tools, leading to the development of SyreDrive—an extension designed to bridge the gap between design and dynamic simulation of electric motors and their control systems.

In subsequent chapters, the thesis meticulously detailed the modeling process of multi-three-phase motors within the PLECS environment. It highlighted the vectorized model's development for efficient simulation across different motor configurations, demonstrating its versatility and accuracy. This model's integration into SyreDrive facilitated a seamless transition from motor design to dynamic simulation, paving the way for comprehensive analysis and optimization of motor control strategies.

The exploration of fault modeling and simulation constituted a significant portion of this research. By simulating various fault scenarios, the thesis provided invaluable insights into the motors' resilience under adverse conditions. These simulations not only showcased the potential for real-time fault detection and mitigation but also highlighted the critical role of Hardware-in-the-Loop technologies in validating the models and control strategies under realistic operational conditions.

In conclusion, this thesis represents a significant step towards the realization of more efficient, reliable, and resilient electric propulsion systems. By combining advanced modeling techniques with practical validation approaches, it offers new perspectives on the design, control, and fault management of multi-phase motors. This work not only contributes to the academic body of knowledge but also holds considerable implications for the automotive, aerospace, and other sectors reliant on electric propulsion technologies.

Appendix A - Six Phase Motor_Control.c

This script outlines the digital control strategy employed for the six-phase motor model within PLECS, illustrating the complexity and adaptability required to manage multi-phase electrical drives effectively.

```
// // /*-------------------------------------------------------------------------*/
/* File: Motor_ctrl.c                                                          */
/* the parameter CTRL_TYPE decides the type of control                 */
/*  CTRL_TYPE = 0 - Current control                          */
/*  CTRL_TYPE = 1 - Flux control                             */
/*  CTRL_TYPE = 2 - Torque control                           */
/*  CTRL_TYPE = 3 - Speed control                            */
/*-------------------------------------------------------------------------*/
/* Motor type: Six-Phase SM                                                 */
/*-------------------------------------------------------------------------*/

    isabc1.a = InputSignal(0,0);
    isabc1.b = InputSignal(0,1);
    isabc1.c = InputSignal(0,2);
    vdc = InputSignal(0,3);

    theta_mec_meas =InputSignal(0,4); // encoder
    n_ref_in =InputSignal(0,5); // rpm
    T_ext =InputSignal(0,6); // reference torque
    isdq_ext.d =InputSignal(0,7); // Current reference
    isdq_ext.q =InputSignal(0,8); // Current reference

    Reset =InputSignal(0,9); // Black button
    Go =InputSignal(0,10);      // Blue  button
    Ctrl_type =InputSignal(0,11);
    inj_waveform =InputSignal(0,12); // Injected waveform (sinusoidal or squarewave)
    dem =InputSignal(0,13); // Current or Flux demoduation
    HS_ctrl =InputSignal(0,14); // High speed position error estimation technique
    SS_on =InputSignal(0,15); // Sensorless ON or OFF
    accel =InputSignal(0,16); // speed acceleration rpm/s
    Quad_Maps =InputSignal(0,17);
    lambda_M =InputSignal(0,18);
    th0 =InputSignal(0,19);
    isabc2.a =InputSignal(0,20);
    isabc2.b =InputSignal(0,21);
    isabc2.c =InputSignal(0,22);
  switch(State){

        case ERROR:
            // Variables Init
            pwm_stop = 1;
            counter = 0;

            offset_current_a = 2120;
            offset_current_b = 2120;
            offset_current_c = 2120;

            n_ref_in = 0.0f;
            omega_ref_in = 0.0f;
            omega_ref_ramp = 0.0f;
            accel = 1000; // rpm/s
            omega_elt_meas = 0.0f;
            omega_elt_meas_f = 0.0f;
            SinCos_elt_meas.sin = 0.0f;
            SinCos_elt_meas.cos = 1.0f;
            SinCos_elt_meas_old.sin = 0.0f;
```

```
            SinCos_elt_meas_old.cos = 1.0f;

            Ld = Ld_inic;
            Lq = Lq_inic;
            ld = Ld_inic;
            lq = Lq_inic;
            flux_nom = 0.0f; // rated flux (Vs)
            v0 = 0.0f; // phase dc voltage (V)

            isdq_refcm.d = 0.0f;
            isdq_refcm.q = 0.0f;
            lambda_dq.d = lambda_M;
            lambda_dq.q = 0.0f;
            lambda_CM_dq.d = lambda_M;
            lambda_CM_dq.q = 0.0f;


            isdq_refdm1.d = 0.0f;
            isdq_refdm1.q = 0.0f;
            isdq1.d = 0.0f;
            isdq1.q = 0.0f;
            vsdq_ref1.d = 0.0f;
            vsdq_ref1.q = 0.0f;
            lambda_OBS1.alpha = lambda_M*cos(PP*th0);
            lambda_OBS1.beta = lambda_M*sin(PP*th0);

            isdq_refdm2.d = 0.0f;
            isdq_refdm2.q = 0.0f;
            isdq2.d = 0.0f;
            isdq2.q = 0.0f;
            vsdq_ref2.d = 0.0f;
            vsdq_ref2.q = 0.0f;
            lambda_OBS2.alpha = lambda_M*cos(PP*th0);
            lambda_OBS2.beta = lambda_M*sin(PP*th0);

            isabc1.a = 0.0f;
            isabc1.b = 0.0f;
            isabc1.c = 0.0f;
            duty_abc1.a = 0.0f;
            duty_abc1.b = 0.0f;
            duty_abc1.c = 0.0f;

            isabc2.a = 0.0f;
            isabc2.b = 0.0f;
            isabc2.c = 0.0f;
            duty_abc2.a = 0.0f;
            duty_abc2.b = 0.0f;
            duty_abc2.c = 0.0f;


            if(Go) State= WAKE_UP;

            break;


        case WAKE_UP:
            duty_abc1.a = 0.5f;
            duty_abc1.b = 0.5f;
            duty_abc1.c = 0.5f;
            pwm_stop1 = 0.0f;

            duty_abc2.a = 0.5f;
```

```
119             duty_abc2.b = 0.5f;
120             duty_abc2.c = 0.5f;
121             pwm_stop2 = 0.0f;
122 
123             if (counter > 0.03/Ts){
124                 counter=0;
125                 State = READY;
126             }
127             counter++;
128 
129             break;
130         case READY:
131             duty_abc1.a = 0.5f;
132             duty_abc1.b = 0.5f;
133             duty_abc1.c = 0.5f;
134             pwm_stop1 = 0.0f;
135 
136             duty_abc2.a = 0.5f;
137             duty_abc2.b = 0.5f;
138             duty_abc2.c = 0.5f;
139             pwm_stop2 = 0.0f;
140 
141             counter = 0.0f;
142             PLL_var.intg = omega_elt_meas_f;
143             theta_PLL = theta_elt_meas;
144             if(Go) State = START;
145             break;
146         case START:
147             //-------------------Speed Compute---------------------------------//
148 
149             theta_elt_meas = PP * theta_mec_meas;
150             while(theta_elt_meas > PI)
151                 theta_elt_meas -= TWOPI;
152             while(theta_elt_meas < -PI)
153                 theta_elt_meas += TWOPI;
154             SinCos_elt_meas.sin = sin(theta_elt_meas);
155             SinCos_elt_meas.cos = cos(theta_elt_meas);
156 
157             speed_compute_sc(SinCos_elt_meas, &SinCos_elt_meas_old, &omega_elt_meas);
158             _Filter(omega_elt_meas, omega_elt_meas_f, Ts*TWOPI*50);
159             omega_mec_meas = omega_elt_meas/PP;
160             omega_mec_meas_f = omega_elt_meas_f/PP;
161             omega_mec_meas_rpm  = omega_mec_meas_f*30/PI;
162             SinCos_elt_meas_old.sin = SinCos_elt_meas.sin;
163               SinCos_elt_meas_old.cos = SinCos_elt_meas.cos;
164             //---------------------------------PLL-------------------------------//
165             if(SS_on) {
166                 PLL_par.kp = 2*OMEGA_B_PLL;
167                 PLL_par.ki = pow(OMEGA_B_PLL,2)*Ts;
168                 PLL_par.lim = RPM2RAD * nmax_mot * PP;
169                 PLL_var.ref = pos_err;
170                 PLL_var.fbk = 0;
171                 PIReg(&PLL_par, &PLL_var);
172                 omega_PLL  = PLL_var.out;
173                 theta_PLL += Ts*PLL_var.out; ;
174                 _Filter(omega_PLL, omega_elt_meas_f, Ts*TWOPI*25);
175                 if(theta_PLL >= TWOPI)
176                     theta_PLL -= TWOPI;
177                 if(theta_PLL <0)
178                     theta_PLL += TWOPI;
179                 SinCos_elt.sin = sin(theta_PLL);
180                 SinCos_elt.cos = cos(theta_PLL);
```

```
181                 position_error_real = asin(sin(theta_elt_meas - theta_PLL));
182             }
183             else { //Encorder
184                 SinCos_elt.sin = sin(theta_elt_meas);
185                 SinCos_elt.cos = cos(theta_elt_meas);
186                 omega_elt = omega_elt_meas_f;
187                 position_error_real = 0;
188             }
189             //------------------------------Control Type
    ------------------------------//
190             switch (Ctrl_type){
191                 case 0: //CurrentControl
192                     isdq_refcm.d = isdq_ext.d;
193                     isdq_refcm.q = isdq_ext.q;
194                 break;
195 
196                 case 2: //TorqueControl
197                     ReadLut(&ID_REF[0], fabs(T_ext), TMAX, TMIN, DT, INV_DT, &
    isdq_refcm.d);
198                     ReadLut(&IQ_REF[0], fabs(T_ext), TMAX, TMIN, DT, INV_DT, &
    isdq_refcm.q);
199                     isdq_refdm1.d=0.0f;
200                     isdq_refdm1.q=0.0f;
201                     switch(Quad_Maps){
202                         case 0:
203                             isdq_refcm.d = sgn(T_ext)*isdq_refcm.d;
204                         break;
205                         case 1:
206                             isdq_refcm.d = sgn(T_ext)*isdq_refcm.d;
207                         break;
208                         case 2:
209                             isdq_refcm.q = sgn(T_ext)*isdq_refcm.q;
210                         break;
211                     }
212                 break;
213 
214                 case 3: //Speed Control
215           omega_ref_in = n_ref_in * RPM2RAD;
216           ramp(omega_ref_in, accel * RPM2RAD*Ts, &omega_ref_ramp);
217           sp_var.ref = omega_ref_ramp;
218           sp_var.fbk = omega_elt/PP;
219           sp_par.lim = T_rated;
220           kp_w = 2*OMEGA_BW*J;
221           ki_w = pow(OMEGA_BW,2)*J;
222           sp_par.ki = ki_w*Ts;
223           sp_par.kp = kp_w;
224           PIReg(&sp_par, &sp_var);
225           T_ext = sp_var.out;
226 
227                     ReadLut(&ID_REF[0], fabs(T_ext), TMAX, TMIN, DT, INV_DT, &
    isdq_refcm.d);
228                     ReadLut(&IQ_REF[0], fabs(T_ext), TMAX, TMIN, DT, INV_DT, &
    isdq_refcm.q);
229                     isdq_refdm1.d=0.0f;
230                     isdq_refdm1.q=0.0f;
231                     switch (Quad_Maps){
232                         case 0:
233                             isdq_refcm.d = sgn(T_ext)*isdq_refcm.d;
234                         break;
235                         case 1:
236                             isdq_refcm.d = sgn(T_ext)*isdq_refcm.d;
237                         break;
```

```
                        case 2:
                            isdq_refcm.q = sgn(T_ext)*isdq_refcm.q;
                        break;
                    }
                    break;
                }


            _clarke(isabc1,isab1);
            _clarke1(isabc2,isab2);
            _rot(isab1,SinCos_elt,isdq1);
            _rot(isab2,SinCos_elt,isdq2);

            is_dm1.d = (isdq1.d-isdq2.d)*0.5f;
            is_dm1.q = (isdq1.q-isdq2.q)*0.5f;

            is_cm.d=(isdq1.d+isdq2.d)/2;
            is_cm.q=(isdq1.q+isdq2.q)/2;
            vsab_km11=vsab_ref1;
            vsab_km12=vsab_ref2;
            FluxObserver();
            //Compute_Inductance();


            //--------------------Current Vector Control--------------------//

            kp_cmd=OMEGA_BI*Ld_inic;
            ki_cmd=OMEGA_BI*RS;
            kp_dmd=OMEGA_BI*L_sigma;
            ki_dmd=OMEGA_BI*RS;

            kp_cmq=OMEGA_BI*Lq_inic;
            ki_cmq=OMEGA_BI*RS;
            kp_dmq=OMEGA_BI*L_sigma;
            ki_dmq=OMEGA_BI*RS;

            id_par.kp = kp_cmd;
            id_par.ki = ki_cmd*Ts;
            iq_par.kp = kp_cmq;
            iq_par.ki = ki_cmq*Ts;

            id_par1.kp = kp_dmd;
            id_par1.ki = ki_dmd*Ts;
            iq_par1.kp = kp_dmq;
            iq_par1.ki = ki_dmq*Ts;

            Current_loop(vdc,Imax_mot,isdq_refcm,is_cm,&id_par,&id_var,&iq_par,&iq_var
    ,&vsdq_cm_ref);
            Current_loop(vdc,Imax_mot,isdq_refdm1,is_dm1,&id_par1,&id_var1,&iq_par1,&
    iq_var1,&vsdq_dm_ref1);

            vsdq_cm_ref.d += RS*is_cm.d-omega_elt*lambda_dq.q;
            vsdq_cm_ref.q += RS*is_cm.q+omega_elt*lambda_dq.d;

            vsdq_dm_ref1.d += RS*is_dm1.d-omega_elt*L_sigma*is_dm1.q;
            vsdq_dm_ref1.q += RS*is_dm1.q+omega_elt*L_sigma*is_dm1.d;

            //Decoupling
            vsdq_ref1.d = vsdq_cm_ref.d+vsdq_dm_ref1.d;
            vsdq_ref1.q = vsdq_cm_ref.q+vsdq_dm_ref1.q;
            vsdq_ref2.d = vsdq_cm_ref.d-vsdq_dm_ref1.d;
            vsdq_ref2.q = vsdq_cm_ref.q-vsdq_dm_ref1.q;
```

```
298
299             dTheta = 1.5f*omega_elt*Ts;
300             SinCos_elt_dTheta.sin = SinCos_elt.sin*cosf(dTheta) +SinCos_elt.cos*sinf(
    dTheta);
301             SinCos_elt_dTheta.cos = SinCos_elt.cos*cosf(dTheta) -SinCos_elt.sin*sinf(
    dTheta);
302
303
304             _invrot(vsdq_ref1,SinCos_elt_dTheta,vsab_ref1);
305             _invclarke(vsab_ref1,vsabc_ref1);
306             PWMdut(vsabc_ref1,vdc,&duty_abc1);
307
308             _invrot(vsdq_ref2,SinCos_elt_dTheta,vsab_ref2);
309             _invclarke1(vsab_ref2,vsabc_ref2);
310             PWMdut(vsabc_ref2,vdc,&duty_abc2);
311
312      }
313
314
315             //Duty cycle saturation
316      if (duty_abc1.a>0.99f) duty_abc1.a = 0.99f;
317      if (duty_abc1.b>0.99f) duty_abc1.b = 0.99f;
318      if (duty_abc1.c>0.99f) duty_abc1.c = 0.99f;
319
320      if (duty_abc1.a<0.01f) duty_abc1.a = 0.01f;
321      if (duty_abc1.b<0.01f) duty_abc1.b = 0.01f;
322      if (duty_abc1.c<0.01f) duty_abc1.c = 0.01f;
323
324      if (pwm_stop1){
325          duty_abc1.a = 0.0f;
326          duty_abc1.b = 0.0f;
327          duty_abc1.c = 0.0f;
328          }
329
330      if (duty_abc2.a>0.99f) duty_abc2.a = 0.99f;
331      if (duty_abc2.b>0.99f) duty_abc2.b = 0.99f;
332      if (duty_abc2.c>0.99f) duty_abc2.c = 0.99f;
333
334      if (duty_abc2.a<0.01f) duty_abc2.a = 0.01f;
335      if (duty_abc2.b<0.01f) duty_abc2.b = 0.01f;
336      if (duty_abc2.c<0.01f) duty_abc2.c = 0.01f;
337
338      if (pwm_stop2){
339          duty_abc2.a = 0.0f;
340          duty_abc2.b = 0.0f;
341          duty_abc2.c = 0.0f;
342          }
343
344     OutputSignal(0,0) = duty_abc1.a;
345     OutputSignal(0,1) = duty_abc1.b;
346     OutputSignal(0,2) = duty_abc1.c;
347     OutputSignal(0,3) = pwm_stop1;
348     OutputSignal(0,4)  = omega_ref_ramp*60/TWOPI;
349     OutputSignal(0,5)  = omega_mec_meas_f*60/TWOPI;
350     OutputSignal(0,6)  = vsdq_cm_ref.d;
351     OutputSignal(0,7)  = vsdq_cm_ref.q;
352     OutputSignal(0,8)  = isab1.alpha;
353     OutputSignal(0,9)  = isab1.beta;
354     OutputSignal(0,10) = isdq_refcm.d;
355     OutputSignal(0,11) = is_cm.d;
356     OutputSignal(0,12) = isdq_refcm.q;
357     OutputSignal(0,13) = is_cm.q;
```

```
358     OutputSignal(0,14) = f_omega;
359     OutputSignal(0,15) = T_elt;
360     OutputSignal(0,16) = lambda_dq.d;
361     OutputSignal(0,17) = lambda_dq.q;
362     OutputSignal(0,18) = lambda_CM_dq.d;
363     OutputSignal(0,19) = lambda_CM_dq.q;
364     OutputSignal(0,20) = T_ext;
365     OutputSignal(0,21) = theta_PLL*180/PI;
366     OutputSignal(0,22) = omega_elt/PP*60/TWOPI;
367     OutputSignal(0,23) = ld;
368     OutputSignal(0,24) = lq;
369     OutputSignal(0,25) = ldq;
370     OutputSignal(0,26) = isabc1.a;
371     OutputSignal(0,27) = isabc1.b;
372     OutputSignal(0,28) = isabc1.c;
373     OutputSignal(0,29) = theta_elt_meas*180/PI;
374     OutputSignal(0,30) = pos_err*180/PI;
375     OutputSignal(0,31) = position_error_real*180/PI;
376     OutputSignal(0,32) = pos_err_LS*180/PI;
377     OutputSignal(0,33) = pos_err_HS*180/PI;
378     OutputSignal(0,34) = duty_abc2.a;
379     OutputSignal(0,35) = duty_abc2.b;
380     OutputSignal(0,36) = duty_abc2.c;
381     OutputSignal(0,37) = pwm_stop2;
```

Appendix B - Motor_Ctrl_0.c

```
// // /*--------------------------------------------------------------------*/
/* File: Motor_ctrl.c                                                      */
/* the parameter CTRL_TYPE decides the type of control    */
/*  CTRL_TYPE = 0 - Current control                       */
/*  CTRL_TYPE = 1 - Flux control                          */
/*  CTRL_TYPE = 2 - Torque control                        */
/*  CTRL_TYPE = 3 - Speed control                         */
/*--------------------------------------------------------------------*/
/* Motor type:                                    */
/*--------------------------------------------------------------------*/

vdc = InputSignal(0,0);

theta_mec_meas = InputSignal(0,1); // encoder
n_ref_in = InputSignal(0,2); // rpm
T_ext = InputSignal(0,3); // reference torque
isdq_ext.d = InputSignal(0,4); // Current reference
isdq_ext.q = InputSignal(0,5); // Current reference

Reset = InputSignal(0,6); // Black button
Go = InputSignal(0,7); // Blue  button

Ctrl_type = InputSignal(0,8);
inj_waveform = InputSignal(0,9); // Injected waveform (sinusoidal or squarewave)
dem = InputSignal(0,10); // Current or Flux demoduation
HS_ctrl = InputSignal(0,11); // High speed position error estimation technique (AF or
    APP)
SS_on = InputSignal(0,12); // Sensorless ON or OFF
accel = InputSignal(0,13); // speed acceleration rpm/s
Quad_Maps = InputSignal(0,14);
lambda_M = InputSignal(0,15);
th0 = InputSignal(0,16);

//----------------INIT STATE MACHINE----------------//

switch(State){

    case ERROR:
        // Variables Init
        pwm_stop = 1;
        counter = 0;

        offset_current_a = 2120;
        offset_current_b = 2120;
        offset_current_c = 2120;

        n_ref_in = 0.0f;
        omega_ref_in = 0.0f;
        omega_ref_ramp = 0.0f;
        accel = 1000;// rpm/s
        omega_elt_meas = 0.0f;
        omega_elt_meas_f = 0.0f;
        SinCos_elt_meas.sin = 0.0f;
        SinCos_elt_meas.cos = 1.0f;
        SinCos_elt_meas_old.sin = 0.0f;
        SinCos_elt_meas_old.cos = 1.0f;

        Ld = Ld_inic;
        Lq = Lq_inic;
```

```
59          ld = Ld_inic;
60          lq = Lq_inic;
61          flux_nom = 0.0f;  // rated flux (Vs)
62          v0 = 0.0f; // phase dc voltage (V)
63          isdq_ref.d = 0.0f;
64          isdq_ref.q = 0.0f;
65          isdq.d = 0.0f;
66          isdq.q = 0.0f;
67          vsdq_ref.d = 0.0f;
68          vsdq_ref.q = 0.0f;
69
70          lambda_CM_dq.d = 0.0f;
71          lambda_CM_dq.q = 0.0f;
72
73 //----------------READY STATE----------------//
74
75 if (counter > 0.03/Ts){
76     counter=0;
77     State = READY;
78 }
79
80 //----------------SPEED CTRL----------------//
81
82 case 3: //SpeedControl
83     omega_ref_in = n_ref_in * RPM2RAD;
84     ramp(omega_ref_in, accel * RPM2RAD*Ts, &omega_ref_ramp);
85     sp_var.ref  = omega_ref_ramp;
86     sp_var.fbk  = omega_elt/PP;
87     sp_par.lim  = T_rated;
88     kp_w        = 2*OMEGA_BW*J;
89     ki_w        = pow(OMEGA_BW,2)*J;
90     sp_par.ki   = ki_w*Ts;
91     sp_par.kp   = kp_w;
92     PIReg(&sp_par, &sp_var);
93     T_ext       = sp_var.out;
94
95 //----------------READLUT----------------//
96
97 switch (Quad_Maps){
98 case 0:
99     ReadLut2d(&FD_LUT[0][0], fabs(isdq.d), fabs(isdq.q), DIDD, INV_DIDD, DIQD,
        INV_DIQD , ID_TAB_MAX, ID_TAB_MIN, IQ_TAB_MAX , IQ_TAB_MIN, n_size, &lambda_CM_dq.
        d);
100    ReadLut2d(&FQ_LUT[0][0], fabs(isdq.q), fabs(isdq.d), DIQQ, INV_DIQQ, DIQD,
        INV_DIQD , IQ_TAB_MAX, IQ_TAB_MIN, ID_TAB_MAX , ID_TAB_MIN, n_size, &lambda_CM_dq.
        q);
101    if (isdq.d < 0)
102        lambda_CM_dq.d = -lambda_CM_dq.d;
103    if (isdq.q < 0)
104        lambda_CM_dq.q = -lambda_CM_dq.q;
105 break;
106
107 case 1:
108    ReadLut2d(&FD_LUT[0][0], fabs(isdq.d), isdq.q, DIDD, INV_DIDD, DIQD, INV_DIQD ,
        ID_TAB_MAX, ID_TAB_MIN, IQ_TAB_MAX , IQ_TAB_MIN, n_size, &lambda_CM_dq.d);
109    ReadLut2d(&FQ_LUT[0][0], isdq.q, fabs(isdq.d), DIQQ, INV_DIQQ, DIDQ, INV_DIDQ ,
        IQ_TAB_MAX, IQ_TAB_MIN, ID_TAB_MAX , ID_TAB_MIN, n_size, &lambda_CM_dq.q);
110    if (isdq.d < 0)
111        lambda_CM_dq.d = -lambda_CM_dq.d;
112 break;
113
114 case 2:
```

```
115     ReadLut2d(&FD_LUT[0][0], isdq.d, fabs(isdq.q), DIDD, INV_DIDD, DIQD, INV_DIQD ,
        ID_TAB_MAX, ID_TAB_MIN, IQ_TAB_MAX , IQ_TAB_MIN, n_size, &lambda_CM_dq.d);
116     ReadLut2d(&FQ_LUT[0][0], fabs(isdq.q), isdq.d, DIQQ, INV_DIQQ, DIDQ, INV_DIDQ ,
        IQ_TAB_MAX, IQ_TAB_MIN, ID_TAB_MAX , ID_TAB_MIN, n_size, &lambda_CM_dq.q);
117     if (isdq.q < 0)
118         lambda_CM_dq.q = -lambda_CM_dq.q;
119 break;
120 }
```

Appendix C - Print_Control_Script.m

```
1
2  function print_PLECS_control_script_2(ctrlFolder_path,n_set,Quad_Maps)
3
4  %Initialization
5
6  line.inputs = 36;
7  line.outputs = 37;
8  line.ia = 94;
9
10 Motor_ctrl_0_path = [ctrlFolder_path '\Motor_ctrl_plecs_0.c'];
11
12 fid = fopen(Motor_ctrl_0_path,'r');
13 i = 1;
14 tline = fgetl(fid);
15 readData{i} = tline;
16 while ischar(tline)
17     i = i+1;
18     tline = fgetl(fid);
19     readData{i} = tline;
20 end
21 fclose(fid);
22
23 Motor_ctrl_0 = string(readData)';
24
25 %% -------------------- Add Inpunts -------------------- %%
26
27 Ctrl_Script = Motor_ctrl_0(1:31);
28 x=17;
29 for i=1:n_set
30     Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"isabc%d.a = InputSignal(0,%d);",i,x
       )];
31     x = x+1;
32 end
33 for i=1:n_set
34     Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"isabc%d.b = InputSignal(0,%d);",i,x
       )];
35     x = x+1;
36 end
37 for i=1:n_set
38     Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"isabc%d.c = InputSignal(0,%d);",i,x
       )];
39     x = x+1;
40 end
41
42 %% --------------------ERROR STATE-------------------- %%
43
44
45 Ctrl_Script = [Ctrl_Script ; strings(2,1)];
46 Ctrl_Script = [Ctrl_Script ; Motor_ctrl_0(35:71)]; %%INIT STATE MACHINE
47 Ctrl_Script = [Ctrl_Script ; strings(1,1)];
48 Ctrl_Script = [Ctrl_Script ; sprintf(blanks(12)+"isdq_refcm.d = 0.0f;")];
49 Ctrl_Script = [Ctrl_Script ; sprintf(blanks(12)+"isdq_refcm.q = 0.0f;")];
50 switch(Quad_Maps)
51     case 0 %SyR Motor
52         Ctrl_Script = [Ctrl_Script ; sprintf(blanks(12)+"lambda_dq.d = 0.0f;")];
53         Ctrl_Script = [Ctrl_Script ; sprintf(blanks(12)+"lambda_dq.q = 0.0f;")];
54         Ctrl_Script = [Ctrl_Script ; sprintf(blanks(12)+"lambda_CM_dq.d = 0.0f;")];
55         Ctrl_Script = [Ctrl_Script ; sprintf(blanks(12)+"lambda_CM_dq.q = 0.0f;")];
56     case 1 %PM-SyR
```

```
57          Ctrl_Script = [Ctrl_Script ; sprintf(blanks(12)+"lambda_dq.d = 0.0f;")];
58          Ctrl_Script = [Ctrl_Script ; sprintf(blanks(12)+"lambda_dq.q = -lambda_M;")];
59          Ctrl_Script = [Ctrl_Script ; sprintf(blanks(12)+"lambda_CM_dq.d = 0.0f;")];
60          Ctrl_Script = [Ctrl_Script ; sprintf(blanks(12)+"lambda_CM_dq.q = -lambda_M;")
        ];
61      case 2 % PM
62          Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"lambda_dq.d = lambda_M;")];
63          Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"lambda_dq.q = 0.0f;")];
64          Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"lambda_CM_dq.d = lambda_M;")];
65          Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"lambda_CM_dq.q = 0.0f;")];
66  end
67
68  Ctrl_Script = [Ctrl_Script; strings(2,1)];
69  index_end = length(Motor_ctrl);
70  for i=1:n_set
71      Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"isdq_refdm%d.d = 0.0f;",i)];
72      Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"isdq_refdm%d.q = 0.0f;",i)];
73      Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"isdq%d.d = 0.0f;",i)];
74      Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"isdq%d.q = 0.0f;",i)];
75      Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"vsdq_ref%d.d = 0.0f;",i)];
76      Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"vsdq_ref%d.q = 0.0f;",i)];
77  end
78
79  for i=1:n_set
80      Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"isabc%d.a = 0.0f;",i)];
81      Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"isabc%d.b = 0.0f;",i)];
82      Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"isabc%d.c = 0.0f;",i)];
83      Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"duty_abc%d.a = 0.0f;",i)];
84      Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"duty_abc%d.b = 0.0f;",i)];
85      Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"duty_abc%d.c = 0.0f;",i)];
86      Ctrl_Script = [Ctrl_Script; strings(1,1)];
87  end
88
89  Ctrl_Script = [Ctrl_Script; strings(1,1)];
90  Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"if(Go) State= WAKE_UP;")];
91  Ctrl_Script = [Ctrl_Script; strings(1,1)];
92  Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"break;")];
93  Ctrl_Script = [Ctrl_Script; strings(2,1)];
94  Ctrl_Script = [Ctrl_Script; sprintf(blanks(8)+"case WAKE_UP:")];
95
96  %% --------------------WAKE UP -------------------- %%
97
98  for i=1:n_set
99      Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"duty_abc%d.a = 0.5f;",i)];
100     Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"duty_abc%d.b = 0.5f;",i)];
101     Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"duty_abc%d.c = 0.5f;",i)];
102     Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"pwm_stop%d = 0.0f;",i)];
103     Ctrl_Script = [Ctrl_Script; strings(1,1)];
104 end
105
106 Ctrl_Script = [Ctrl_Script; Motor_ctrl_0(75:78)];%%READY STATE
107 Ctrl_Script = [Ctrl_Script;sprintf(blanks(12)+"counter++;")];
108 Ctrl_Script = [Ctrl_Script; strings(1,1)];
109 Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"break;")];
110 Ctrl_Script = [Ctrl_Script; sprintf(blanks(8)+"case READY:")];
111
112 %% --------------------READY -------------------- %%
113
114 for i=1:n_set
115     Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"duty_abc%d.a = 0.5f;",i)];
116     Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"duty_abc%d.b = 0.5f;",i)];
117     Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"duty_abc%d.c = 0.5f;",i)];
```

```
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"pwm_stop%d = 0.0f;",i)];
    Ctrl_Script = [Ctrl_Script; strings(1,1)];
end
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"counter = 0.0f;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"PLL_var.intg = omega_elt_meas_f;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"theta_PLL = theta_elt_meas;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"if(Go) State = START;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"break;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(8)+"case START:")];
%% --------------------START-------------------- %%

%%--------------------Speed Computation--------------------%%
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"//-------------------Speed Compute
    ---------------------------------//")];
Ctrl_Script = [Ctrl_Script; strings(1,1)];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"theta_elt_meas = PP * theta_mec_meas
    ;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"while(theta_elt_meas > PI)")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"theta_elt_meas -= TWOPI;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"while(theta_elt_meas < -PI)")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"theta_elt_meas += TWOPI;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"SinCos_elt_meas.sin = sin(
    theta_elt_meas);")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"SinCos_elt_meas.cos = cos(
    theta_elt_meas);")];
Ctrl_Script = [Ctrl_Script; strings(1,1)];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"speed_compute_sc(SinCos_elt_meas, &
    SinCos_elt_meas_old, &omega_elt_meas);")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"_Filter(omega_elt_meas,
    omega_elt_meas_f, Ts*TWOPI*50);")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"omega_mec_meas = omega_elt_meas/PP;")
    ];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"omega_mec_meas_f = omega_elt_meas_f/PP
    ;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"omega_mec_meas_rpm = omega_mec_meas_f
    *30/PI;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"SinCos_elt_meas_old.sin =
    SinCos_elt_meas.sin;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"  SinCos_elt_meas_old.cos =
    SinCos_elt_meas.cos;")];


Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"//---------------------------------PLL
    -------------------------------//")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"if(SS_on) { ")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"PLL_par.kp = 2*OMEGA_B_PLL;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"PLL_par.ki = pow(OMEGA_B_PLL,2)*Ts;")
    ];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"PLL_par.lim = RPM2RAD * nmax_mot * PP
    ;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"PLL_var.ref = pos_err;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"PLL_var.fbk = 0;")];

Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"PIReg(&PLL_par, &PLL_var);")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"omega_PLL = PLL_var.out;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"theta_PLL += Ts*PLL_var.out; ;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"_Filter(omega_PLL, omega_elt_meas_f,
    Ts*TWOPI*25);")];

Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"if(theta_PLL >= TWOPI)")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20)+"theta_PLL -= TWOPI;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"if(theta_PLL <0)")];
```

```
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20)+"theta_PLL += TWOPI;")];

Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"SinCos_elt.sin = sin(theta_PLL);")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"SinCos_elt.cos = cos(theta_PLL);")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"position_error_real = asin(sin(
    theta_elt_meas - theta_PLL));")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"}")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"else { //Encorder ")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"SinCos_elt.sin = sin(theta_elt_meas)
    ;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"SinCos_elt.cos = cos(theta_elt_meas)
    ;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"omega_elt = omega_elt_meas_f;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"position_error_real = 0;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"}")];

%%--------------------Control trajectory--------------------%%

Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"//------------------------------
    Control Type ------------------------------//")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"switch (Ctrl_type){")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"case 0: //CurrentControl")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20)+"isdq_refcm.d = isdq_ext.d;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20)+"isdq_refcm.q = isdq_ext.q;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"break;")];
Ctrl_Script = [Ctrl_Script; strings(1,1)];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"case 2: //TorqueControl")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20)+"ReadLut(&ID_REF[0], fabs(T_ext), TMAX,
     TMIN, DT, INV_DT, &isdq_refcm.d);")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20)+"ReadLut(&IQ_REF[0], fabs(T_ext), TMAX,
     TMIN, DT, INV_DT, &isdq_refcm.q);")];

for i=1:(n_set-1)
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(20)+"isdq_refdm%d.d=0.0f;",i)];
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(20)+"isdq_refdm%d.q=0.0f;",i)];
end

Ctrl_Script = [Ctrl_Script; sprintf(blanks(20)+"switch(Quad_Maps){")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20+4)+"case 0:")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20+8)+"isdq_refcm.d = sgn(T_ext)*isdq_refcm
    .d;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20+4)+"break;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20+4)+"case 1:")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20+8)+"isdq_refcm.d = sgn(T_ext)*isdq_refcm
    .d;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20+4)+"break;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20+4)+"case 2:")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20+8)+"isdq_refcm.q = sgn(T_ext)*isdq_refcm
    .q;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20+4)+"break;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20)+"}")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"break;")];
Ctrl_Script = [Ctrl_Script; strings(1,1)];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"case 3: //Speed Control")];
Ctrl_Script = [Ctrl_Script; Motor_ctrl_0(82:93)]; %%SPEED CTRL
Ctrl_Script = [Ctrl_Script; strings(1,1)];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20)+"ReadLut(&ID_REF[0], fabs(T_ext), TMAX,
     TMIN, DT, INV_DT, &isdq_refcm.d);")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20)+"ReadLut(&IQ_REF[0], fabs(T_ext), TMAX,
     TMIN, DT, INV_DT, &isdq_refcm.q);")];
for i=1:(n_set-1)
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(20)+"isdq_refdm%d.d=0.0f;",i)];
```

```
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(20)+"isdq_refdm%d.q=0.0f;",i)];
end

Ctrl_Script = [Ctrl_Script; sprintf(blanks(20)+"switch (Quad_Maps){")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20+4)+"case 0:")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20+8)+"isdq_refcm.d = sgn(T_ext)*isdq_refcm
    .d;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20+4)+"break;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20+4)+"case 1:")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20+8)+"isdq_refcm.d = sgn(T_ext)*isdq_refcm
    .d;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20+4)+"break;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20+4)+"case 2:")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20+8)+"isdq_refcm.q = sgn(T_ext)*isdq_refcm
    .q;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20+4)+"break;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20)+"}")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(20)+"break;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(16)+"}")];
Ctrl_Script = [Ctrl_Script; strings(2,1)];

% --------------------rotational transformation--------------------%%
for i=1:n_set
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(8+4)+"_clarke%d(isabc%d,isab%d);",i,i,i
    )];
end

for i=1:n_set
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(8+4)+"_rot(isab%d,SinCos_elt,isdq%d);",
    i,i)];
end
Ctrl_Script = [Ctrl_Script; sprintf("//--------------------Flux Observer
    --------------------//")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(8+4)+"isdq=is_cm;")];
Ctrl_Script = [Ctrl_Script; Motor_ctrl_0(97:120)]; %%READLUT
Ctrl_Script = [Ctrl_Script; sprintf(blanks(8+4)+"T_elt=N_PHASES*0.5f*PP*(lambda_CM_dq.
    d*is_cm.q* - lambda_CM_dq.q*is_cm.d);")];
Ctrl_Script = [Ctrl_Script; strings(1,1)];

for i=1:n_set
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(8+4)+"vsab_km1%d=vsab_ref%d;",i,i)];
end

Ctrl_Script = [Ctrl_Script; sprintf(blanks(8+4)+"vsab_km1=vsab_ref1;")];

%% --------------------Current Decoupling--------------------%%
tmp_s = sprintf('_Decoupling(');

for i=1:n_set
tmp_s = [tmp_s sprintf('isdq%d,',i)];
end

tmp_s = [tmp_s sprintf('is_cm')];

for i=1:(n_set-1)
tmp_s = [tmp_s sprintf(',is_dm%d',i)];
end

Ctrl_Script = [Ctrl_Script; sprintf(blanks(8+4)+"%s);",tmp_s)];
Ctrl_Script = [Ctrl_Script; strings(1,1)];

%% --------------------Current Vector Control--------------------%%
```

```
271
272 Ctrl_Script = [Ctrl_Script; strings(1,1)];
273 Ctrl_Script = [Ctrl_Script; sprintf(blanks(8+4)+"//--------------------Current Vector
    Control--------------------//")];
274 Ctrl_Script = [Ctrl_Script; strings(1,1)];
275 Ctrl_Script = [Ctrl_Script; sprintf(blanks(8+4)+"kp_cmd=OMEGA_BI*Ld_inic;")];
276 Ctrl_Script = [Ctrl_Script; sprintf(blanks(8+4)+"ki_cmd=OMEGA_BI*RS;")];
277 Ctrl_Script = [Ctrl_Script; sprintf(blanks(8+4)+"kp_dmd=OMEGA_BI*L_sigma;")];
278 Ctrl_Script = [Ctrl_Script; sprintf(blanks(8+4)+"ki_dmd=OMEGA_BI*RS;")];
279 Ctrl_Script = [Ctrl_Script; strings(1,1)];
280 Ctrl_Script = [Ctrl_Script; sprintf(blanks(8+4)+"kp_cmq=OMEGA_BI*Lq_inic;")];
281 Ctrl_Script = [Ctrl_Script; sprintf(blanks(8+4)+"ki_cmq=OMEGA_BI*RS;")];
282 Ctrl_Script = [Ctrl_Script; sprintf(blanks(8+4)+"kp_dmq=OMEGA_BI*L_sigma;")];
283 Ctrl_Script = [Ctrl_Script; sprintf(blanks(8+4)+"ki_dmq=OMEGA_BI*RS;")];
284
285 Ctrl_Script = [Ctrl_Script; strings(1,1)];
286
287 Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"id_par.kp = kp_cmd;")];
288 Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"id_par.ki = ki_cmd*Ts;")];
289 Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"iq_par.kp = kp_cmq;")];
290 Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"iq_par.ki = ki_cmq*Ts;")];
291 Ctrl_Script = [Ctrl_Script; strings(1,1)];
292
293 % define PI regulators gains
294 for i=2:n_set
295     Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"id_par%d.kp = kp_dmd;",i-1)];
296     Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"id_par%d.ki = ki_dmd*Ts;",i-1)];
297     Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"iq_par%d.kp = kp_dmq;",i-1)];
298     Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"iq_par%d.ki = ki_dmq*Ts;",i-1)];
299     Ctrl_Script = [Ctrl_Script; strings(1,1)];
300 end
301
302 % define current loops
303 Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"Current_loop(vdc,Imax_mot,isdq_refcm,
    is_cm,&id_par,&id_var,&iq_par,&iq_var,&vsdq_cm_ref);")];
304 for i=2:n_set
305     Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"Current_loop(vdc,Imax_mot,
    isdq_refdm%d,is_dm%d,&id_par%d,&id_var%d,&iq_par%d,&iq_var%d,&vsdq_dm_ref%d);",i
    -1,i-1,i-1,i-1,i-1,i-1,i-1)];
306 end
307
308 %% --------------------Feedforward--------------------%%
309
310 Ctrl_Script = [Ctrl_Script; strings(2,1)];
311
312 Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"vsdq_cm_ref.d += RS*is_cm.d-omega_elt*
    lambda_dq.q;")];
313 Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"vsdq_cm_ref.q += RS*is_cm.q+omega_elt*
    lambda_dq.d;")];
314 Ctrl_Script = [Ctrl_Script; strings(1,1)];
315
316 for i=2:n_set
317     Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"vsdq_dm_ref%d.d += RS*is_dm%d.d-
    omega_elt*L_sigma*is_dm%d.q;",i-1,i-1,i-1)];
318     Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"vsdq_dm_ref%d.q += RS*is_dm%d.q+
    omega_elt*L_sigma*is_dm%d.d;",i-1,i-1,i-1)];
319     Ctrl_Script = [Ctrl_Script; strings(1,1)];
320 end
321
322 %% --------------------Voltage Decoupling--------------------%%
323 Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"//Voltage Decoupling")];
324
```

```
tmp_s = sprintf('_InvDecoupling(');
tmp_s = [tmp_s sprintf('vsdq_cm_ref')];

for i=1:(n_set-1)
    tmp_s = [tmp_s sprintf(',vsdq_dm_ref%d',i)];
end

for i=1:n_set
    tmp_s = [tmp_s sprintf(',vsdq_ref%d',i)];
end

Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"%s);",tmp_s)];
Ctrl_Script = [Ctrl_Script; strings(1,1)];

%% --------------------PWM GENERATION--------------------%%
for i=1:n_set
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"_invrot(vsdq_ref%d,SinCos_elt,
    vsab_ref%d);",i,i)];
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"_invclarke%d(vsab_ref%d,vsabc_ref%
    d);",i,i,i)];
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"PWMduty(vsabc_ref%d,vdc,&duty_abc%
    d);",i,i)];
    Ctrl_Script = [Ctrl_Script; strings(1,1)];
end
Ctrl_Script = [Ctrl_Script; sprintf(blanks(5)+"}")];
Ctrl_Script = [Ctrl_Script; strings(2,1)];
% duty cycle saturation
Ctrl_Script = [Ctrl_Script; sprintf(blanks(12)+"//Duty cycle saturation")];
for i=1:n_set
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(5)+"if (duty_abc%d.a>0.99f) duty_abc%d.
    a = 0.99f;",i,i)];
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(5)+"if (duty_abc%d.b>0.99f) duty_abc%d.
    b = 0.99f;",i,i)];
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(5)+"if (duty_abc%d.c>0.99f) duty_abc%d.
    c = 0.99f;",i,i)];
    Ctrl_Script = [Ctrl_Script; strings(1,1)];
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(5)+"if (duty_abc%d.a<0.01f) duty_abc%d.
    a = 0.01f;",i,i)];
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(5)+"if (duty_abc%d.b<0.01f) duty_abc%d.
    b = 0.01f;",i,i)];
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(5)+"if (duty_abc%d.c<0.01f) duty_abc%d.
    c = 0.01f;",i,i)];
    Ctrl_Script = [Ctrl_Script; strings(1,1)];
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(5)+"if (pwm_stop%d){",i)];
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(9)+"duty_abc%d.a = 0.0f;",i)];
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(9)+"duty_abc%d.b = 0.0f;",i)];
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(9)+"duty_abc%d.c = 0.0f;",i)];
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(9)+"}")];
    Ctrl_Script = [Ctrl_Script; strings(1,1)];
end
%% Outputs
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,0)  = omega_ref_ramp*60/
    TWOPI;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,1)  = omega_mec_meas_f
    *60/TWOPI;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,2)  = vsdq_cm_ref.d;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,3)  = vsdq_cm_ref.q;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,4)  = isab1.alpha;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,5)  = isab1.beta;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,6) = isdq_refcm.d;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,7) = is_cm.d;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,8) = isdq_refcm.q;")];
```

```
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,9) = is_cm.q;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,10) = f_omega;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,11) = T_elt;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,12) = lambda_dq.d;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,13) = lambda_dq.q;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,14) = lambda_CM_dq.d;")
    ];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,15) = lambda_CM_dq.q;")
    ];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,16) = T_ext;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,17) = theta_PLL*180/PI
    ;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,18) = omega_elt/PP*60/
    TWOPI;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,19) = ld;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,20) = lq;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,21) = ldq;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,22) = isabc1.a;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,23) = isabc1.b;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,24) = isabc1.c;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,25) = theta_elt_meas
    *180/PI;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,26) = pos_err*180/PI;")
    ];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,27) =
    position_error_real*180/PI;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,28) = pos_err_LS*180/PI
    ;")];
Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,29) = pos_err_HS*180/PI
    ;")];
k = 30;
for i=1:n_set
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,%d) = duty_abc%d.a
    ;",k,i)];
    k=k+1;
end
for i=1:n_set
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,%d) = duty_abc%d.b
    ;",k,i)];
    k=k+1;
end
for i=1:n_set
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,%d) = duty_abc%d.c
    ;",k,i)];
    k=k+1;
end
for i=1:n_set
    Ctrl_Script = [Ctrl_Script; sprintf(blanks(4)+"OutputSignal(0,%d) = pwm_stop%d;",k
    ,i)];
    k=k+1;
end
Ctrl_Script = [Ctrl_Script; strings(2,1)];
Ctrl_Script = [Ctrl_Script; strings(2,1)];
%% -------------------- Print New File --------------------%%
Motor_ctrl_path = [ctrlFolder_path '\Motor_ctrl_plecs.c'];
fid = fopen(Motor_ctrl_path, 'w');
for i = 1:numel(Ctrl_Script)
    fprintf(fid,'%s\n', Ctrl_Script{i});
end
fclose(fid);
end
```

Appendix D - Compute_TVSD.m

```
function Compute_TVSD(ctrlFolder_path,n_set)
%% --------------------Define parameters--------------------%%
l = n_set;      % number of winding sets
k = 3;      % number of phases per each winding set
n = k*l;    % total number of phases
%% ---------------Define theta_n MATRIX with phases position-------------%
theta_n = zeros(1,n);
for j=1:k
    for i=1:l
        theta_n(i+(j-1)*l) = (180/n)*(2*l*(j-1)+i-1);
    end
end
%% -----------------Define non-zero sequence subspace constants-----------%
% generate c vector with odd numbers <n
C0 =1; x=1;
while(C0<n)
    c(x) = C0;
    x=x+1;
    C0=C0+2;
end
% find zero sequence harmonics constants C_zs
x=1; i=1;
while(i*k<n)
    if(rem(i*k,2)~=0)
        C_zs(x) = i*k;
        x=x+1;
    end
    i=i+1;
end
% Find non-zero sequence harmonics constants C_nzs, removing the C_zs
% constants from the c vector
[tmp,~] = ismember(c,C_zs);
index_tmp = find(tmp==1);
C_nzs = c;
C_nzs(index_tmp)=[];
%% Compute T VSD Matrix
sigma = 2/n;            % amplitude invariant transformation
T_VSD = zeros(n,n);     % initialize matrix
row=1;
for x=1:length(C_nzs)
    c = C_nzs(x);
    T_VSD(row,:) = cosd(c*theta_n);
    T_VSD(row+1,:) = sind(c*theta_n);
    row=row+2;
end
% Separate neutral points zero sequence subspaces
z=1;
for x=row:n
    T_VSD(x,z) = 1;
    T_VSD(x,z+l) = 1;
    T_VSD(x,z+2*l) = 1;
    z=z+1;
end
T =sigma*T_VSD;
inv_T = inv(T);
%% --------------------Save file--------------------%%
file_name = ['T_VSD' '.mat'];
save([ctrlFolder_path '\SimMatFiles\' file_name],"T","inv_T");
end
```

Appendix E - ComputeDecouplingMatrix.m

```
function ComputeDecouplingMatrix(ctrlFolder_path,n_set)
%% Get User Macros path
UserMacrosH_path = [ctrlFolder_path '\User_functions\Inc\User_Macros.h'];
%% number of sets
n = n_set;
%% -------------------- Compute decoupling Matrix -------------------- %%
TD = zeros(n,n);
TD(1,:) = 1;
x=1;
row = 2;
for k=1:(n-1)
    wk = sqrt((n*n-n*k)/(n-k+1));
    qk = -sqrt(n/((n-k)*(n-k+1)));
    TD(row,x) = wk;
    TD(row,x+1:end) = qk;
    x=x+1;
    row = row+1;
end
TD = (1/n)*TD;
inv_TD = inv(TD);

%% -------------------- Open and read User Macros -------------------- %%
fid = fopen(UserMacrosH_path,'r');
i = 1;
tline = fgetl(fid);
readData{i} = tline;
while ischar(tline)
    i = i+1;
    tline = fgetl(fid);
    readData{i} = tline;
end
fclose(fid);
readData(end) = [];
User_Macros = string(readData)';
%% --------------------Print Direct Decoupling Algorithm -------------------- %%
User_Macros = [ User_Macros; strings(2,1)];
User_Macros = [User_Macros;sprintf("//Decoupling algorithm dq1 dq2..dqn -> dq_cm
    dq_dm1 dq_dm2...dq_dm(n-1)")];
User_Macros = [ User_Macros; strings(1,1)];
%% Define Macro
tmp_s = sprintf('#define _Decoupling(');
for i=1:n
tmp_s = [tmp_s sprintf('dq%d,',i)];
end
tmp_s = [tmp_s sprintf('dq_cm')];
for i=1:(n-1)
tmp_s = [tmp_s sprintf(',dq_dm%d',i)];
end
User_Macros = [User_Macros; sprintf("%s); \\",tmp_s)];
%% Common mode subspace

tmp_d = sprintf('dq_cm.d =');
tmp_q = sprintf('dq_cm.q =');

for i=1:n
    tmp_d = [tmp_d sprintf('+(%.4f)*dq%d.d',TD(1,i),i)];
    tmp_q = [tmp_q sprintf('+(%.4f)*dq%d.q',TD(1,i),i)];
end
User_Macros = [User_Macros; sprintf(blanks(4)+"%s; \\",tmp_d)];
```

```
User_Macros = [User_Macros; sprintf(blanks(4)+"%s; \\",tmp_q)];

%% Differential mode subspace

for i=1:(n-1)
    tmp_d = sprintf('dq_dm%d.d =',i);
    tmp_q = sprintf('dq_dm%d.q =',i);
    for j=1:n
        tmp_d = [tmp_d sprintf('+(%.4f)*dq%d.d',TD(i+1,j),j)];
        tmp_q = [tmp_q sprintf('+(%.4f)*dq%d.q',TD(i+1,j),j)];
    end
    User_Macros = [User_Macros; sprintf(blanks(4)+"%s; \\",tmp_d)];
    User_Macros = [User_Macros; sprintf(blanks(4)+"%s; \\",tmp_q)];
end

%% Inverse Decoupling Algorithm

User_Macros = [ User_Macros; strings(2,1)];
User_Macros = [User_Macros;sprintf("//Decoupling algorithm dq_cm dq_dm1 dq_dm2...dq_dm
    (n-1)-> dq1 dq2..dqn  ")];
User_Macros = [ User_Macros; strings(1,1)];

%% Define Macro
tmp_s = sprintf('#define _InvDecoupling(');
tmp_s = [tmp_s sprintf('dq_cm')];

for i=1:(n-1)
    tmp_s = [tmp_s sprintf(',dq_dm%d',i)];
end

for i=1:n
    tmp_s = [tmp_s sprintf(',dq%d',i)];
end

User_Macros = [User_Macros; sprintf("%s); \\",tmp_s)];

for i=1:n
    tmp_d = sprintf('dq%d.d=(%.4f)*dq_cm.d',i,inv_TD(i,1));
    tmp_q = sprintf('dq%d.q=(%.4f)*dq_cm.q',i,inv_TD(i,1));
    for j=1:(n-1)
        tmp_d = [tmp_d sprintf('+(%.4f)*dq_dm%d.d',inv_TD(i,j+1),j)];
        tmp_q = [tmp_q sprintf('+(%.4f)*dq_dm%d.q',inv_TD(i,j+1),j)];
    end
    User_Macros = [User_Macros; sprintf(blanks(4)+"%s; \\",tmp_d)];
    User_Macros = [User_Macros; sprintf(blanks(4)+"%s; \\",tmp_q)];
end
%% Write in User Macros
fid = fopen(UserMacrosH_path, 'w');
for i = 1:numel(User_Macros)
    fprintf(fid,'%s\n', User_Macros{i});
end
fclose(fid);
end
```

Appendix F - PrintClarke.m

```
    function PrintClarke(ctrlFolder_path,n_set)
%% Get User_Macros.h path
UserMacrosH_path = [ctrlFolder_path '\User_functions\Inc\User_Macros.h'];
l = n_set;%%Number of sets
k=3;%%Phases for each set
n = k*l;%%Total number of phases
%% -------------------- Define theta_n MATRIX -------------------- %%
theta_n = zeros(l,k);
for j=1:l
    for i=1:k
        theta_n(j,i) = (180/n)*(2*l*(i-1)+j-1);
    end
end
%% -------------------- Compute Clake transformation -------------------- %%
for j=1:l
    tmp_cos = (2/3)*[cosd(theta_n(j,1)) cosd(theta_n(j,2)) cosd(theta_n(j,3))];
    tmp_sin = (2/3)*[sind(theta_n(j,1)) sind(theta_n(j,2)) sind(theta_n(j,3))];
    eval(['clarke_' num2str(j) '=[ tmp_cos; tmp_sin];']);
end
%% -------------------- Compute Inverse Clake transformation -------------------- %%
for j=1:l
    tmp_cos = [cosd(theta_n(j,1)) ; cosd(theta_n(j,2)); cosd(theta_n(j,3))];
    tmp_sin = [sind(theta_n(j,1)) ; sind(theta_n(j,2)) ; sind(theta_n(j,3))];
    eval(['clarke_inv_' num2str(j) '=[ tmp_cos tmp_sin];']);
end
%% -------------------- Read User_Macros.h file -------------------- %%
fid = fopen(UserMacrosH_path,'r');
i = 1;
tline = fgetl(fid);
readData{i} = tline;
while ischar(tline)
    i = i+1;
    tline = fgetl(fid);
    readData{i} = tline;
end
fclose(fid);
readData(end) = [];
%% -------------------- Add Clarke -------------------- %%
User_Macros = string(readData)';
User_Macros = [ User_Macros; strings(2,1)];
User_Macros = [User_Macros;sprintf("//Direct Clarke transformation (a,b,c)--> (alpha,
    beta)")];
User_Macros = [ User_Macros; strings(2,1)];
for i=1:l
    clarke_tmp = eval(['clarke_' num2str(i)]);
    User_Macros = [User_Macros;sprintf("#define _clarke%d(abc,ab); \\",i)];
    User_Macros = [ User_Macros;sprintf(blanks(4)+"ab.alpha = %.4f*abc.a+(%.4f)*abc.b
    +(%.4f)*abc.c;  \\",clarke_tmp(1,1),clarke_tmp(1,2),clarke_tmp(1,3))];
    User_Macros = [ User_Macros;sprintf(blanks(4)+"ab.beta = %.4f*abc.a+(%.4f)*abc.b
    +(%.4f)*abc.c;  \\",clarke_tmp(2,1),clarke_tmp(2,2),clarke_tmp(2,3))];
    User_Macros = [ User_Macros; strings(1,1)];
end
User_Macros = [User_Macros;sprintf("//Inverse Clarke transformation (alpha,beta)--> (a
    ,b,c)")];
User_Macros = [ User_Macros; strings(2,1)];
for i=1:l
    clarke_inv_tmp = eval(['clarke_inv_' num2str(i)]);
    User_Macros = [User_Macros;sprintf("#define _invclarke%d(ab,abc);  \\",i)];
    User_Macros = [ User_Macros;sprintf(blanks(4)+"abc.a = %.4f*ab.alpha+(%.4f)*ab.
```

```
    beta;  \\",clarke_inv_tmp(1,1),clarke_inv_tmp(1,2))];
    User_Macros = [ User_Macros;sprintf(blanks(4)+"abc.b = %.4f*ab.alpha+(%.4f)*ab.
    beta;  \\",clarke_inv_tmp(2,1),clarke_inv_tmp(2,2))];
    User_Macros = [ User_Macros;sprintf(blanks(4)+"abc.c = %.4f*ab.alpha+(%.4f)*ab.
    beta;  \\",clarke_inv_tmp(3,1),clarke_inv_tmp(3,2))];
    User_Macros = [ User_Macros; strings(1,1)];
end
%% -------------------- Write new file -------------------- %%
fid = fopen(UserMacrosH_path, 'w');
for i = 1:numel(User_Macros)
    fprintf(fid,'%s\n', User_Macros{i});
end
fclose(fid);
end
```

Appendix G - print_PLECS_fault_script.m

```
function print_PLECS_fault_script(ctrlFolder_path,n_set)
%%
%Variables Declaration
Fault_Script=[""];
for i=1:n_set
    Fault_Script=[Fault_Script;sprintf("float qa_in%d,qan_in%d,qb_in%d,qbn_in%d,qc_in%
    d,qcn_in%d;",i,i,i,i,i,i)];
    Fault_Script=[Fault_Script;sprintf("float qa_out%d,qan_out%d,qb_out%d,qbn_out%d,
    qc_out%d,qcn_out%d;",i,i,i,i,i,i)];
end
Fault_Script=[Fault_Script;sprintf("int fault_type;")];
Fault_Script=[Fault_Script;sprintf("float trigger;")];
Fault_Script=[Fault_Script;sprintf("")];

fid = fopen([ctrlFolder_path '\Fault_Scripts\fault_declarations.c'], 'w');
for i = 1:numel(Fault_Script)
    fprintf(fid,'%s\n', Fault_Script{i});
end
fclose(fid);

%%
Fault_Script=[""];
%Input import
l=0;

for j=1:n_set
    Fault_Script=[Fault_Script;sprintf("qa_in%d = InputSignal(0,%d);",j,l)];
    l=l+1;
end
for j=1:n_set
    Fault_Script=[Fault_Script;sprintf("qan_in%d = InputSignal(0,%d);",j,l)];
    l=l+1;
end
for j=1:n_set
    Fault_Script=[Fault_Script;sprintf("qb_in%d = InputSignal(0,%d);",j,l)];
    l=l+1;
end
for j=1:n_set
    Fault_Script=[Fault_Script;sprintf("qbn_in%d = InputSignal(0,%d);",j,l)];
    l=l+1;
end
for j=1:n_set
    Fault_Script=[Fault_Script;sprintf("qc_in%d = InputSignal(0,%d);",j,l)];
    l=l+1;
end
for j=1:n_set
    Fault_Script=[Fault_Script;sprintf("qcn_in%d = InputSignal(0,%d);",j,l)];
    l=l+1;
end
Fault_Script=[Fault_Script;sprintf("trigger = InputSignal(0,%d);",l)];
Fault_Script=[Fault_Script;sprintf("fault_type = InputSignal(0,%d);",l+1)];


%%
%Switch machine
l=0; %Counter for switch machine
Fault_Script=[Fault_Script;sprintf("if(trigger==0){")];
Fault_Script = [Fault_Script; sprintf("//--------------------------------No fault
    -------------------------------//")];
```

```
57
58 for i=1:n_set
59     Fault_Script=[Fault_Script;sprintf(blanks(4)+"qa_out%d = qa_in%d;",i,i)];
60     Fault_Script=[Fault_Script;sprintf(blanks(4)+"qan_out%d = qan_in%d;",i,i)];
61     Fault_Script=[Fault_Script;sprintf(blanks(4)+"qb_out%d = qb_in%d;",i,i)];
62     Fault_Script=[Fault_Script;sprintf(blanks(4)+"qbn_out%d = qbn_in%d;",i,i)];
63     Fault_Script=[Fault_Script;sprintf(blanks(4)+"qc_out%d = qc_in%d;",i,i)];
64     Fault_Script=[Fault_Script;sprintf(blanks(4)+"qcn_out%d = qcn_in%d;",i,i)];
65 end
66 Fault_Script=[Fault_Script;sprintf("}")];
67 Fault_Script=[Fault_Script;sprintf("else{")];
68 Fault_Script=[Fault_Script;sprintf(blanks(4)+"switch(fault_type){")];
69 Fault_Script = [Fault_Script; sprintf("//--------------------------------- ASC for
       each nset ---------------------------------//")];
70 for k=1:n_set
71     l=l+1;
72     Fault_Script=[Fault_Script;sprintf(blanks(4)+"case %d:",l)];
73
74     Fault_Script=[Fault_Script;sprintf(blanks(8)+"qa_out%d = 0;",k)];
75     Fault_Script=[Fault_Script;sprintf(blanks(8)+"qan_out%d = 1;",k)];
76     Fault_Script=[Fault_Script;sprintf(blanks(8)+"qb_out%d = 0;",k)];
77     Fault_Script=[Fault_Script;sprintf(blanks(8)+"qbn_out%d = 1;",k)];
78     Fault_Script=[Fault_Script;sprintf(blanks(8)+"qc_out%d = 0;",k)];
79     Fault_Script=[Fault_Script;sprintf(blanks(8)+"qcn_out%d = 1;",k)];
80
81     for i=1:n_set
82         if i~=k
83            Fault_Script=[Fault_Script;sprintf(blanks(8)+"qa_out%d = qa_in%d;",i,i)];
84            Fault_Script=[Fault_Script;sprintf(blanks(8)+"qan_out%d = qan_in%d;",i,i)];
85            Fault_Script=[Fault_Script;sprintf(blanks(8)+"qb_out%d = qb_in%d;",i,i)];
86            Fault_Script=[Fault_Script;sprintf(blanks(8)+"qbn_out%d = qbn_in%d;",i,i)];
87            Fault_Script=[Fault_Script;sprintf(blanks(8)+"qc_out%d = qc_in%d;",i,i)];
88            Fault_Script=[Fault_Script;sprintf(blanks(8)+"qcn_out%d = qcn_in%d;",i,i)];
89         end
90     end
91     Fault_Script=[Fault_Script;sprintf(blanks(4)+"break;")];
92 end
93 Fault_Script = [Fault_Script; sprintf("//--------------------------------- Open for
       each nset ---------------------------------//")];
94 for k=1:n_set
95     l=l+1;
96     Fault_Script=[Fault_Script;sprintf(blanks(4)+"case %d:",l)];
97
98     Fault_Script=[Fault_Script;sprintf(blanks(8)+"qa_out%d = 0;",k)];
99     Fault_Script=[Fault_Script;sprintf(blanks(8)+"qan_out%d = 0;",k)];
100    Fault_Script=[Fault_Script;sprintf(blanks(8)+"qb_out%d = 0;",k)];
101    Fault_Script=[Fault_Script;sprintf(blanks(8)+"qbn_out%d = 0;",k)];
102    Fault_Script=[Fault_Script;sprintf(blanks(8)+"qc_out%d = 0;",k)];
103    Fault_Script=[Fault_Script;sprintf(blanks(8)+"qcn_out%d = 0;",k)];
104
105    for i=1:n_set
106        if i~=k
107           Fault_Script=[Fault_Script;sprintf(blanks(8)+"qa_out%d = qa_in%d;",i,i)];
108           Fault_Script=[Fault_Script;sprintf(blanks(8)+"qan_out%d = qan_in%d;",i,i)];
109           Fault_Script=[Fault_Script;sprintf(blanks(8)+"qb_out%d = qb_in%d;",i,i)];
110           Fault_Script=[Fault_Script;sprintf(blanks(8)+"qbn_out%d = qbn_in%d;",i,i)];
111           Fault_Script=[Fault_Script;sprintf(blanks(8)+"qc_out%d = qc_in%d;",i,i)];
112           Fault_Script=[Fault_Script;sprintf(blanks(8)+"qcn_out%d = qcn_in%d;",i,i)];
113        end
114    end
115    Fault_Script=[Fault_Script;sprintf(blanks(4)+"break;")];
116 end
```

```
k=1;
Fault_Script = [Fault_Script; sprintf("//-------------------------------- Open A leg first set------------------------------//")];
l=l+1;
Fault_Script=[Fault_Script;sprintf(blanks(4)+"case %d:",l)];

Fault_Script=[Fault_Script;sprintf(blanks(8)+"qa_out%d = 0;",k)];
Fault_Script=[Fault_Script;sprintf(blanks(8)+"qan_out%d = 0;",k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qb_out%d = qb_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qbn_out%d = qbn_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qc_out%d = qc_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qcn_out%d = qcn_in%d;",k,k)];

for i=1:n_set
    if i~=k
       Fault_Script=[Fault_Script;sprintf(blanks(8)+"qa_out%d = qa_in%d;",i,i)];
       Fault_Script=[Fault_Script;sprintf(blanks(8)+"qan_out%d = qan_in%d;",i,i)];
       Fault_Script=[Fault_Script;sprintf(blanks(8)+"qb_out%d = qb_in%d;",i,i)];
       Fault_Script=[Fault_Script;sprintf(blanks(8)+"qbn_out%d = qbn_in%d;",i,i)];
       Fault_Script=[Fault_Script;sprintf(blanks(8)+"qc_out%d = qc_in%d;",i,i)];
       Fault_Script=[Fault_Script;sprintf(blanks(8)+"qcn_out%d = qcn_in%d;",i,i)];
    end
end
Fault_Script=[Fault_Script;sprintf(blanks(4)+"break;")];
Fault_Script = [Fault_Script; sprintf("//-------------------------------- Short A leg first set ------------------------------//")];
l=l+1;
Fault_Script=[Fault_Script;sprintf(blanks(4)+"case %d:",l)];

Fault_Script=[Fault_Script;sprintf(blanks(8)+"qa_out%d = 1;",k)];
Fault_Script=[Fault_Script;sprintf(blanks(8)+"qan_out%d = 1;",k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qb_out%d = qb_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qbn_out%d = qbn_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qc_out%d = qc_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qcn_out%d = qcn_in%d;",k,k)];

for i=1:n_set
    if i~=k
       Fault_Script=[Fault_Script;sprintf(blanks(8)+"qa_out%d = qa_in%d;",i,i)];
       Fault_Script=[Fault_Script;sprintf(blanks(8)+"qan_out%d = qan_in%d;",i,i)];
       Fault_Script=[Fault_Script;sprintf(blanks(8)+"qb_out%d = qb_in%d;",i,i)];
       Fault_Script=[Fault_Script;sprintf(blanks(8)+"qbn_out%d = qbn_in%d;",i,i)];
       Fault_Script=[Fault_Script;sprintf(blanks(8)+"qc_out%d = qc_in%d;",i,i)];
       Fault_Script=[Fault_Script;sprintf(blanks(8)+"qcn_out%d = qcn_in%d;",i,i)];
    end
end
Fault_Script=[Fault_Script;sprintf(blanks(4)+"break;")];

Fault_Script = [Fault_Script; sprintf("//-------------------------------- Open upper switch A leg first set ------------------------------//")];
l=l+1;
Fault_Script=[Fault_Script;sprintf(blanks(4)+"case %d:",l)];

Fault_Script=[Fault_Script;sprintf(blanks(8)+"qa_out%d = 0;",k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qan_out%d = qan_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qb_out%d = qb_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qbn_out%d = qbn_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qc_out%d = qc_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qcn_out%d = qcn_in%d;",k,k)];

for i=1:n_set
    if i~=k
```

```
        Fault_Script=[Fault_Script;sprintf(blanks(8)+"qa_out%d = qa_in%d;",i,i)];
        Fault_Script=[Fault_Script;sprintf(blanks(8)+"qan_out%d = qan_in%d;",i,i)];
        Fault_Script=[Fault_Script;sprintf(blanks(8)+"qb_out%d = qb_in%d;",i,i)];
        Fault_Script=[Fault_Script;sprintf(blanks(8)+"qbn_out%d = qbn_in%d;",i,i)];
        Fault_Script=[Fault_Script;sprintf(blanks(8)+"qc_out%d = qc_in%d;",i,i)];
        Fault_Script=[Fault_Script;sprintf(blanks(8)+"qcn_out%d = qcn_in%d;",i,i)];
    end
end
Fault_Script=[Fault_Script;sprintf(blanks(4)+"break;")];

Fault_Script = [Fault_Script; sprintf("//--------------------------------- Short upper
     switch A leg first set ---------------------------------//")];
l=l+1;
Fault_Script=[Fault_Script;sprintf(blanks(4)+"case %d:",l)];

Fault_Script=[Fault_Script;sprintf(blanks(8)+"qa_out%d = 1;",k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qan_out%d = qan_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qb_out%d = qb_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qbn_out%d = qbn_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qc_out%d = qc_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qcn_out%d = qcn_in%d;",k,k)];
for i=1:n_set
    if i~=k
        Fault_Script=[Fault_Script;sprintf(blanks(8)+"qa_out%d = qa_in%d;",i,i)];
        Fault_Script=[Fault_Script;sprintf(blanks(8)+"qan_out%d = qan_in%d;",i,i)];
        Fault_Script=[Fault_Script;sprintf(blanks(8)+"qb_out%d = qb_in%d;",i,i)];
        Fault_Script=[Fault_Script;sprintf(blanks(8)+"qbn_out%d = qbn_in%d;",i,i)];
        Fault_Script=[Fault_Script;sprintf(blanks(8)+"qc_out%d = qc_in%d;",i,i)];
        Fault_Script=[Fault_Script;sprintf(blanks(8)+"qcn_out%d = qcn_in%d;",i,i)];
    end
end
Fault_Script=[Fault_Script;sprintf(blanks(4)+"break;")];
Fault_Script = [Fault_Script; sprintf("//--------------------------------- Open lower
    switch A leg first set ---------------------------------//")];
l=l+1;
Fault_Script=[Fault_Script;sprintf(blanks(4)+"case %d:",l)];

Fault_Script=[Fault_Script;sprintf(blanks(8)+"qan_out%d = 0;",k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qa_out%d = qa_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qb_out%d = qb_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qbn_out%d = qbn_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qc_out%d = qc_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qcn_out%d = qcn_in%d;",k,k)];

for i=1:n_set
    if i~=k
        Fault_Script=[Fault_Script;sprintf(blanks(8)+"qa_out%d = qa_in%d;",i,i)];
        Fault_Script=[Fault_Script;sprintf(blanks(8)+"qan_out%d = qan_in%d;",i,i)];
        Fault_Script=[Fault_Script;sprintf(blanks(8)+"qb_out%d = qb_in%d;",i,i)];
        Fault_Script=[Fault_Script;sprintf(blanks(8)+"qbn_out%d = qbn_in%d;",i,i)];
        Fault_Script=[Fault_Script;sprintf(blanks(8)+"qc_out%d = qc_in%d;",i,i)];
        Fault_Script=[Fault_Script;sprintf(blanks(8)+"qcn_out%d = qcn_in%d;",i,i)];
    end
end
Fault_Script=[Fault_Script;sprintf(blanks(4)+"break;")];

Fault_Script = [Fault_Script; sprintf("//--------------------------------- Short lower
     switch A leg first set ---------------------------------//")];
l=l+1;
Fault_Script=[Fault_Script;sprintf(blanks(4)+"case %d:",l)];

Fault_Script=[Fault_Script;sprintf(blanks(8)+"qan_out%d = 1;",k)];
```

```
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qa_out%d = qa_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qb_out%d = qb_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qbn_out%d = qbn_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qc_out%d = qc_in%d;",k,k)];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"qcn_out%d = qcn_in%d;",k,k)];
for i=1:n_set
    if i~=k
       Fault_Script=[Fault_Script;sprintf(blanks(8)+"qa_out%d = qa_in%d;",i,i)];
       Fault_Script=[Fault_Script;sprintf(blanks(8)+"qan_out%d = qan_in%d;",i,i)];
       Fault_Script=[Fault_Script;sprintf(blanks(8)+"qb_out%d = qb_in%d;",i,i)];
       Fault_Script=[Fault_Script;sprintf(blanks(8)+"qbn_out%d = qbn_in%d;",i,i)];
       Fault_Script=[Fault_Script;sprintf(blanks(8)+"qc_out%d = qc_in%d;",i,i)];
       Fault_Script=[Fault_Script;sprintf(blanks(8)+"qcn_out%d = qcn_in%d;",i,i)];
    end
end
Fault_Script=[Fault_Script;sprintf(blanks(4)+"break;")];
Fault_Script=[Fault_Script;sprintf(blanks(4)+"}")];
Fault_Script=[Fault_Script;sprintf("}")];

%%
%Print outputs
l=0;
for j=1:n_set
    Fault_Script=[Fault_Script;sprintf("OutputSignal(0,%d)=qa_out%d;",l,j)];
    l=l+1;
end
for j=1:n_set
    Fault_Script=[Fault_Script;sprintf("OutputSignal(0,%d)=qan_out%d;",l,j)];
    l=l+1;
end
for j=1:n_set
    Fault_Script=[Fault_Script;sprintf("OutputSignal(0,%d)=qb_out%d;",l,j)];
    l=l+1;
end
for j=1:n_set
    Fault_Script=[Fault_Script;sprintf("OutputSignal(0,%d)=qbn_out%d;",l,j)];
    l=l+1;
end
for j=1:n_set
    Fault_Script=[Fault_Script;sprintf("OutputSignal(0,%d)=qc_out%d;",l,j)];
    l=l+1;
end
for j=1:n_set
    Fault_Script=[Fault_Script;sprintf("OutputSignal(0,%d)=qcn_out%d;",l,j)];
    l=l+1;
end

fid = fopen([ctrlFolder_path '\Fault_Scripts\fault_script.c'], 'w');
for i = 1:numel(Fault_Script)
    fprintf(fid,'%s\n', Fault_Script{i});
end
fclose(fid);

end
```

Bibliography

[1] Andrei Bojoi. Advanced dynamic model of e-motor for control rapid prototyping. Master thesis in electrical engineering, Politecnico di Torino, 3 2022.

[2] Radu Bojoi, Sandro Rubino, Alberto Tenconi, and Silvio Vaschetto. Multiphase electrical machines and drives: A viable solution for energy generation and transportation electrification. In *2016 International Conference and Exposition on Electrical and Power Engineering (EPE)*, pages 632–639, 2016.

[3] Lorenzo Cifalinò. Study and advanced modelling of multiphase motors with reconfigurable number of phases. Master's degree thesis, Politecnico di Torino, 3 2023. Supervised by Prof. Gianmario Pellegrino, Dr. Simone Ferrari, and ETEL SA Tutor Dr. Alessandro Fasolo.

[4] Simone Ferrari, Paolo Ragazzo, Gaetano Dilevrano, and Gianmario Pellegrino. Flux-map based fea evaluation of synchronous machine efficiency maps. In *2021 IEEE Workshop on Electrical Machines Design, Control and Diagnosis (WEMDCD)*, pages 76–81, 2021.

[5] Plexim GmbH. Implicit model vectorization. PLECS Tutorial, 2021. Available online at `www.plexim.com`.

[6] Plexim GmbH, Zurich, Switzerland. *Multiphase Synchronous Buck Converter*, 2023. PLECS 4.6.1 demo model.

[7] Plexim GmbH. *PLECS User Manual*. Plexim GmbH, 2023. The software PLECS described in this manual is furnished under a license agreement and may be used or copied only under the terms of the license agreement.

[8] Yao Rao, Yuntao Wang, and Wei Wang. A simplified modeling and analysis method for interturn short-circuit fault of permanent magnet synchronous motor. In *2023 26th International Conference on Electrical Machines and Systems (ICEMS)*, pages 4450–4455, 2023.

[9] Sandro Rubino, Radu Bojoi, Davide Cittanti, and Luca Zarri. Decoupled and modular torque control of multi-three-phase induction motor drives. *IEEE Transactions on Industry Applications*, 56(4):3831–3845, 2020.

[10] Sandro Rubino, Obrad Dordevic, Eric Armando, Iustin Radu Bojoi, and Emil Levi. A novel matrix transformation for decoupled control of modular multiphase pmsm drives. *IEEE Transactions on Power Electronics*, 36(7):8088–8101, 2021.

[11] Luisa Tolosano, Eric Armando, Sandro Rubino, Fabio Mandrile, and Radu Bojoi. Modular torque control of multi-three-phase synchronous motor drives. In *2023 IEEE Energy Conversion Congress and Exposition (ECCE)*, pages 4742–4749, 2023.

[12] Anantaram Varatharajan, Dario Brunelli, Simone Ferrari, Paolo Pescetto, and Gianmario Pellegrino. syredrive: Automated sensorless control code generation for synchronous reluctance motor drives. In *2021 IEEE Workshop on Electrical Machines Design, Control and Diagnosis (WEMDCD)*, pages 192–197, 2021.

[13] Yinghui Yang and Georg Möhlenkamp. Short circuit behavior of dual three-phase permanent magnet synchronous motors with different mutual inductance in electric propulsion application. In *2022 24th European Conference on Power Electronics and Applications (EPE'22 ECCE Europe)*, pages 1–10, 2022.

[14] Ivan Zoric, Martin Jones, and Emil Levi. Vector space decomposition algorithm for asymmetrical multiphase machines. In *2017 International Symposium on Power Electronics (Ee)*, pages 1–6, 2017.

[15] İ. Şahin, G. H. Bayazıt, and O. Keysan. A simulink model for the induction machine with an inter-turn short circuit fault. In *2020 International Conference on Electrical Machines (ICEM)*, volume 1, pages 1273–1279, 2020.