UNIVERSITÀ DI PAVIA
FACULTY OF ENGINEERING
DEPARTMENT OF INDUSTRIAL AND INFORMATION ENGINEERING
MASTER'S DEGREE IN INDUSTRIAL AUTOMATION ENGINEERING

MASTER THESIS

LUT-based Modeling of IPMSM for Electric Vehicle Application

Candidate: Mattia Bordonali

Supervisor: Prof. Lorenzo Mantione

Co-supervisors: Prof.ssa Lucia Frosini, Dott. Ing. Gabriele De Boni

A.Y. 2025/2026

For those who believed in me when I did not.

Abstract

Accurately simulating Permanent Magnet Synchronous Motor (PMSM) drives requires motor models that capture magnetic nonlinearities — something classical linear dq-axis models with constant parameters simply cannot do. This thesis presents a complete workflow for bringing Finite Element Method (FEM) motor-characterization data into a Simulink-based Field-Oriented Control (FOC) environment.

The methodology relies on three-dimensional FEM lookup tables (LUTs) defined over d-axis current, q-axis current, and rotor angle. To use these data with the Simscape Electrical FEM-Parameterized PMSM block, they require several processing steps. Every stage is implemented in MATLAB, forming a reproducible FEM-to-Simulink pipeline.

The processed LUTs then feed the derivation of all control-oriented quantities needed to implement the drive model — local inductances, torque constant, effective flux-linkage parameter, and proportional-integral (PI) gains for the current and speed controllers. These quantities come together in a complete, closed-loop Simulink model comprising the LUT-based motor representation, inverter, measurement blocks, cascaded control loops, and test-signal generation.

The final validation is focused on speed tracking. The reported results include a speed step test and three WLTP-based operating segments corresponding to the Medium-, High-, and Extra-High-speed phases. The WLTP profile was examined by separate simulations because a full continuous run exceeded the available computational and memory limits. The results show that the LUT-based model can reproduce stable closed-loop behavior over progressively more demanding reference-speed conditions, while highlighting the effect of control and operating constraints in the upper speed region.

Overall, the work shows that FEM-based non-linear motor models can be integrated inside a structured control-oriented simulation framework, providing higher electromagnetic fidelity than simplified linear analytical models.

Index

List of Symbols and Abbreviations

Symbol / Abbreviation - Definition:

- PMSM - Permanent Magnet Synchronous Motor
- IPMSM - Interior Permanent Magnet Synchronous Motor
- SPMSM - Surface Permanent Magnet Synchronous Motor
- SynRM - Synchronous Reluctance Motor
- FOC - Field-Oriented Control
- FEM - Finite Element Method
- LUT - Lookup Table
- MTPA - Maximum Torque Per Ampere
- MTPV - Maximum Torque Per Voltage
- PWM - Pulse Width Modulation
- WLTP - Worldwide harmonized Light vehicles Test Procedure
- PI - Proportional-Integral
- dq - Direct-Quadrature rotating reference frame
- abc - Three-phase stationary reference frame
- αβ - Two-phase stationary orthogonal reference frame
- i_d, iq - D- and Q-axis current components (A)
- i_d^*, i_q^* - D- and Q-axis current references (A)
- i_a, i_b, i_c - Phase currents (A)
- v_d, v_q - D- and Q-axis voltage components (V)
- v_d^*, v_q^* - D- and Q-axis voltage references (V)
- v_a, v_b, v_c - Phase voltages (V)
- ψ_d, ψ_q - D- and Q-axis flux linkages (Wb)

- ψ_a, ψ_b, ψ_c - Phase flux linkages (Wb)
- Φ_M - Permanent magnet flux linkage (Wb)
- Φ - Effective flux-linkage parameter used in the VelocityController block (Wb)
- L_d, L_q - D- and Q-axis inductances (H)
- $L_{d,stat}$, $L_{q,stat}$ - Static d-axis and q-axis inductances (H)
- $L_{d,dyn}$, $L_{q,dyn}$ - Dynamic d-axis and q-axis inductances (H)
- R_s - Stator phase resistance (Ω)
- V_{dc} - DC bus voltage (V)
- V_{max} - Maximum inverter voltage (V)
- ω_e - Electrical angular velocity (rad/s)
- ω_m - Mechanical angular velocity (rad/s)
- $\omega_{m,ref}$ - Mechanical speed reference (rad/s)
- ω_{base} - Base speed (rad/s or rpm)
- θ_e - Electrical rotor angle (rad or deg)
- θ_m - Mechanical rotor angle (rad or deg)
- p - Number of pole pairs
- 2p - Total number of poles
- T_e - Electromagnetic torque (Nm)
- T_e^* - Electromagnetic torque reference (Nm)
- T_m^* - Torque command generated by the speed controller (Nm)
- T_r - Resistant load torque applied to the plant (Nm)
- K_t - Torque constant (Nm/A)
- J - Rotor moment of inertia (kg·m²)
- D_m - Viscous damping coefficient (N·m·s/rad)
- K_p, K_i - Proportional and integral controller gains
- $K_{p,id}$, $K_{i,id}$ - D-axis current-controller gains

- $K_{p,iq}$, $K_{i,iq}$ - Q-axis current-controller gains
- $K_{p,s}$, $K_{i,s}$ - Speed-controller gains
- ω_c - Current-loop bandwidth (rad/s)
- ω_n - Speed-loop natural frequency (rad/s)
- ζ - Damping ratio
- T_s - Simulation time step (s)
- f_c - Switching frequency (Hz)
- nId, nIq, nTh - Number of LUT breakpoints in id, iq, and θ axes
- v - Vehicle speed (m/s)
- r_{wheel} - Wheel radius (m)
- G - Transmission ratio (-)
- μ - Rolling-resistance coefficient (-)
- m - Vehicle mass (kg)
- g - Gravitational acceleration (m/s^2)
- ρ - Air density (kg/m^3)
- C_d - Aerodynamic drag coefficient (-)
- A_r - Vehicle frontal area (m^2)
- F_{roll} - Rolling-resistance force (N)
- F_{aero} - Aerodynamic drag force (N)
- F_{res} - Total resistant force (N)

List of Figures

List of Tables

List of Equations

(24) - High-speed approximation of the voltage constraint.

(25) - d-axis current reference LUT for flux weakening.

(26) - q-axis current reference LUT for flux weakening.

(27) - Relationship between electrical and mechanical rotor angle used in FEM characterisation.

(28) - Static d-axis inductance.

(29) - Static q-axis inductance.

(30) - Dynamic d-axis inductance.

(31) - Dynamic q-axis inductance.

(32) - Inductance Jacobian matrix including cross-saturation terms.

(33) - Electrical period expressed in mechanical degrees.

(34) - Angular span of the imported dataset.

(35) - Ratio used to determine whether theta tiling is required.

(36) - Cosine-weighted blending coefficient for angular junction smoothing.

(37) - Blended value used in the junction blending stage.

(38) - Equivalent stator current magnitude at the selected operating point.

(39) - Torque-constrained minimum-current condition used to define the MTPA operating point.

(40) - Local differential d-axis inductance at the selected operating point.

(41) - Local differential q-axis inductance at the selected operating point.

(42) - Local torque constant defined as the derivative of torque with respect to q-axis current.

(43) - Rotor-angle-averaged electromagnetic torque map.

(44) - Finite-difference approximation of the local torque constant along the MTPA trajectory.

(45) - Equivalent flux-linkage parameter used for control.

(46) - d-axis current-loop transfer function.

(47) - q-axis current-loop transfer function.

(48) - d-axis PI current controller.

(49) - q-axis PI current controller.

(50) - d-axis proportional current-controller gain.

(51) - d-axis integral current-controller gain.

(52) - q-axis proportional current-controller gain.

(53) - q-axis integral current-controller gain.

(54) - Mechanical dynamics of the motor-drive system.

(55) - Local torque approximation used in speed-loop design.

(56) - Transfer function from q-axis current to mechanical speed.

(57) - PI speed controller.

(58) - Proportional gain of the speed controller.

(59) - Integral gain of the speed controller.

(60) - Local electromechanical term $\Phi + (L_d - L_q)i_d$ used in the torque-to-current conversion of the speed loop.

(61) - q-axis current reference derived from the torque reference using the local machine model.

(62) - Conversion from vehicle speed to motor mechanical speed reference.

(63) - Rolling-resistance force.

(64) - Aerodynamic drag force.

(65) - Total resistant force.

(66) - Equivalent motor-side resistant torque.

Chapter 1 - Introduction

1.1 Motivation and Industrial Context

Over the past decade, electric vehicles have become an increasingly important part of the automotive industry. This has been driven by stricter emissions regulations, lower power-electronics costs, and growing interest in energy efficiency [1], [2]. Permanent Magnet Synchronous Motors (PMSMs) and synchronous reluctance machines are widely used in EV traction applications because of their high torque density, wide speed range, and good efficiency [2], [3].

To develop and tune the control systems for these drives, accurate simulation models are needed. The classical approach uses linear dq-frame motor models with constant parameters. These models are simple and well understood, but they do not capture the non-linear magnetic traits of real machines, which become significant as new motor designs are developed to achieve higher performance [4], [5]. This limits their usefulness for accurate control design and system validation.

A more accurate alternative is to use data from Finite Element Method (FEM) simulations, which model the machine's electromagnetic behavior in detail. The resulting data, flux linkages, torque, and inductances as functions of current and rotor position, can be stored in lookup tables (LUTs) and can be used directly in closed-loop Simulink simulations [4], [6], [7]. This approach keeps the physical correctness of FEM while remaining compatible with real-time systems, thanks to a lower computational cost.

However, using FEM data in simulation is not straightforward. The raw output needs to be carefully processed before it can be loaded into a simulation block: axes must be monotonically ordered, missing values must be filled, and angular coverage must span a full electrical period. This processing is rarely documented in full in the literature, and no standard pipeline exists.

This thesis presents a complete pipeline to satisfy these requirements, from raw FEM data to a working closed-loop FOC simulation in Simulink, along with the associated Simulink implementation to validate the control results. The overall workflow is shown in Figure 1.1.

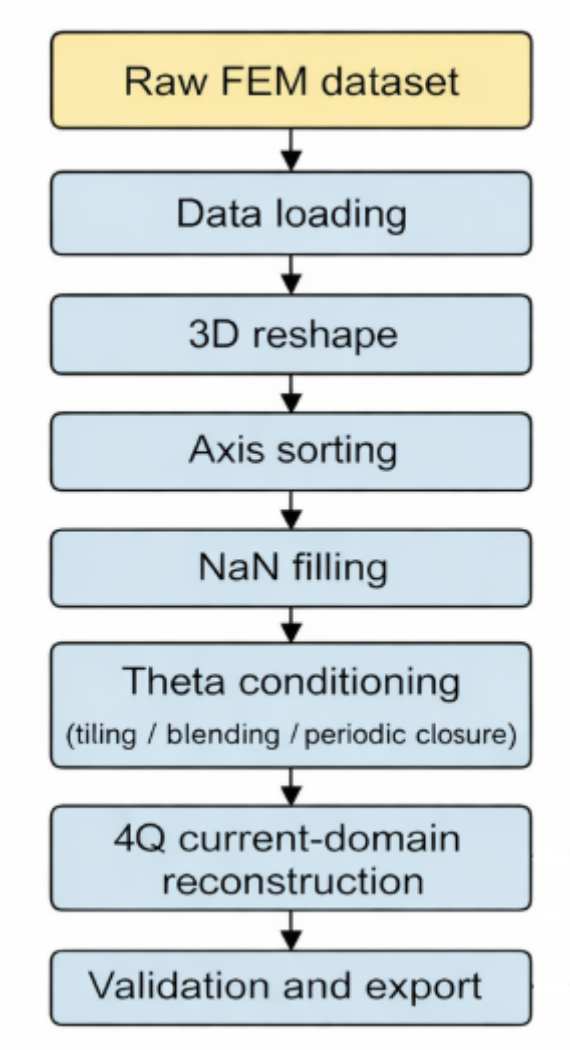

Figure 1.1 - From FEM simulation to Simulink FOC: overview of the proposed workflow.

1.2 State of the Art

The use of FEM-derived lookup tables in drive simulation has been studied since the late 1990s. In [4], it was shown that FEM-based models of saturated IPMSMs are more accurate than linear models, especially for predicting Maximum Torque Per Ampere (MTPA) trajectories and flux-weakening limits. In [5], this was extended to include cross-saturation effects, showing that the coupling between the d- and the q-axis fluxes moves the MTPA operating point away from standard analytical calculations.

Methods for extracting control parameters from FEM maps have been proposed by several authors [8], [9]. On the simulation side, the Simscape Electrical FEM-Parametrised PMSM

block [9] provides a standard interface for LUT-based motor models in Simulink but requires strict input data requirements that are not fully documented.

MTPA strategies for non-linear machines have progressively moved from analytical formulas [10], [11] to numerically derived trajectories obtained from FEM torque maps [12], [13]. Similarly, Flux-weakening strategies have evolved, using more often offline optimization methods [14], [15], [16].

Despite this body of work, a complete and reproducible preprocessing pipeline that covers reshaping, NaN filling, angular tiling, four-quadrant reconstruction, and angle periodicity has not yet been published as a standalone contribution. This thesis intends to address that gap.

1.3 Linear Motor Models Limitations

The classical dq-frame PMSM model is built on three assumptions [17], [18]:

1. that the *d*- and *q*-axis inductances (*Ld*, *Lq*) are constant scalar quantities,
2. that the permanent magnet flux linkage *ΦM* is position-independent
3. that *d*- and *q*-axis flux paths are magnetically decoupled.

Under these assumptions, the *dq*-frame stator voltage equations take the standard linear form:

$$v_d = R_s i_d + L_d \frac{di_d}{dt} - \omega_e L_q i_q \qquad (1)$$

$$v_q = R_s i_q + L_q \frac{di_q}{dt} - \omega_e (L_d i_d + \phi_M) \qquad (2)$$

where R_s is the stator resistance and ω_e is the electrical angular speed.

In practice, none of these assumptions holds for a saturated or saliency-optimised machine [4], [5]: Magnetic saturation causes the inductances to decrease at high current levels, cross-saturation means the flux on one axis depends on the current on the other, and the variation of flux linkage with rotor position produces a torque ripple component that the linear model cannot represent [19].

For one of the motor datasets considered in this work, these effects are particularly relevant, since torque is produced mainly using magnetic saliency. The reasons why a small artificial permanent-magnet flux may still appear in the FEM-based representation are discussed later

in the thesis. More generally, however, a linear model is not sufficient for precise simulation of the non-linear machine behavior considered in this work, which motivates the LUT-based approach developed here.

1.4 Research Objectives

The objectives of this thesis are to:

1. Develop a MATLAB preprocessing pipeline to convert raw FEM data into Simscape-compatible lookup tables.
2. Compute all control parameters directly from the imported data.
3. Implement a complete FOC architecture in Simulink, utilizing the LUT-based motor model.
4. Validate the framework and its drawbacks by simulating multiple scenarios and studying the response speed.

Chapter 2 - Theoretical Background

2.1 PMSM Operating Principles

A Permanent Magnet Synchronous Motor (PMSM) is a three-phase AC machine in which the rotor magnetic field is generated by permanent magnets. The stator carries a three-phase winding that, when excited with balanced AC currents, produces a rotating magnetic field. At steady state, the rotor follows this field exactly, with no slip [17], [18].

The synchronous electrical speed is related to the mechanical rotor speed by:

$$\omega_e = p\,\omega_m \quad (3)$$

where p is the number of pole pairs, and ω_m is the mechanical angular speed.

In the three-phase stationary reference frame, the stator voltage equations are:

$$v_a = R_s\, i_a + \frac{d\psi_a}{dt} \quad (4)$$

$$v_b = R_s\, i_b + \frac{d\psi_b}{dt} \quad (5)$$

$$v_c = R_s\, i_c + \frac{d\psi_c}{dt} \quad (6)$$

where R_s is the stator resistance and ψ_a,ψ_b,ψ_c are the phase flux linkages. Because the inductances in this frame vary with rotor position, these equations are not convenient for control design. The standard solution is to transform them into a rotating reference frame fixed to the rotor, as described in Section 2.2.

In the rotating dq frame, the voltage equations become:

$$v_d = R_s\, i_d + \frac{d\psi_d}{dt} - \omega_e \psi_q \quad (7)$$

$$v_q = R_s\, i_q + \frac{d\psi_q}{dt} - \omega_e \psi_d \quad (8)$$

The terms $\omega_e \psi_q$ and $\omega_e \psi_d$ are cross-coupling terms that arise from the rotation of the reference frame. They play an important role in the control design and are discussed in Section 2.3.

The total electromagnetic torque of a PMSM in the dq reference frame is given by [17], [18]:

$$T_e = \frac{3}{2} p(\psi_d i_q - \psi_q i_d) \qquad (9)$$

For a linear PMSM model, where $\psi_d = L_D i_d + \phi_M$ and $\psi_q = L_q i_q$, this is becomes:

$$T_e = \frac{3}{2} p \left(\phi_M i_q + (L_d - L_q) i_d i_q \right) \qquad (10)$$

This expression shows two separate contributions: The first term, $\frac{3}{2} p \, \phi_M \, i_q$, is the magnet torque, which depends on the permanent magnet flux. The second term, $\frac{3}{2} p \, (L_d - L_q) \, i_d i_q$ is the reluctance torque, which is caused by the difference between L_d and L_q.

The relative importance of these two terms depends on the machine type, as illustrated in Figure 2.1:

- In a surface PMSM (SPMSM), $L_d \approx L_q$, so torque is dominated by the magnet term.
- In an interior PMSM (IPMSM), $L_d \neq L_q$, and both terms contribute.
- In a synchronous reluctance machine (SynRM), $\phi_M \approx 0$ and torque is produced entirely by saliency.

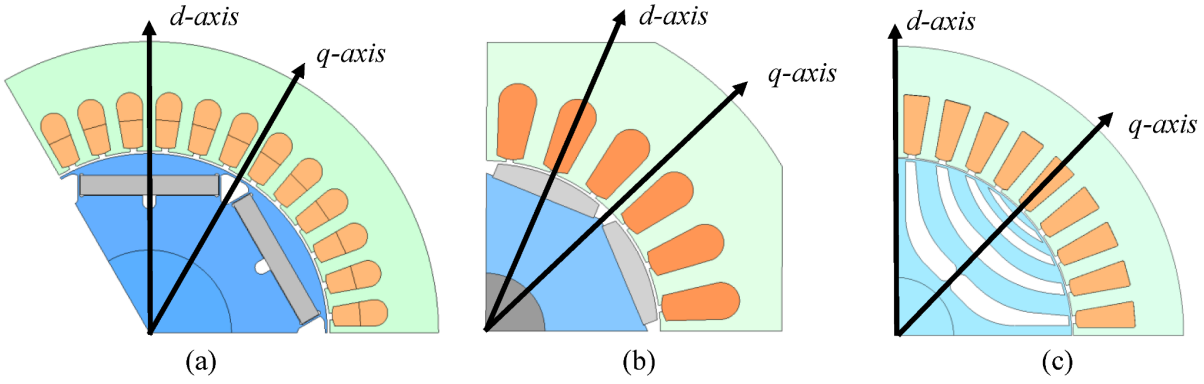

Figure 2.1 - Partial rotor cross-sections of (a) IPMSM, (b) SPMSM, and (c) SynRM. Adapted from [20].

In practice, the inductances L_d and L_q are not constant. Magnetic saturation and cross-coupling cause them to vary with the operating point. For this reason, the flux linkages cannot in general be written as linear functions of the currents, and the FEM-derived lookup tables described in Chapter 4 are used instead:

$$\psi_d = \psi_d(i_d, i_q, \theta_e) \qquad (11)$$

$$\psi_q = \psi_q(i_d, i_q, \theta_e) \qquad (12)$$

2.2 DQ Reference Frame Transformation

The analysis and control of three-phase AC machines is greatly simplified by transforming the natural three-phase (abc) reference frame into a rotating two-phase reference frame fixed to the rotor. This operation converts the three-phase quantities into a two-axis rotating frame that follows the rotor. In this frame, the machine equations have constant coefficients at steady state, which makes them optimal for control design [21].

The transformation is done in two steps. The first step is the projection of three-phase quantities onto a stationary orthogonal reference frame using the Clarke Transformation (*abc* → *αβ*) [22]:

$$\begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} = \frac{2}{3} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} \qquad (13)$$

Then, the Park transformation rotates the *αβ* frame into the *dq* frame aligned with the rotor [21]:

$$\begin{bmatrix} i_d \\ i_q \end{bmatrix} = \begin{bmatrix} \cos\theta_e & \sin\theta_e \\ -\sin\theta_e & \cos\theta_e \end{bmatrix} \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} \qquad (14)$$

where the electrical rotor angle is defined as:

$$\theta_e = p\,\theta_m \qquad (15)$$

and p is the number of pole pairs.

In the dq frame, i_d controls the flux and i_q controls the torque [17], [18]. This separation is what makes field-oriented control possible.

2.3 Field-Oriented Control Architecture

Field-Oriented Control (FOC) is the standard control strategy for high-performance AC drives. The key idea is to regulate i_d and i_q independently, so that flux and torque can be controlled separately, similar to the control strategy of a DC motor [23], [24].

This is accomplished using a cascaded control structure with two nested loops, shown in Figure 2.2. The outer loop regulates speed and produces a torque current reference i_q*, while the inner loop regulates the dq currents and produces voltage commands v_d*, v_q*.

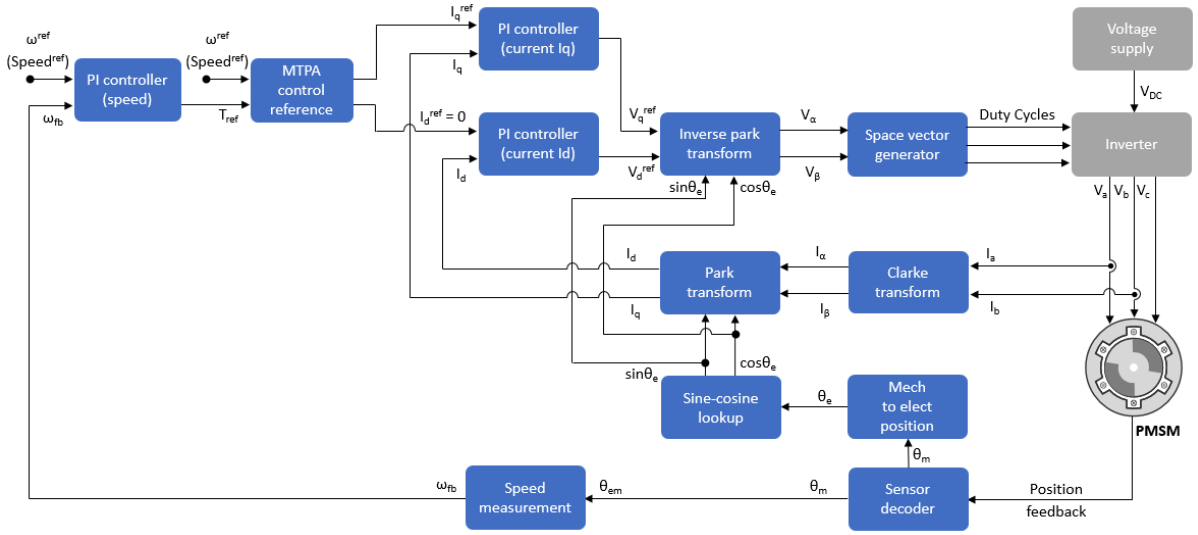

Figure 2.2 - Cascaded FOC architecture: outer speed loop, inner dq current loops, Park/Clarke transformations, and PWM inverter stage. Adapted from [10].

Two independent PI controllers regulate the d- and q-axis currents:

$$e_d = i_d^* - i_d \qquad (16)$$

$$e_q = i_q^* - i_q \qquad (17)$$

so that the controller outputs are the reference voltages v_d* and v_q*.

However, as seen in (7) and (8), the system is inherently coupled through the terms $\omega_e \psi_q$ and $\omega_e \psi_d$. To allow each axis to be controlled independently, these coupling terms are canceled using feedforward compensation [23], [8]:

$$v_d^* = v_{d,PI} - \omega_e \psi_q \qquad (18)$$

$$v_q^* = v_{q,PI} + \omega_e \psi_d \qquad (19)$$

where $v_{d,PI}$ and $v_{q,PI}$ are the outputs of the PI controllers.

With this compensation, each axis behaves approximately as an independent first-order system, which simplifies PI design. The associated gain values are derived in Chapter 5 using parameters extracted from the FEM data at the MTPA operating point.

2.4 Maximum Torque Per Ampere (MTPA) Strategy

For a given operating condition, the controller must generate the requested torque while respecting the machine's constraints. In the constant-torque region, the voltage limit is not yet reached, so the Maximum Torque Per Ampere strategy is used to select the *dq* current vector that produces the required torque while minimising stator current:

$$|i_s| = \sqrt{i_d^2 + i_q^2} \qquad (20)$$

This minimises copper losses for a given torque output and defines the Maximum Torque Per Ampere (MTPA) strategy [18], [19], [20].

The MTPA trajectory is computed numerically from the FEM-derived torque map. For each torque level, the current pair (i_d , i_q) that minimises $|i_s|$ as defined in (20) is:

$$min(i_d^2 + i_q^2)\ Subject\ to\ T_e(i_d, i_q) = T_e^* \qquad (21)$$

This numerical approach accounts for saturation and cross-coupling [10], [11], [13]. The resulting trajectory is stored as a lookup table and used directly in the Simulink model. The specific MTPA operating region for the machine used in this work is identified in Chapter 5, once the FEM data has been processed and the operating point extracted.

The use of a numerically derived MTPA trajectory holds several important implications and will be discussed in Chapters 5 and 7.

2.5 Flux Weakening

Above the motor base speed, the back electromotive force (back-EMF) approaches the inverter voltage limit, reducing the voltage available for current regulation. As a result, the stator voltage constraint becomes [15], [16], [25], [26]:

$$v_d^2 + v_q^2 \leq V_{max}^2 \qquad (22)$$

where Vmax is the maximum voltage produced by the inverter, calculated as:

$$V_{max} = V_{DC}/2 \qquad (23)$$

At high speed, the resistive and derivative terms in (7) and (8) become negligible compared to the back-EMF terms, so the voltage constraint in (22) can be approximated as:

$$\omega_e^2(\psi_d^2 + \psi_q^2) \leq V_{max}^2 (24)$$

This shows that as speed increases, the allowable flux magnitude must decrease [15], [16], [25], [26].

In the proposed Simulink model, flux weakening is implemented using precomputed lookup tables:

$$i_d^* = f(\omega_m, T_e^*) \qquad (25)$$

$$i_q^* = g(\omega_m, T_e^*) \qquad (26)$$

These tables are generated offline by enforcing the constraint in (24) at each operating point and selecting the current pair that maximises torque within the available voltage region [15],

[16]. As speed increases, the operating point moves away from the MTPA locus defined by (21), and moves toward the Maximum Torque Per Voltage (MTPV) region.

Table 2.1 summarises the main machine and drive parameters introduced in Sections 2.1–2.5 and used throughout the remainder of the thesis.

Parameter	Symbol	Meaning	Unit	Note
Stator resistance	R_s	Stator phase resistance	ω	Used in the *dq* voltage equations
d-axis inductance	L_d	d-axis inductance	H	In nonlinear operation, depends on operating point
q-axis inductance	L_q	q-axis inductance	H	In nonlinear operation, depends on operating point
Permanent-magnet flux linkage	Φ_M	Magnetic flux linkage	Wb	Used in the classical PMSM torque expression
Pole pairs	p	Number of pole pairs	–	($\omega_e = p\ \omega_m$)
Number of poles	$2p$	Total number of poles	–	–
Mechanical speed	ω_m	Rotor mechanical angular speed	rad/s	Shaft speed
Electrical speed	ω_e	Electrical angular speed	rad/s	($\omega_e = p\ \omega_m$)
d-axis current	i_d	Flux / magnetisation current component	A	Negative in FW region for the present control convention
q-axis current	i_q	Torque-producing current component	A	Main torque control input
Maximum inverter voltage	V_{max}	Maximum available stator voltage	V	Defines the voltage constraint
Base speed	ω_{base}	Speed at which the voltage limit becomes active	rad/s or rpm	Boundary between MTPA and FW region

Table 2.1 - Summary of the main machine and drive parameters introduced in Chapter 2.

2.6 Role of FEM in Motor Characterization

Electric Machines are commonly characterized using the Finite Element Method (FEM). This is done by solving Maxwell's equations over a discretized model of the motor geometry. FEM computes flux density, flux linkage, and torque with high accuracy [6], [7].

For drive-control applications, the FEM characterization is performed by sweeping the motor over its operating range in (i_d, i_q, θ_e). At each point, the solver computes the flux linkages ψ_d and ψ_q as in (11) and (12), the electromagnetic torque Te as in (9), and the inductance values. These results are stored as multidimensional lookup tables that fully describe the machine's nonlinear electromagnetic behavior [4], [6], [7].

FEM captures several effects that the linear model of (10) misses:

- Magnetic saturation: the non-linear B–H relationship of ferromagnetic materials causes the effective inductances to decrease at high current levels [4], [5].
- Cross-saturation: the flux on one axis depends on the current on the other, shifting the MTPA trajectory defined by (19) away from the anticipated results [5], [11].
- Spatial harmonics: stator slotting and rotor geometry cause regular fluctuations in flux linkage with rotor position, producing torque ripple not visible in (10) [19].

While FEM is computationally expensive and cannot be run in real time, the resulting lookup tables can be evaluated efficiently during the simulation. This is the approach adopted in this work, and it forms the foundation of the pipeline developed in Chapter 4 [4], [6], [7].

Chapter 3 - Motor Characterization via FEM

3.1 FEM Simulation Setup

The electromagnetic characterisation of the motor under study was obtained through magnetostatic finite-element simulations performed using *Altair Flux 2024* [7]. The dataset used in this thesis is not presented as an original modelling contribution of the present work. Rather, it was generated within previous research carried out in the laboratory and is adopted here as the source dataset for the LUT-based preprocessing along with the control-oriented Simulink implementation. A related lookup-table-based reduced-order modelling approach for synchronous motors aimed at digital-twin applications has already been presented by the same research group in previous work [27].

The characterisation was performed on a three-dimensional parametric grid defined by the d-axis peak current i_d, the q-axis peak current i_q, and the mechanical rotor angle θ_m. The dimensions of the dataset are therefore determined directly by the number of samples available along these three axes, as exported from the FEM environment and later loaded into MATLAB.

At each grid point, the magnetostatic solver computes the main electromagnetic quantities required for control-oriented modelling: d-axis and q-axis flux linkages, electromagnetic torque, Joule losses, iron losses, and the full set of static, dynamic, and cross-coupling inductance terms [6], [7]. These data provide the complete electromagnetic description of the machine and form the direct input to the preprocessing pipeline developed in Chapter 4.

The FEM sweep was conducted over three independent variables (i_d,i_q,θ_m), where i_d, i_q are the d- and q-axis peak currents, consistent with the convention later adopted in the MATLAB preprocessing as well as in the Simulink implementation.

The rotor-angle axis covers only part of the full periodic behaviour of the machine. More precisely, the available angular range corresponds to approximately half of the electrical period. Since electrical machines exhibit cyclic behaviour, it is not always necessary to simulate the full revolution directly. A reduced angular domain can be sufficient, provided that the missing part is reconstructed correctly during preprocessing [6], [7].

The relationship between mechanical and electrical angle is given by:

$$\theta_e = p\,\theta_m \qquad (27)$$

This reduction of the simulation domain is standard practice in FEM-based motor characterisation because it lowers computational cost whilst preserving the electromagnetic fidelity needed for modelling and control analysis [6], [7].

For each operating point (i_d,i_q,θ_m), the FEM solver provides:

- flux linkages ψ_d,ψ_q
- electromagnetic torque T_e
- inductance quantities (static and dynamic)
- Joule losses

These outputs form the basis of the multidimensional lookup tables used in subsequent chapters for control-oriented modelling [4], [6], [7].

3.2 Output Data Structure

The FEM characterisation produces a multidimensional dataset that describes how the electromagnetic behaviour of the machine changes over the selected operating range. These data are stored as MATLAB .mat files and are used to build the lookup-table-based motor model adopted in the simulation framework [6], [7].

Each quantity is organised over the discretised grid (i_d, i_q, θ_m). In other words, each exported variable is defined over the same three-dimensional operating domain, where each point relates to a specific combination of d-axis current, q-axis current, and rotor position. This organisation is consistent with FEM-based dq modelling, where flux linkages, torque, and inductances are nonlinear functions of both current and position [4], [6].

The dataset contains the main electromagnetic variables needed for the later modelling stages, including:

- d-axis flux linkage FLUX_D,
- q-axis flux linkage FLUX_Q,
- electromagnetic torque TORQUE,

- d-axis and q-axis dynamic inductances,
- cross-coupling inductance terms,
- static inductance terms

Iron losses and Joule losses are recorded in the FEM dataset, but they are not included in the present simulation model. In this thesis, the focus is on the accurate representation of torque and flux-linkage behaviour for control design. Including iron losses would require an additional loss-model subsystem in Simulink, which is beyond the scope of the present work and is therefore left as a possible future extension.

The static inductance is defined as:

$$L_{d,stat} = \frac{\psi_d}{i_d} \qquad (28)$$

$$L_{q,stat} = \frac{\psi_q}{i_q} \qquad (29)$$

and represents an average relationship between flux linkage and current.

The Dynamic inductance is defined as the local derivative [5], [7]:

$$L_{d,dyn} = \frac{\partial \psi_d}{\partial i_d} \qquad (30)$$

$$L_{q,dyn} = \frac{\partial \psi_q}{\partial i_q} \qquad (31)$$

These quantities are especially important for control because they describe how the machine responds to small current variations around a given operating point.

More generally, in a nonlinear machine the inductance behaviour can be represented through the Jacobian matrix [5], [17]:

$$\mathbf{L} = \begin{bmatrix} \frac{\partial \psi_d}{\partial i_d} & \frac{\partial \psi_d}{\partial i_q} \\ \frac{\partial \psi_q}{\partial i_d} & \frac{\partial \psi_q}{\partial i_q} \end{bmatrix} \qquad (32)$$

which explicitly includes the cross-saturation terms.

From a control point of view, the dynamic inductances are the most relevant because they directly affect current-loop design and the tuning of the PI controllers. The static inductances

remain useful mainly for explaining and visualising how saturation evolves across the operating range.

In addition to the LUT data, the MATLAB file contains the stator phase resistance, end-winding inductance, rotor inertia, and viscous damping coefficient. These parameters are later combined with the lookup tables to build the complete Simulink drive model.

3.3 Flux Linkage, Inductance and Torque Maps

The main results from FEM characterisation are the d-axis flux linkage ψd, q-axis flux linkage ψ_q, and electromagnetic torque T_e. These form the core of the nonlinear motor model in Simulink. Their dependence on (i_d, i_q, θ_m) gives a full picture of the machine's electromagnetic behaviour [4], [6], [7].

The d-axis flux linkage $\psi_d(i_d, i_q, \theta_e)$ varies in a non-linear manner with operating points. At low current levels, its dependence on i_d is close to linear, which corresponds to weak magnetic saturation. As the current increases, however, the slope of the flux-current characteristic decreases because magnetic saturation reduces the effective permeability of the iron [4], [5]

At the same time, ψ_d does not depend only on i_d, it also varies with i_q, which is a direct indication of cross-saturation. In other words, the magnetic condition of one axis is influenced by the current flowing on the other axis [5]. This behaviour is captured naturally by the FEM data but cannot be represented correctly by the classical linear *dq* model [4], [5], [17], [18].

A similar behaviour is observed for the q-axis flux linkage $\psi_q(i_d, i_q, \theta_e)$. It depends mainly on i_q, but it also shows saturation and linkage effects. In the dataset used in this work, the flux maps also indicate a non-negligible magnet-related contribution on the *d* axis. This means that the machine behaviour cannot be described only in terms of current-dependent saliency, since a magnet-flux contribution is also present and has to be taken into account in the final control-oriented interpretation.

Both flux-linkage components also vary with rotor position. This angular dependence comes from slotting, rotor geometry, and spatial harmonics, and it introduces periodic fluctuations in

the electromagnetic quantities [19], [6]. One direct consequence of this behaviour is torque ripple.

The electromagnetic torque $T_e(i_d, i_q, \theta_e)$ is obtained in the FEM environment through standard torque-evaluation methods, such as the Maxwell tensor or the magnetic co-energy approach [6], [7]. Unlike simplified analytical dq models, which generally describe only average torque, the FEM map also captures the variation of torque with rotor position. For a fixed current operating point, the torque is therefore not constant but periodic in m, producing a torque-ripple profile [19], [6].

The average torque over one electrical period can be interpreted as the useful torque delivered by the machine, whereas the deviation from that mean value represents the ripple component. This is important both for performance assessment and for the later control design.

The torque map also provides information about which regions of the (i_d, i_q) plane are more favourable for torque production. For this reason, it is used later to determine the MTPA operating trajectory adopted in the control system [11], [12], [13].

The inductance maps offer a complementary view of the machine behaviour. At low current, the d-axis and q-axis inductances are mainly determined by the geometry of the machine and remain relatively constant. At higher current levels, magnetic saturation reduces their effective values [4], [5]. In addition, the off-diagonal terms of the inductance matrix highlight the presence of cross-saturation, meaning that the electrical dynamics of the two axes are coupled [5]. This aspect is especially relevant for control, because it affects current-loop tuning and contributes to the difference between the real nonlinear machine and its local linear approximation. It also influences the accuracy of optimisation strategies such as MTPA [5], [11].

Taken together, the flux-linkage, torque, and inductance maps provide a detailed nonlinear description of the motor, including magnetic saturation, axis coupling, and position-dependent effects such as torque ripple [4], [5], [19], [6]. This is what makes lookup-table-based modelling attractive in the present context: the model keeps the physical richness of FEM data while remaining usable inside a control-oriented simulation framework.

3.4 Motor Datasets Considered in This Work

Two different motor datasets were used in this work: a synchronous reluctance machine (SynRM) dataset and the 2010 Toyota Prius traction motor dataset. These two datasets were not adopted as alternative case studies, but as complementary sources for the development of a wider and more comprehensive LUT-based modeling workflow

Their combined use is important because the FEM-to-Simulink pipeline described in Chapter 4 includes several preprocessing stages whose relevance depends on the particular numerical characteristics of the source data. In practice, not all conditioning operations become equally visible on every motor dataset. For this reason, using both the SynRM and Prius cases makes it possible to discuss the technique in a more general way, without tying the whole workflow to the peculiarities of a single machine only.

Figure 3.1 shows the finite-element model used for the electromagnetic characterization of the Toyota Prius motor. The FE model represents the machine on a reduced sector, which is a standard choice in finite-element analysis, since the cyclic symmetry of the motor allows the behavior of the full machine to be reconstructed from a smaller portion of the geometry with a significant cut in computational cost.

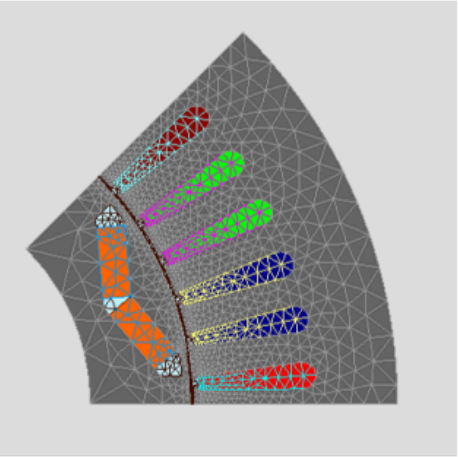

Figure 3.1 - FE Model of the 2010 Toyota Prius traction motor.

The Prius motor is characterized by a **rated DC voltage of 400 V**, a **rated RMS current of 100 A**, a **rated torque of 177 Nm**, a **base speed of 2050 rpm**, and a **rated power of 38 kW**.

The machine has **8 poles**, corresponding to **4 pole pairs**. These quantities define the nominal operating region of the final application-oriented case and provide the main reference for the Simulink implementation.

The **SynRM dataset** is used in parallel as a second reference case to support the development of the preprocessing methodology. In this case, torque production is mainly associated with magnetic saliency, and the dataset is particularly useful for discussing preprocessing stages that are not equally evident in the Prius case. Even when a complete nameplate-style description is not reported here, this second dataset remains methodologically important because it contributes to the construction of a more general LUT-conditioning procedure.

Taken together, the two datasets allow the present work to remain both **application-oriented** and **methodologically general**. The Prius motor provides the main engineering reference for the implementation stage, while the SynRM dataset helps cover additional preprocessing situations that are discussed explicitly in Chapter 4.

Chapter 4 - FEM-to-Simulink LUT Pipeline

4.1 Pipeline Overview and Design Rationale

The raw output of a FEM characterization is not directly usable in a Simulink simulation. The Simscape Electrical FEM-Parametrised PMSM block [9] enforces specific requirements on the input data: breakpoint axes must be strictly monotonically increasing, no NaN values are allowed, the angular axis must span a complete electrical period, and the first and last angular slices must be identical to enforce periodicity. None of these conditions is guaranteed by the FEM output as exported.

The preprocessing pipeline developed in this work converts the raw FEM data into a simulation-ready dataset through a sequence of eight stages. Each stage addresses a specific incompatibility between the raw data and the Simulink block requirements.

The complete preprocessing workflow consists of the following main stages:

1. Data loading - import of raw FEM arrays from the .mat file.
2. Reshape - conversion of flat arrays into structured 3D datasets indexed by (id,iq,m).
3. Sorting - enforcement of strictly monotonic breakpoint vectors.
4. NaN filling - reconstruction of missing data via scattered interpolation.
5. Theta tiling - extension of angular coverage to the full electrical period.
6. Junction blending - smoothing of discontinuities along tiling boundaries.
7. Four-quadrant reconstruction - extension of the dataset to the full (id,iq) domain.
8. Periodic closure and export - enforcement of the exact periodicity and the assignment to the Simulink workspace.

These stages collectively convert raw FEM outputs into a numerically well-conditioned dataset compatible with online simulation requirements.

The pipeline operates uniformly on all quantities, including flux linkages, torque and inductances, guaranteeing that all lookup tables remain mutually consistent throughout the processing chain. Such consistency is critical, as any mismatch between datasets would

introduce non-physical behaviour in the simulation, such as non-physical torque responses, instability in closed-loop simulation, and compilation errors.

The pipeline is methodological in scope and is presented using the two motor datasets introduced in Section 3.4. The Toyota Prius dataset is the final application-oriented reference case used for the Simulink implementation, while the synchronous reluctance motor dataset is used where needed to illustrate preprocessing stages that are not equally visible in the Prius case. As a result, not every conditioning step is activated in the same way for both datasets, even though the overall workflow remains the same.

This point is especially relevant for stages such as NaN filling, theta tiling, and junction blending. In the final Prius workflow, some of these operations reduce to verification steps only, whereas in the complementary reluctance-motor case, they provide a clearer illustration of the role of the corresponding conditioning stage. Presenting both situations within the same chapter makes the preprocessing methodology more general and more transparent.

The result is a set of lookup tables that preserve the original non-linear electromagnetic characteristics, while satisfying the numerical constraints of real-time control simulation environments [4], [6], [9].

In addition to the angular conditioning steps, if required, the final implementation also reconstructs a full four-quadrant current-domain representation from the available source FEM dataset. The raw source map is first interpreted on its native current domain, and the control-oriented MTPA trajectory is identified there. The lookup tables are then extended to the remaining current quadrants through symmetry-based reconstruction rules applied consistently to flux, torque, and inductance quantities. This step is necessary because the final Simulink model runs on a full i_d-i_q domain, while the source FEM dataset does not directly provide all four quadrants.

To make the effect of each conditioning stage easier to follow, the main preprocessing steps are illustrated in the following sections through representative plots of the data. Where relevant, the figures explicitly refer to the dataset on which the corresponding effect is most clearly visible.

4.2 Data Loading and Axis Vector Extraction

The first step of the pipeline is to clear the MATLAB base workspace and the Simulink model workspace to eliminate any variables from previous simulations.

After workspace initialization, the FEM dataset is loaded from the .mat file containing the characterisation results. This dataset includes the multidimensional arrays representing the electromagnetic quantities, together with the breakpoint vectors that define the discretized operating domain.

The dataset is structured along three independent axes:

- i_d, i_q are the d- and q-axis peak currents
- θ_m is the mechanical rotor angle

The corresponding axis vectors, ID_PEAK, IQ_PEAK, and ANGPOS_ROTOR_DEG, are extracted as row vectors from the lookup-table dataset. From these, the dimensions of the three-dimensional operating grid are explicitly defined.

This dimensional definition establishes the domain over which all FEM quantities are represented and makes sure that each variable can later be consistently reshaped into a three-dimensional LUT.

In the MATLAB implementation, each source dataset is first identified on its native current domain before any four-quadrant reconstruction is performed. This distinction is important because the MTPA search is carried out on the available source domain, whereas the final lookup tables exported to Simulink are later extended to a full four-quadrant representation.

Before any data transformation is applied, the pipeline performs a set of integrity checks to verify the structural validity of the loaded data. These checks verify that the breakpoint vectors and the electromagnetic arrays are dimensionally consistent, and that the datasets considered provide a complete sampling of the source current domain used for the initial control-oriented reconstruction.

4.3 Three-Dimensional Reshape Strategy

The FEM datasets, as loaded from the .mat file, are stored as one-dimensional arrays, but to construct the required lookup tables for Simulink, these arrays must first be reshaped into structured three-dimensional matrices [9].

The reshape operation is applied uniformly to all FEM quantities and restores the physical structure of the dataset. In the adopted convention, the three dimensions correspond to:

- the d-axis current i_d,
- the q-axis current i_q,
- the mechanical rotor angle θ_m.

In MATLAB, this operation is implemented by reshaping each flat vector into an array of dimensions $nid, niq, n\theta$. The same procedure is used for the d-axis and q-axis flux linkages, the torque map, and the available inductance maps.

An important aspect of this step is that the reshape operation implicitly assumes a consistent ordering in the exported FEM data. The correctness of the reconstructed three-dimensional arrays therefore depends on the order in which the original operating grid was linearised before export. If this assumption is wrong, the resulting LUTs may appear numerically valid while actually representing incorrect operating points.

For this reason, the reshape step is not treated as a purely formal data-formatting operation. Its correctness is checked against known physical properties of the considered machine and of the expected operating region. In particular:

- at fixed i_q, the d-axis flux linkage is expected to vary monotonically with i_d in the weakly saturated region [4], [5];
- the torque map should be consistent with the control-oriented operating region identified for the considered dataset;
- the reconstructed surfaces should vary smoothly across neighbouring operating points, without artificial discontinuities.

The following figures show the reshaped surfaces of the Toyota Prius dataset at a fixed rotor-angle slice. In particular, Figure 4.1 reports the reshaped d-axis flux linkage, q-axis flux linkage, and torque maps, while Figure 4.2 shows the corresponding inductance maps.

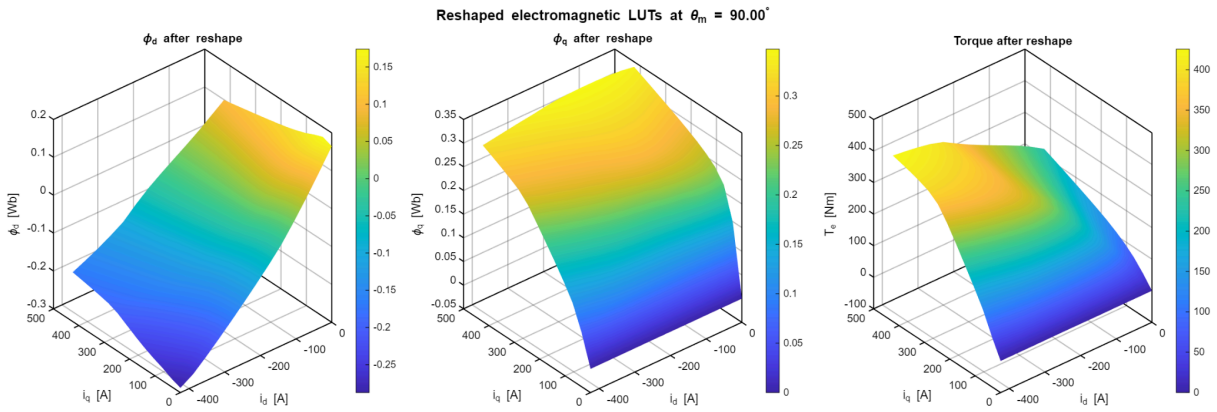

Figure 4.1 - Reshaped electromagnetic LUTs at a fixed rotor-angle slice: d-axis flux linkage, q-axis flux linkage, and electromagnetic torque.

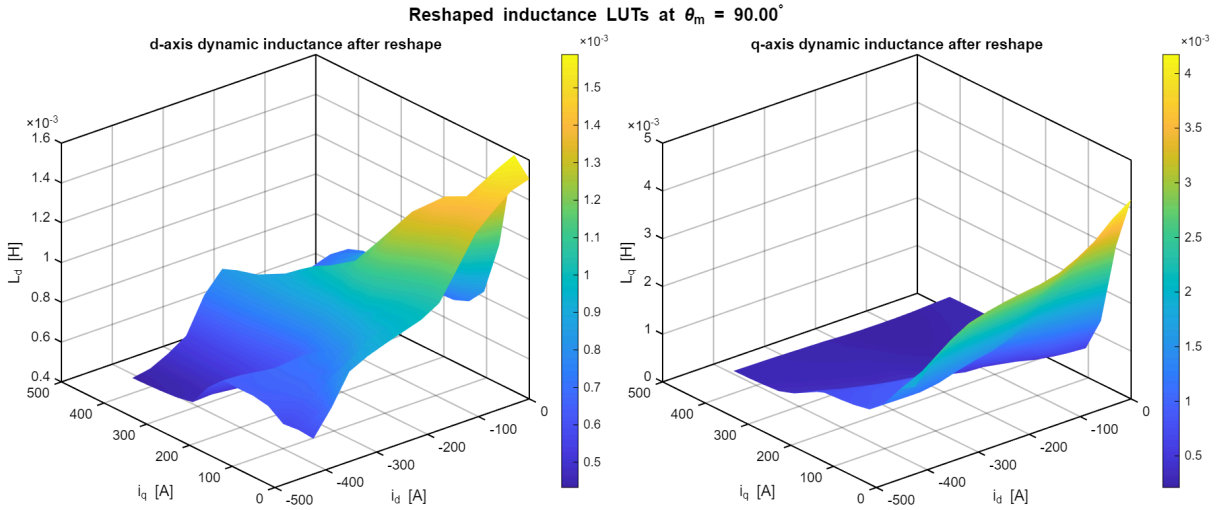

Figure 4.2 - Reshaped inductance LUTs at a fixed rotor-angle slice: representative d-axis and q-axis inductance maps.

The purpose of these plots is not yet to show the final conditioned dataset, but to verify that the reshaped array already shows a physically meaningful structure.

The reshape operation is applied identically to all FEM-derived quantities, including flux linkages, torque, inductances, and loss maps. Keeping consistent indexing across all arrays is essential, since these variables are used jointly in the simulation model. Any inconsistency at this stage would lead to discordant physical quantities and, therefore, to non-physical system behavior.

4.4 Breakpoint Sorting and Monotone Behaviour Enforcement

The selected Simulink block requires that all the breakpoint axes are strictly increasing to ensure correct interpolation [9]. For this reason, the pipeline enforces the monotonicity by sorting each axis vector in ascending order.

In MATLAB, this operation is applied to the d-axis current vector, the q-axis current vector, and the rotor-angle vector. The corresponding permutation indices are then used to consistently reorder all lookup-table arrays across the three dimensions. Since the sorting operation must be applied not only to the breakpoint vectors, but also to the associated LUT data, each value remains linked to the correct operating point after reindexing.

Another important requirement of the Simulink interpolation logic is strict monotonicity. This means that duplicate breakpoint values are not allowed, since they may lead to undefined interpolation behavior and possible runtime errors [9]. After sorting, the pipeline therefore verifies that all breakpoint vectors are strictly increasing and that the reordered arrays remain dimensionally consistent with the updated axes.

This step is not a simple formatting operation. It is necessary to ensure that interpolation is well-defined over the entire operating domain. If the axes are not ordered correctly, the resulting LUT evaluation may produce incorrect interpolation weights, surface discontinuities, and non-physical flux or torque responses in simulation.

In the present dataset, the breakpoints were already ordered correctly, so this step did not produce a substantial visible change in the LUT surfaces. Its role was therefore mainly to guarantee strict monotonicity and consistent reindexing before the following steps.

4.5 NaN Filling via Scattered Interpolation

The results of an FEM characterization might produce incomplete datasets because current limits, convergence difficulties in highly saturated regions, or solver instability can prevent the computation of valid results at some operating points [7], [8]. In the exported lookup tables, these missing values appear as undefined entries (NaN).

The selected Simulink Motor block requires complete datasets, and any undefined entry would propagate through interpolation, potentially leading to simulation failure or non-physical outputs [9].

For this reason, the preprocessing pipeline includes a dedicated NaN-filling stage based on slice-by-slice interpolation over the (id,iq) plane at a fixed rotor angle. In the general implementation, missing values can be reconstructed using a scattered interpolation procedure with natural-neighbor interpolation, with nearest-neighbor extrapolation applied only at the boundaries. In MATLAB, this can be implemented through the *scatteredInterpolant* function [28].

In the Toyota Prius dataset used, no NaN values were found. This stage was consequently used only to confirm that the processed tables were already complete before the subsequent conditioning steps.

To illustrate the role of this operation within a more general preprocessing workflow, Figure 4.3 is instead reported using the synchronous reluctance motor dataset, in which undefined points were effectively present in the raw FEM export. This second case makes the effect of the NaN-filling stage more clearly visible by showing the dataset before and after the slice-by-slice scattered interpolation procedure.

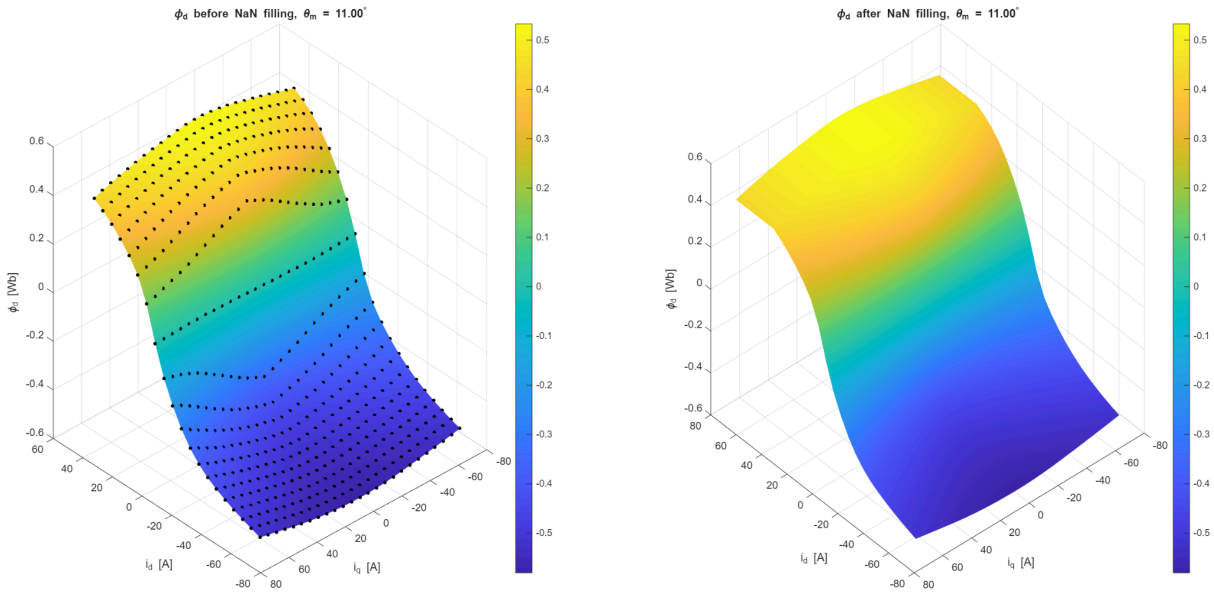

Figure 4.3 - Example of NaN filling on the synchronous reluctance motor dataset: LUT surface before and after scattered interpolation at a fixed rotor-angle slice.

Even though this operation was not activated in the final dataset, it remains part of the general preprocessing logic, since it would be required whenever missing entries are present in raw FEM exports, to avoid errors while compiling the Simulink model.

At the end of this stage, the lookup tables are either confirmed to be already complete, as in the Prius case, or reconstructed into a fully populated form, as illustrated by the synchronous reluctance motor example, so that they are ready for the following angular-conditioning stages.

4.6 Theta Tiling to Cover the Full Electrical Period

The FEM dataset used in this work spans only the angular range exported by the characterization process, which may be smaller than the full electrical period of the machine. In general, this is not unusual in FEM-based motor analysis, since symmetry can be used to reduce the simulated angular domain and therefore lower the computational cost [6], [7].

For use in the selected Simulink block, however, the lookup tables must be defined over a complete electrical period and must remain consistent with the periodic nature of the electromagnetic quantities [9]. For this reason, the preprocessing pipeline includes a theta-tiling stage that can extend the available angular domain whenever the source FEM dataset covers only a fraction of the required period.

The electrical period expressed in mechanical degrees is evaluated as:

$$\theta_{per} = \frac{360}{p} \qquad (33)$$

where p is the number of pole pairs.

The angular span of the imported dataset is then computed as:

$$\theta_{span} = \theta_{m,end} - \theta_{m,start} \qquad (34)$$

and the ratio:

$$r = \frac{\theta_{per}}{\theta_{span}} \qquad (35)$$

is used to determine if the angular range covers only a fraction of the motor's electrical period. In that case, the dataset can be extended by copying the LUTs along the angular

dimension to reconstruct the missing angular sector, while preserving the continuity of the original data and avoiding duplicated breakpoints.

In the Toyota Prius dataset used as the final application-oriented reference case, this operation was not required, since p = 4 and $\theta_{span} = 90°$.

To illustrate the role of this operation within the general preprocessing methodology, Figure 4.4 is instead reported using the synchronous reluctance motor dataset, in which angular extension was required to obtain a dataset consistent with the full electrical-period representation expected by the following conditioning stages. This example makes the effect of the theta-tiling operation more clearly visible than in the Prius case.

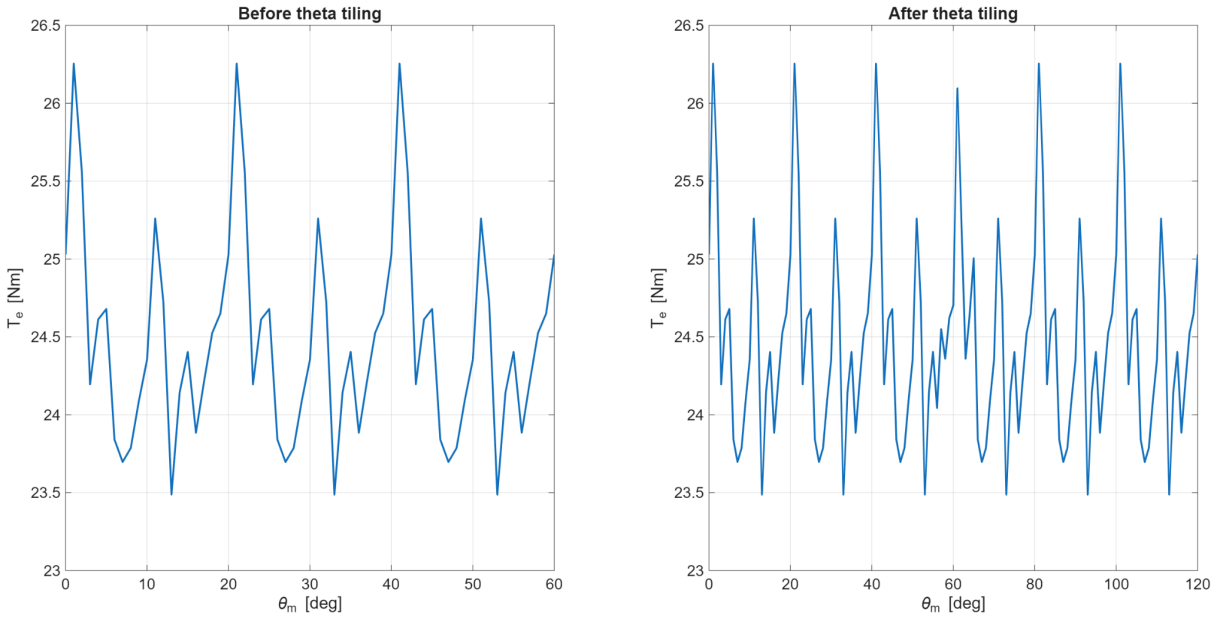

Figure 4.4 - Theta tiling illustrated on the synchronous reluctance motor dataset: angular extension of the LUTs to cover the full electrical period.

For the Prius dataset, the role of this stage was therefore limited to verification, confirming that the available angular coverage was already compatible with the following conditioning steps. More generally, however, theta tiling remains an integral part of the preprocessing logic, since it becomes necessary whenever the raw FEM export covers only a fraction of the electrical period, as illustrated by the synchronous reluctance motor case.

At the end of this step, the angular domain was confirmed to be suitable for the subsequent periodic-consistency operations and for final export to the Simulink model.

4.7 Junction Blending

When the tiling operation is required, the resulting dataset may cover the full electrical period, but small discontinuities may occur at the junction between the original segment and the appended one. These discontinuities are generally caused by discretization effects or by a source angular range that does not end exactly at the ideal symmetry boundary.

Even when the values on the two sides of the junction are close, their local slope with respect to rotor angle may still differ. If left untreated, this may introduce artificial irregularities in the interpolated LUT surfaces and, in simulation, may lead to non-physical oscillations in the reconstructed electromagnetic quantities.

For this reason, the general preprocessing pipeline includes an optional junction-blending stage. Its purpose is to smooth the transition region around the angular boundary by gradually combining the values from the two sides of the junction over a limited number of samples.

In the present implementation, this smoothing is based on a cosine-weighted blending coefficient of the form

$$\beta(k) = \frac{1}{2}\left(1 - cos\left(\pi \frac{k-1}{n_{blend}-1}\right)\right) \qquad (36)$$

where n_{blend} is the number of points included in the blending region.

If X_{left} and X_{right} denote the data approaching the junction from the two sides, the blended value is computed as

$$X_{blend} = (1 - \beta)X_{left} + \beta X_{right} \qquad (37)$$

This formulation provides a smooth transition between the two segments and avoids abrupt slope changes at the angular boundary.

In the Toyota Prius dataset used as the final application-oriented reference case, this stage was not activated. Since no additional theta tiling was required, no artificial junction was introduced in the angular domain, and therefore no dedicated blending operation was needed in the final LUTs used for the Simulink implementation.

The role of this step was therefore limited to defining the general preprocessing logic for cases in which angular reconstruction is required. More generally, whenever theta tiling

generates a synthetic boundary between two angular segments, junction blending acts as an intermediate conditioning stage between angular extension and final periodic closure.

4.8 Four-Quadrant Current-Domain Reconstruction

The source FEM dataset used for the final Simulink implementation does not directly provide a full four-quadrant current-domain representation. In fact, in the Toyota Prius case considered for the final Simulink implementation, the raw data are first identified on their native current domain, which in the MATLAB workflow corresponds to a reduced region with negative or zero d-axis current and positive or zero q-axis current. The preprocessing pipeline therefore includes a dedicated reconstruction step to extend the lookup tables to the full (i_d, i_q) plane before export to Simulink [9].

This step is performed only after the source axes have been identified, sorted, and validated, and after the MTPA trajectory has been evaluated on the source domain. In this way, the control-oriented MTPA search is based only on original FEM-supported data, whereas the final motor model exported to Simulink operates on a full four-quadrant current domain.

The first stage of the extension reconstructs the negative-i_q half-plane starting from the available positive-i_q source data. The new q-axis is obtained through a symmetric extension of the original current vector, and the electromagnetic quantities are continued according to parity rules chosen to preserve the expected physical behaviour with respect to the sign of i_q.

In particular:

- the d-axis flux linkage ϕ_d is extended as an even function of i_q
- the q-axis flux linkage ϕ_q is extended as an odd function of i_q
- and the electromagnetic torque T_e is also extended as an odd function of i_q

This choice is consistent with the expected sign reversal of torque and q-axis flux when the torque-producing current component changes sign.

After the q-axis extension, the zero-i_q line is explicitly regularised to enforce the expected physical constraints. In particular, ϕ_q and T_e are forced to zero on the $i_q = 0$ line, while ϕ_d

and the inductance maps are locally smoothed through interpolation between neighbouring points in order to avoid discontinuities introduced by discrete mirroring.

Once the q-axis has been extended, the dataset is further reconstructed over the positive-i_d region so as to obtain a complete four-quadrant current-domain representation. In this second step, different quantities are extended using different continuation rules, showing their particular physical behavior.

For the d-axis flux linkage, a continuation starting from $i_d = 0$ is applied instead. This choice avoids generating an unrealistically positive-id region and creates a smoother extension of the original FEM-supported data.

For the remaining quantities, a more conservative mirrored continuation is adopted. In particular, the q-axis flux linkage and torque maps are extended by reflecting the already reconstructed domain while preserving the parity rules imposed with respect to i_q. The inductance maps are extended using the same conservative logic, after which positivity is enforced wherever required for Simulink compatibility [9].

Figure 4.5 and Figure 4.6 show the Toyota Prius LUT surfaces after four-quadrant reconstruction. These plots make visible the extension of the original source dataset to the full current domain and provide a view of the electromagnetic and inductance maps used in the final Simulink model.

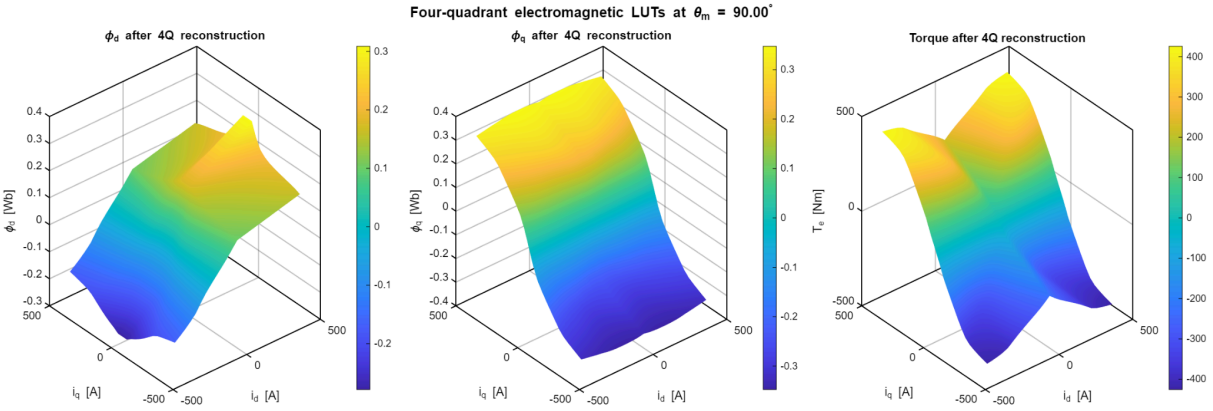

Figure 4.5 - Four-quadrant electromagnetic LUTs of the Toyota Prius dataset at a fixed rotor-angle slice: d-axis flux linkage, q-axis flux linkage, and electromagnetic torque after current-domain reconstruction.

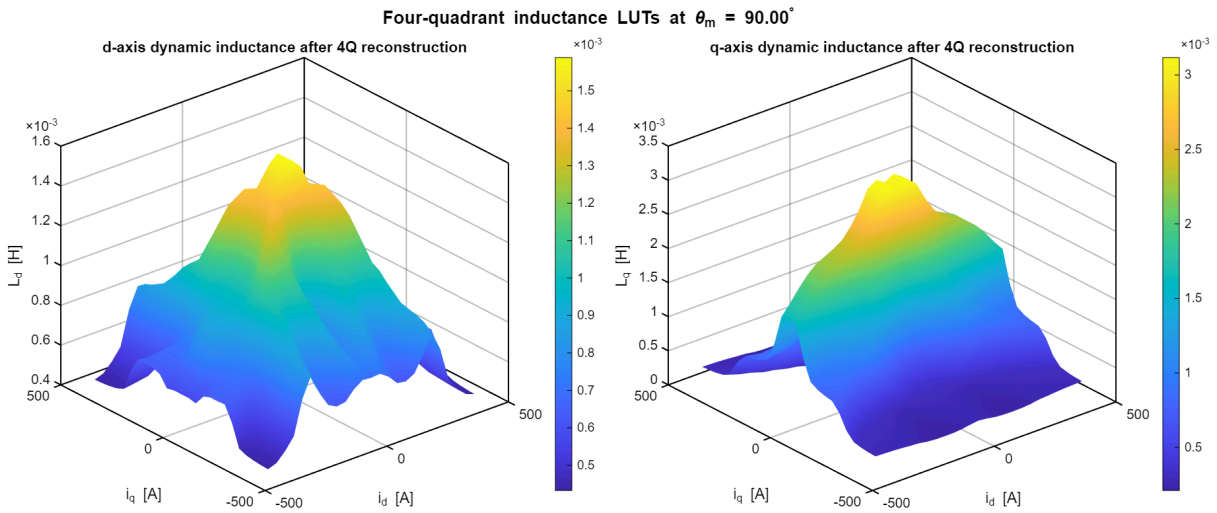

Figure 4.6 - Four-quadrant inductance LUTs of the Toyota Prius dataset at a fixed rotor-angle slice: representative d-axis and q-axis dynamic inductance maps after current-domain reconstruction.

After reconstruction, the resulting lookup tables are checked for dimensional consistency, symmetry, zero-line behavior, and the positive nature of the inductance maps. This verification validates that the extended dataset is numerically consistent and suitable for direct use in the LUT-based motor model.

The final result of this stage is therefore a physically regularised and numerically consistent four-quadrant representation of the machine, suitable for export to Simulink and for use in the subsequent control-oriented stages of the workflow.

4.9 Periodic Closure

After the previous conditioning stages, the Toyota Prius dataset used for the final Simulink implementation is continuous over the operating domain, but the exact periodicity with respect to the rotor angle still needs to be enforced explicitly. The selected Simulink block requires the lookup tables to be periodic along the angular dimension, which means that the first and last slices of each LUT must be identical [9].

Even when the angular domain is already suitable, as in the Toyota Prius case, small numerical mismatches may still remain between the first and last angular slices. These residuals are big enough to violate the periodicity condition required for the simulation.

For this reason, the final slice of each lookup table is explicitly set equal to the first. In this way, all datasets remain mutually aligned while satisfying the angular periodicity requirement.

This step does not alter the physical behaviour represented by the LUTs in any substantial way. Its purpose is purely numerical, because it removes any residual mismatch at the angular boundary and prevents artificial discontinuities when the rotor position wraps around the electrical period.

4.10 Final Dataset Validation and Export

At this stage, all conditioning operations are complete and the dataset is ready for final validation before export to the Simulink model workspace.

The purpose of this last step is to verify that the processed lookup tables satisfy all structural requirements needed for simulation. In particular, the pipeline checks that:

- all arrays have the expected dimensions,
- all breakpoint vectors are strictly increasing,
- the reconstructed quantities are finite over the full operating domain,
- and periodicity along the angular axis is satisfied.

These checks are essential because any inconsistency introduced during the previous stages can lead to non-physical responses or simulation errors.

In the implemented MATLAB workflow, this verification is carried out through a set of assertions applied to the final breakpoint vectors and to the processed LUT arrays. Additional tests are also performed on the reconstructed four-quadrant dataset, including zero-line behavior, symmetry properties, and angular periodicity. Together, these checks confirm that the final data are numerically well posed and ready to be used by the Simulink model.

Once validation is completed, the final lookup tables and the associated parameter vectors are assigned to the MATLAB base workspace so that they can be accessed directly by the Simulink model during initialisation.

Chapter 5 - Control Parameter Derivation from LUT

5.1 Overview and Objectives

The preprocessing developed in Chapter 4 provides a nonlinear motor model based on multidimensional lookup tables (LUTs). These datasets describe the electromagnetic behaviour of the machine over the operating domain and make it possible to represent effects such as magnetic saturation, cross-coupling, and rotor-position dependence in a way that is suitable for simulation [4], [6], [9].

At the same time, this nonlinear LUT-based description does not translate directly into conventional control design. Standard field-oriented control (FOC) is usually formulated in terms of a reduced set of motor parameters, such as inductances, torque constant, and effective flux linkage, within a simpler parametric model [17], [18], [23], [24].

The objective of this chapter is therefore to bridge the gap between the nonlinear FEM-based motor model and the needs of classical control design. To do so, the conditioned lookup tables are used to extract a set of local control-oriented parameters from a selected operating point. In this way, the machine's non-linear electromagnetic behavior is converted into a form that can be used to tune the controllers of the Simulink drive model.

The parameter calculation is carried out around an operating point selected on the Maximum Torque Per Ampere (MTPA) trajectory. This choice is consistent with the control strategy adopted in the thesis and with the normal operating region of the machine in traction applications [10], [11], [12].

The procedure developed in this chapter includes:

- determination of the MTPA operating point from the torque LUT,
- extraction of differential inductances from the flux-linkage maps,
- computation of the torque constant from the local torque variation,
- calculation of an equivalent flux-linkage parameter for control purposes,
- and tuning of the current and speed PI controllers based on the extracted quantities.

The parameters obtained in this way are then used in Chapter 6 to build the complete Simulink-based FOC model.

5.2 Operating Point

To extract the control parameters from the lookup tables, a representative operating point must first be selected. In this work, the reference operating point is chosen on the Maximum Torque Per Ampere (MTPA) trajectory, which means in the locus that provides the required torque with minimum stator current magnitude [10], [11], [12].

The stator current magnitude for a given current vector (i_d, i_q), was defined in (20) and similarly, the operating point is obtained from the constrained optimisation problem earlier defined in (21).

In the constant-torque region, the MTPA condition can therefore be written as the search for the current vector that minimises copper losses for a given torque request, calculated using the average electromagnetic torque over the rotor-angle dimension.

In the nonlinear FEM-based model used here, the torque function is not available in closed analytical form. Instead, it is obtained directly from the processed lookup tables [4], [6], [9]. For this reason, the MTPA trajectory must be identified numerically from the torque map rather than derived from a simplified analytical expression [10], [11], [12].

In the final MATLAB implementation, the average torque map is first computed by averaging the torque LUT over the rotor-angle dimension. The resulting mean torque surface is then interpolated on a dense current grid, and the MTPA trajectory is obtained through a torque-constrained minimum-current search. A mild continuity regularisation is introduced between adjacent operating points in order to avoid non-physical jumps along the final trajectory.

The operating point used for parameter extraction is then selected directly from this numerically identified MTPA curve. In this way, the chosen point is fully consistent with the actual nonlinear behaviour represented by the FEM-derived LUTs and with the control-oriented trajectory later exported to the Simulink model. Figure 5.1 shows the mean torque map, the resulting MTPA trajectory, and the selected operating point used in the subsequent parameter-extraction steps.

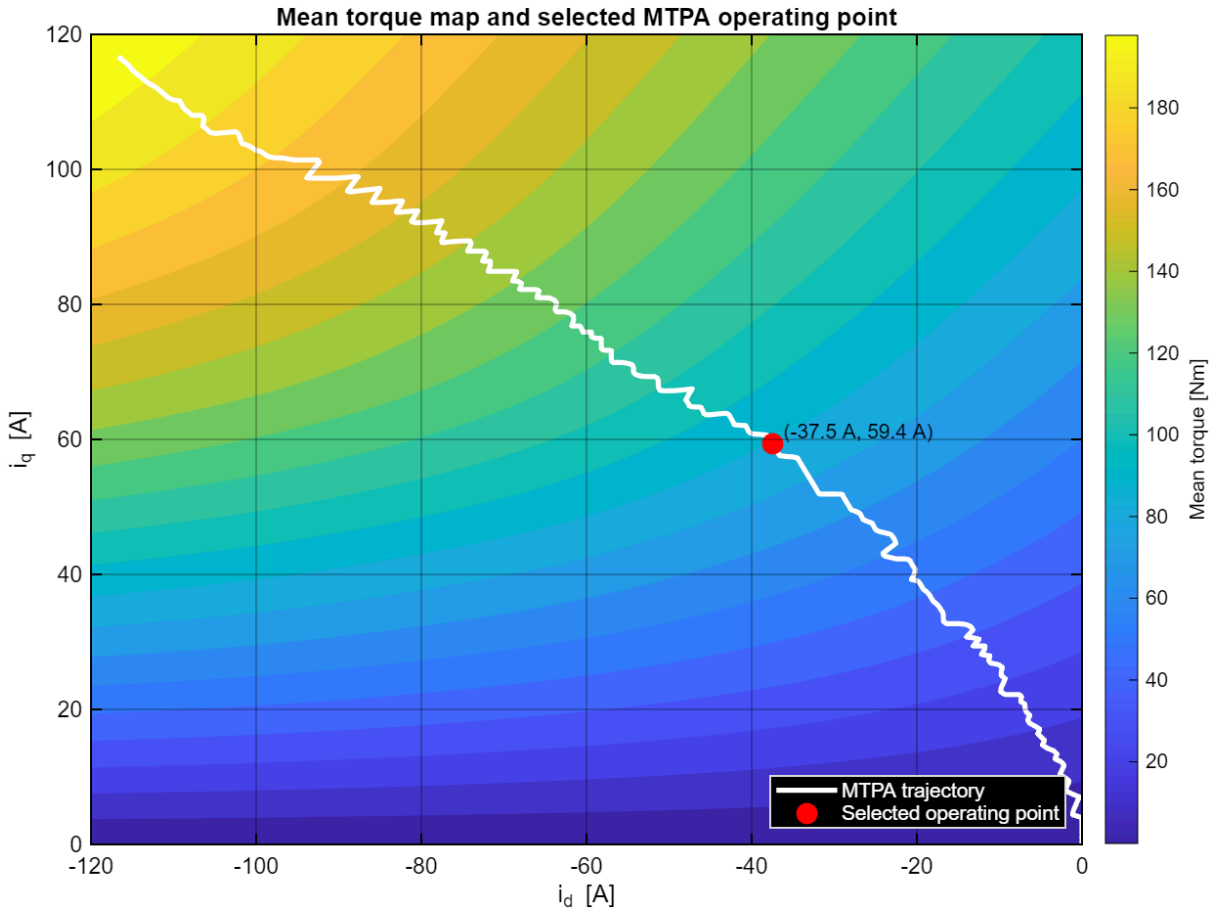

Figure 5.1 - Mean torque map over the source current domain, numerically identified MTPA trajectory, and selected operating point used for controller tuning.

Because the operating point is selected from the processed FEM torque map, it reflects the actual steady-state behaviour of the machine within the available source domain rather than an idealised analytical approximation. In the present implementation, the selected MTPA operating region lies in the quadrant $i_d < 0$, $i_q > 0$, consistently with the current-reference LUTs used for control.

All control-relevant quantities derived in the following sections, including differential inductances, torque constant, and equivalent flux-linkage parameter, are evaluated at this operating point or in its local neighbourhood.

5.3 Differential Inductance Extraction

In non-linear electric machines, the d-axis and q-axis inductances are not constant quantities. Magnetic saturation and cross-coupling cause them to vary with the operating point [4], [5], [8].

For control design, the relevant quantities are therefore the differential inductances, i.e., the local derivatives of the flux linkages with respect to the dq currents, evaluated at the selected operating point (i_d^*, i_q^*):

$$L_d^* = \left.\frac{\partial \phi_d}{\partial i_d}\right|_{(i_d^*, i_q^*)} \tag{38}$$

$$L_q^* = \left.\frac{\partial \phi_q}{\partial i_q}\right|_{(i_d^*, i_q^*)} \tag{39}$$

In addition to these main-axis terms, cross-saturation introduces the associated cross-derivative terms as seen in (32), but evaluated at (i_d^*, i_q^*), which quantifies the dependence of each flux component on the orthogonal current axis [5], [8].

The differential inductances represent the local slope of the flux-current characteristics and therefore determine how the current responds to voltage variations around the chosen operating point. For this reason, they are the appropriate inductances to use for controller design, rather than the static or apparent inductances, which do not capture the local nonlinear behaviour of the machine [5], [8].

In the present work, the FEM dataset directly provides dynamic inductance maps, which are used for parameter extraction [4], [6], [7]. The control-oriented inductances are therefore obtained by evaluating the corresponding LUTs at the selected MTPA operating point:

$$L_d^* = L_{d,dyn}(i_d^*, i_q^*) \tag{40}$$

$$L_q^* = L_{q,dyn}(i_d^*, i_q^*) \tag{41}$$

In the final MATLAB implementation, the selected operating point is first mapped to the nearest available indices of the source current grid. The corresponding dynamic inductance values are then extracted from the FEM maps and averaged over the rotor-angle dimension in order to obtain representative local values for control design. The end-winding inductance is then added to both axes so that the extracted quantities remain consistent with the electrical model used in the controller implementation.

Accordingly, the inductances used in the control model are computed as:

$$L_d = \overline{L_{d,dyn}}(i_{d'}^{*} i_q^{*}) + L_{ew} \tag{42}$$

$$L_q = \overline{L_{q,dyn}}(i_{d'}^{*} i_q^{*}) + L_{ew} \tag{43}$$

where $\overline{(\cdot)}$ denotes the average over the rotor-angle dimension and L_{ew} is the end-winding inductance.

The FEM dataset also provides cross-derivative information, which reflects the presence of cross-saturation. Although these terms are not used explicitly in the PI tuning formulas developed later, they still influence the effective inductance values extracted at the operating point and contribute to the residual coupling between the d-axis and q-axis dynamics [5], [8], [23].

The extracted inductances are then used in the following sections for:

- current-controller tuning,
- decoupling-term parameterisation,
- and estimation of the equivalent flux-linkage parameter.

These quantities remain local parameters. They describe the behaviour of the machine only in a neighbourhood of the selected MTPA point, and their value changes over the operating domain because of the nonlinear magnetic behaviour of the motor. As a result, controller performance may degrade if the operating condition moves too far away from the selected reference point. This limitation is one of the reasons why more advanced strategies, such as gain scheduling or adaptive control, may be of interest in more demanding applications.

5.4 Torque Constant Extraction

In classical PMSM modelling, the torque constant Kt is often introduced as a fixed parameter directly linked to the permanent-magnet flux linkage [17], [18]. This interpretation is convenient in linear models, but it is not fully adequate for the nonlinear LUT-based machine considered in this work, where torque depends on operating point, magnetic saturation, saliency, and cross-coupling effects [4], [5], [10]–[12].

For control design, the quantity of interest is the local sensitivity of electromagnetic torque to the torque-producing current component. Accordingly, in this work, the torque constant is introduced as a local quantity, evaluated in the neighborhood of the selected MTPA operating point:

$$K_t^* = \left.\frac{\partial T_e}{\partial i_q}\right|_{(i_d^*,\, i_q^*)} \tag{44}$$

This definition reflects how much the electromagnetic torque changes for a small variation of the q-axis current around the operating point. It is therefore the most relevant quantity for the outer speed control loop.

Since the torque is available only through FEM-derived lookup tables, the derivative must be evaluated numerically rather than obtained from a closed analytical model [4], [6], [9]. The first step is therefore to compute the mean torque map by averaging the torque LUT over the rotor-angle dimension:

$$\overline{T}_e(i_d, i_q) = \frac{1}{n_\theta} \sum_{k=1}^{n_\theta} T_e(i_d, i_q, \theta_{m,k}) \tag{45}$$

The MTPA trajectory is then identified numerically from this mean torque map, as described in Section 5.2. In the final MATLAB implementation, the selected operating point is taken from the interpolated MTPA curve, and the torque constant is estimated using the neighboring points around that location. The effective torque constant used for controller design is computed as:

$$K_t \approx \frac{T_2 - T_1}{i_{q,2} - i_{q,1}} \tag{46}$$

where T_1 and T_2 are the torque values of the two neighbouring points on the dense MTPA trajectory, and $i_{q,1}$, $i_{q,2}$ are the corresponding q-axis currents.

This procedure provides a local estimate of the torque sensitivity that is directly consistent with the nonlinear FEM data and with the actual MTPA trajectory exported to the control model. In contrast with simplified analytical formulations, the resulting torque constant is not assumed to be globally constant. Its value depends on the operating point and reflects the combined influence of magnet flux, saliency, and magnetic saturation [4], [5], [10]–[12].

Once extracted, the torque constant is used as a control-oriented parameter for the speed-loop design. Around the selected operating point, the electromagnetic torque can therefore be approximated locally as:

$$\Delta T_e \approx K_t \Delta i_q \qquad (47)$$

or, equivalently, as a first-order relationship between small variations in torque and small variations in q-axis current.

This approximation holds only near the selected operating point and serves three purposes:

- tuning the speed-controller gains,
- represent the electromechanical coupling in a control-oriented form,
- and preserve consistency between the non-linear plant and the classical FOC design framework.

Because of the nonlinear nature of the machine, several limitations must still be acknowledged:

- K_t varies with operating point,
- The local linear approximation remains valid only near the selected MTPA condition,
- Deviations from the MTPA region lead to variations in torque sensitivity, potentially resulting in a gain mismatch in the speed loop.

Despite these limitations, this numerical method to compute K_t from the FEM torque map provides a physically consistent parameter for controller tuning.

5.5 Flux Linkage Parameter for Control

The flux-linkage used in the controller is derived from the classical torque expression already introduced in (10), and using the local torque constant concept discussed in Section 5.4, it can be obtained by rearranging the torque-current relationship at the selected operating point (i_d^*, i_q^*). This leads to:

$$\Phi_{eq} = \frac{K_t}{1.5p} - (L_d - L_q) i_d^* \qquad (48)$$

In the present work, this is the quantity used for controller parameterisation. It is evaluated after the extraction of the local torque constant and differential inductances at the selected MTPA operating point, so that its value remains fully consistent with the nonlinear FEM-based LUT model and with the operating condition chosen for tuning.

This parameter is especially useful in the speed-control design, since it provides a compact control-oriented representation of the electromechanical coupling within the simplified FOC structure. Together with K_t, it allows the nonlinear machine behaviour to be expressed through a reduced set of local parameters suitable for PI-based controller design.

As with the quantities extracted in the previous sections, Φ_{eq} must be interpreted as a local parameter. Its value is valid only in the neighbourhood of the selected design point and is affected by saturation, current angle, and operating region. It should therefore not be interpreted as a globally constant machine parameter, but rather as an equivalent quantity introduced to preserve consistency between the nonlinear LUT-based plant and the analytical controller model [4], [5].

5.6 PI Current Controller Design

Once the local differential inductances have been extracted, the inner current loops can be tuned. As discussed in Chapter 2, field-oriented control is based on two PI regulators acting on the d-axis and q-axis currents, with the coupling terms compensated through feedforward

decoupling. Under this assumption, the two axes can be treated locally as first-order electrical subsystems for controller tuning purposes [29].

Around the selected operating point, the current dynamics can therefore be approximated by the transfer functions

$$G_d(s) = \frac{i_d(s)}{v_d(s)} = \frac{1}{L_d s + R_s} \quad (49)$$

$$G_q(s) = \frac{i_q(s)}{v_q(s)} = \frac{1}{L_q s + R_s} \quad (50)$$

where L_d and L_q are the local differential inductances extracted in Section 5.3, and R_s is the stator phase resistance.

The current controllers are chosen in PI form:

$$C_d(s) = K_{p,id} + \frac{K_{i,id}}{s} \quad (51)$$

$$C_q(s) = K_{p,iq} + \frac{K_{i,iq}}{s} \quad (52)$$

To obtain a simple and consistent tuning rule, the controller zero is placed so as to compensate for the electrical pole of each axis. By imposing the current-loop bandwidth ω_c, the controller gains become:

$$K_{p,id} = L_d \omega_c \quad (53)$$

$$K_{i,id} = R_s \omega_c \quad (54)$$

$$K_{p,ip} = L_q \omega_c \quad (55)$$

$$K_{i,iq} = R_s \omega_c \quad (56)$$

With this choice, the two current loops are tuned to have the same target bandwidth while preserving the difference between the d-axis and q-axis electrical dynamics through the local inductance values.

In the present work, these gains are computed using the local inductances extracted at the selected MTPA operating point and the stator resistance used in the electrical model. Since

the end-winding inductance has already been included in the values of L_d and L_q extracted in Section 5.3, the resulting PI gains remain consistent with the plant model used in simulation.

This tuning is intentionally local. The current loops are designed around the selected operating condition, so the resulting gains are most accurate in the neighbourhood of that point. As the operating condition moves away from the chosen MTPA region, the effective machine inductances change because of saturation and cross-coupling, and the local first-order approximation becomes less accurate.

Even so, this approach provides a simple and effective way of tuning the inner current loops while preserving consistency between the nonlinear LUT-based motor model and the classical PI-based FOC structure adopted in the Simulink implementation [23], [24].

5.7 Speed Controller Design

The outer control loop regulates the mechanical speed of the motor by generating the torque-producing current reference i_q^*. This loop operates at a lower bandwidth than the inner current loop and governs the overall dynamic response of the drive system [13], [14].

The electromechanical dynamics of the motor are described by:

$$J\frac{d\omega_m}{dt} + D_M\omega_m = T_e - T_L \qquad (57)$$

where J is the rotor inertia, D_m is the viscous damping coefficient, T_L is the load torque, and ω_m is the mechanical angular speed

Using the local linearisation introduced in Section 5.1 and the torque constant extracted in Section 5.4, the electromagnetic torque can be approximated locally as

$$T_e \approx K_t i_q \qquad (58)$$

Substituting this relation into the mechanical equation and treating the load torque as a disturbance input gives the transfer function from i_q to ω_m:

$$G(s) = \frac{\omega_m(s)}{i_q(s)} = \frac{K_t}{Js+D_m} \qquad (59)$$

This is a first-order system whose gain and time constant depend on the locally extracted torque constant and on the mechanical parameters.

A PI controller is then used for speed regulation:

$$C_{\omega}(s) = K_{p,s} + \frac{K_{i,s}}{s} \qquad (60)$$

The controller gains are selected by imposing a second-order closed-loop response with natural frequency ω_n and damping ratio ζ [30]. By coefficient matching, the resulting gains are

$$K_{p,s} = \frac{2J\zeta\omega_n - D_m}{K_t} \qquad (61)$$

In the final MATLAB implementation, the speed loop is tuned using the locally extracted values of K_t, the rotor inertia J, and the damping coefficient D_m. The adopted design values are

$$\omega_n = 1.5\, rad/s, \quad \zeta = 1.5$$

which lead directly to the PI gains used in the controller implementation.

This choice gives a well-damped speed response and preserves a clear separation of time scales with respect to the current loop. The design therefore remains aligned with the standard cascaded FOC assumption that the inner current loop is significantly faster than the outer speed loop [13], [14].

As in the previous sections, this tuning is local. The controller gains are derived around the selected MTPA operating point, so they are most accurate in the neighbourhood of that condition. Variations in torque sensitivity, saturation level, and operating point may therefore lead to changes in the effective outer-loop dynamics whenever the machine moves far away from the selected reference condition.

Despite this limitation, the resulting speed-controller design provides a simple and effective control-oriented representation of the electromechanical dynamics and is fully consistent with the nonlinear LUT-based parameter calculation procedure adopted in this thesis.

Chapter 6 - Simulink Model Architecture

6.1 Overview of the Simulation Framework

The complete drive system is implemented in Simulink as a closed-loop simulation framework organised around three main subsystems: TestSignals, Controller, and Plant.

The TestSignals subsystem generates the external inputs used to define the simulation scenario. These include the reference quantities required to excite the model under different operating conditions. In the present implementation, the most important monitored quantity is the mechanical speed reference, which is compared directly with the measured motor speed during the simulation.

The Controller subsystem receives the external test signals together with the measured variables returned by the plant. Based on these inputs, it computes the control action and outputs the command signals collected in the MotCmds bus. These commands are then applied to the plant model.

The Plant subsystem represents the electrical and mechanical drive system. It receives the controller outputs and returns the measured variables grouped in the MeasSigs bus. These signals are fed back to the controller in order to close the loop and are also routed to the top level for monitoring and validation.

This top-level organization provides a clear separation between scenario generation, control implementation, and physical system response. Such a structure improves the readability of the Simulink model and makes it easier to analyse, validate, and modify the different parts of the drive system independently while preserving their interaction in closed-loop.

Figure 6.1 shows the top-level Simulink architecture adopted in this work. The internal structure of the Plant and Controller subsystems is described in the following sections.

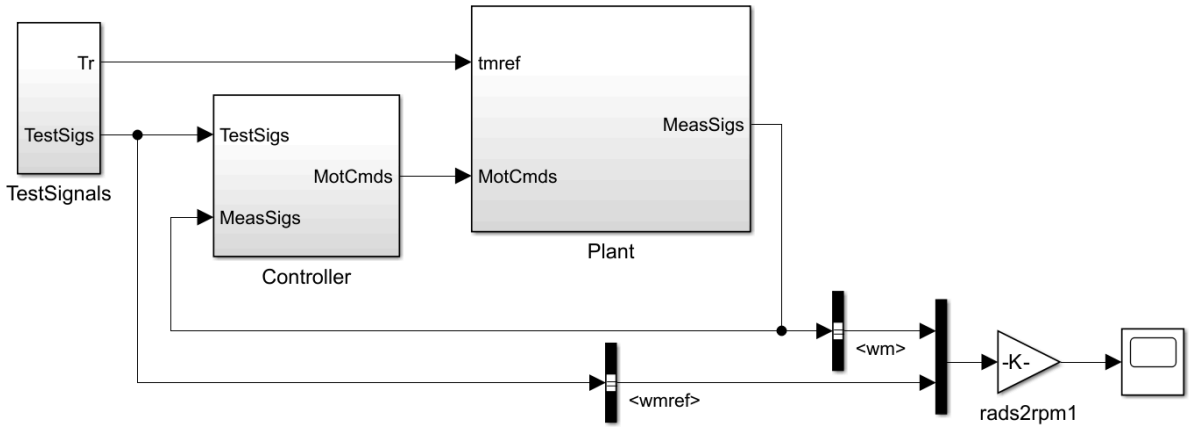

Figure 6.1 - Top-level Simulink architecture of the closed-loop drive model, including the TestSignals, Controller, and Plant subsystems.

6.2 Plant Subsystem Motor Model

The most important element of the Plant subsystem is the motor block, since it defines the electromagnetic and electromechanical behavior on which the controller is tested. In the present work, the machine is implemented using the FEM-Parameterized PMSM block available in Simscape Electrical [9].

The block is configured in the 3-D flux linkage data mode, without a thermal port, so that the machine behavior is described directly through lookup tables derived from the FEM preprocessing pipeline developed in Chapter 4 [9]. Rather than relying on a simplified analytical model with constant inductances and constant permanent-magnet flux, the block uses tabulated nonlinear data to reproduce the machine behavior over the operating domain.

More specifically, the selected data format is based on d-axis and q-axis flux linkages as functions of d-axis current, q-axis current, and rotor angle. The block, therefore, receives as inputs the current breakpoint vectors ID_PEAK and IQ_PEAK, the angular breakpoint vector theta_m_deg, the d-axis and q-axis flux-linkage tables FD and FQ, and the electromagnetic torque table T. These quantities correspond to the processed LUTs exported at the end of the MATLAB preprocessing workflow.

The machine is configured as a wye-wound system with no neutral port exposed. The number of pole pairs is set consistently with the considered machine dataset and with the preprocessing stage described earlier in the thesis. The Park convention used for the tabulated data is also explicitly defined in the block settings, making sure that the dq interpretation of

the LUTs is consistent with the convention adopted during the pipeline and in the controller's design. This point is particularly important, since any mismatch in dq convention or rotor-angle definition would lead to incorrect torque production and non-physical current-flux relationships in simulation [9]. The block also includes the stator phase resistance, R_phase.

From a modeling point of view, the main advantage of this implementation is that the motor block preserves the non-linear features of the FEM: magnetic saturation, cross-coupling between axes, and rotor-angle-dependent effects such as torque ripple. These phenomena would not be represented correctly by a classical linear dq model with constant parameters. By embedding the processed LUTs directly into the Simulink plant, the motor model used in simulation remains much closer to the machine's original electromagnetic characteristics.

On the mechanical side, the load torque is applied via the Simscape torque source block to enable the drive to be tested under different operating conditions. The resulting mechanical quantities, including rotor speed and rotor angle, are then measured and made available for both control feedback and result analysis.

Figure 6.2 shows the motor-related part of the Plant subsystem, while Figure 6.3 reports the main configuration settings of the FEM-Parameterized PMSM block.

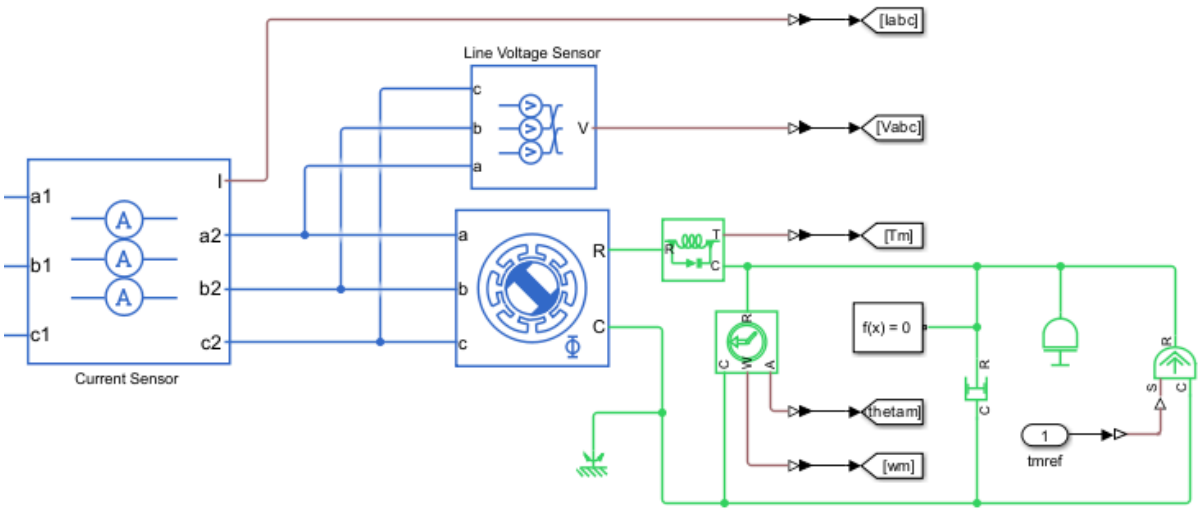

Figure 6.2 - Internal structure of the Plant subsystem, including the current and voltage sensors, LUT-based PMSM block, mechanical branch, and measurement outputs.

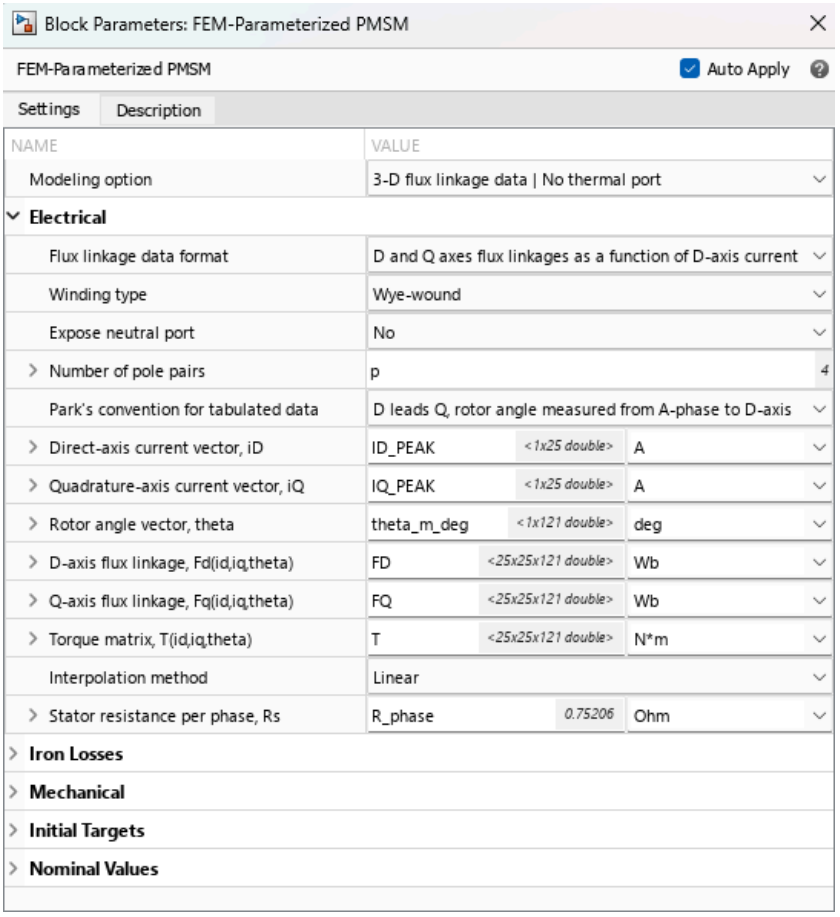

Figure 6.3 - Main configuration settings of the FEM-Parameterized PMSM block used in the Plant subsystem.

In addition to the motor model itself, the Plant subsystem includes the power-conversion stage and the measurement blocks required to close the control loop, as shown in Figure 6.4.

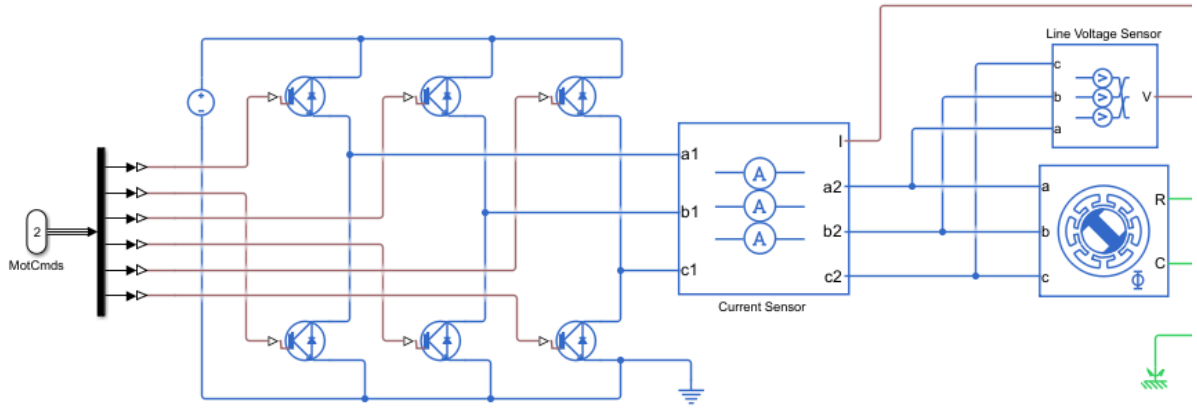

Figure 6.4 - Inverter and measurement part of the Plant subsystem, including the three-phase two-level converter, and current and voltage sensors.

At the electrical level, the plant is driven by a three-phase two-level inverter. The switching commands generated by the controller are received through the MotCmds bus and applied directly to the inverter switches. The converter then produces the three-phase stator voltages supplied to the motor block.

Between the inverter and the machine, current-sensing blocks are used to measure the stator phase currents, while a dedicated voltage-sensing block measures the motor terminal voltages. These measured quantities are required both for internal control feedback and for post-processing of the simulation results. Their role is particularly important because the controller operates in the dq reference frame, while the electrical interface of the plant remains naturally expressed in three-phase variables.

The measurements generated inside the plant are grouped into the MeasSigs bus. In the implemented structure, this bus includes the measured phase currents, the measured terminal voltages, the mechanical speed ω_m, the rotor angle, and the electromagnetic torque. These signals are then returned to the controller and are also routed to the top level for monitoring during simulation.

This organisation makes the Plant subsystem a compact but complete representation of the controlled drive system. It includes the converter, the nonlinear electromechanical model, and the measurement layer needed to connect the physical plant to the control algorithm, while preserving a clear separation between plant dynamics and controller logic.

6.3 Controller Subsystem

The Controller subsystem implements the complete control chain of the drive and generates the switching commands applied to the inverter in the Plant subsystem. At top level, it receives the speed reference from the TestSignals block and the measured signals returned by the plant through the MeasSigs bus.

Its internal structure is organised around a Field-Oriented Control (FOC) block followed by a dedicated PWM Generator (Three-phase, Two-level) block. The FOC block performs the control calculations in the rotating dq frame, while the PWM block converts the resulting three-phase voltage reference into the gate signals required by the two-level inverter. The overall structure of the Controller subsystem is shown in Figure 6.5.

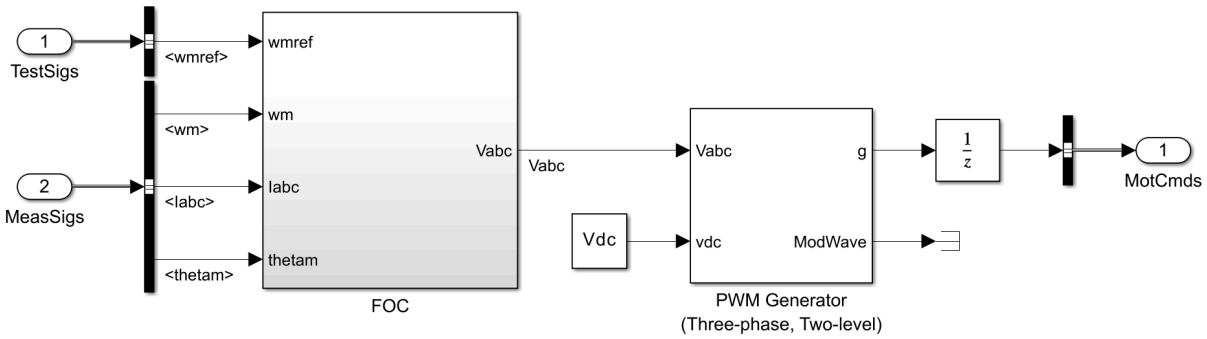

Figure 6.5 - Top-level structure of the Controller subsystem, including the FOC block, PWM generator, and output command bus.

Inside the FOC subsystem, the control structure follows the standard cascaded architecture introduced in Chapter 2, but with parameters and reference-generation logic derived from the FEM-based LUT model [23], [24]. The measured phase currents are first transformed into the rotating dq frame. The measured speed and speed reference are then processed by the outer speed controller, which generates the torque-producing current request. In parallel, the d-axis current reference is generated through the MTPA and flux-weakening logic. The resulting dq current references are passed to the inner current-control loop, which computes the dq voltage commands. Finally, these voltage commands are converted back into three-phase quantities and sent to the PWM generator. The internal organisation of the FOC block is reported in Figure 6.6.

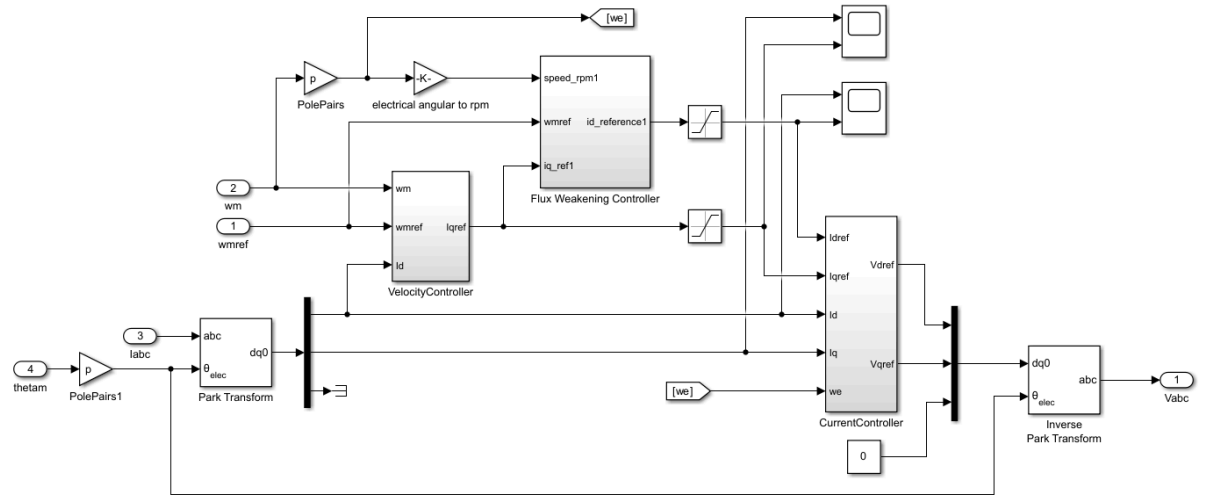

Figure 6.6 - Internal structure of the FOC block, including the Park Transform, VelocityController, Flux Weakening Controller, CurrentController, and Inverse Park Transform.

The PWM block receives the three-phase voltage reference produced by the FOC together with the DC-link voltage and outputs the gating signals required to drive the six switches of the inverter [23], [24], [31]. In this way, the controller is able to operate in terms of dq control variables while still driving the physical plant through realistic inverter commands.

A unit delay is also introduced before the final output bus so that the controller commands are synchronised with the discrete-time implementation of the simulation. This makes the interaction between controller and plant numerically cleaner and more representative of a sampled control structure.

From a modelling point of view, the Controller subsystem acts as the link between the control-oriented parameter set derived from the LUT data and the nonlinear plant model. It receives the plant response, computes the corresponding control action, and closes the loop through the inverter gate commands applied to the plant.

6.4 Current Control Loop Implementation

The inner current-control loop is implemented inside the CurrentController block and is responsible for regulating the d-axis and q-axis current components around their respective references. In the adopted FOC structure, this loop is the fastest control layer of the drive and directly determines the voltage commands applied to the inverter through the modulation stage [23], [24].

The CurrentController block receives five input signals: the d-axis current reference Idref, the q-axis current reference Iqref, the measured d-axis current Id, the measured q-axis current Iq, and the electrical angular speed we. The block outputs the corresponding dq voltage references Vdref and Vqref, which are then passed to the inverse transformation stage. The overall Simulink implementation of the current-control stage is shown in Figure 6.7.

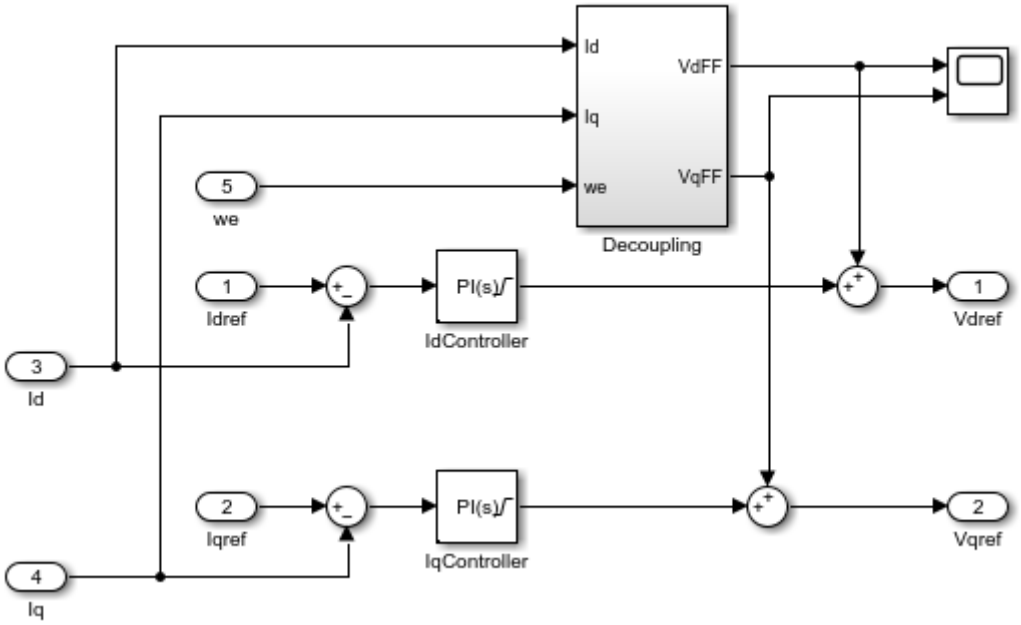

Figure 6.7 - Internal structure of the CurrentController block, including d-axis and q-axis PI regulators and the feedforward decoupling block.

The regulation is organised through two parallel PI controllers, one for the d axis and one for the q axis. In each case, the error signal is formed as the difference between the current reference and the measured current. These errors are processed independently by the IdController and IqController blocks, which generate the PI control actions associated with the two current channels.

In addition to the PI regulators, the implemented structure includes a dedicated Decoupling block. This block receives the measured dq currents and the electrical speed and computes the feedforward compensation terms required by the rotating-frame motor model [23], [8]. Its internal structure is reported separately in Figure 6.8, where the dq coupling contributions are generated and then routed back to the main current-control summation nodes.

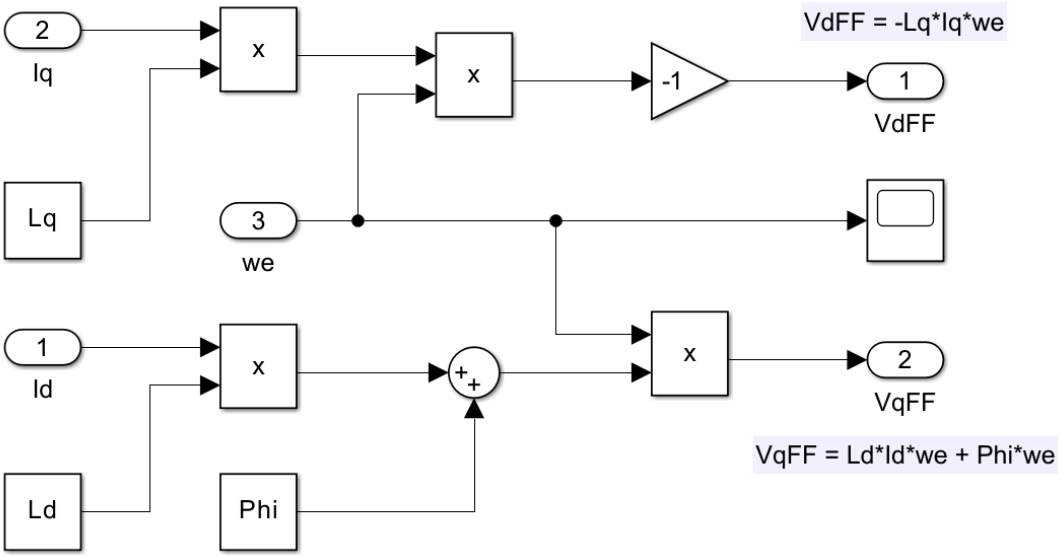

Figure 6.8 - Internal structure of the Decoupling block

More specifically, the decoupling subsystem provides the feedforward terms associated with the natural coupling of the dq model, while the PI regulators generate the corrective action required to eliminate the current-tracking error. The two contributions are then summed to form the final outputs Vdref and Vqref, as visible in the overall controller structure of Figure 6.7. In this way, the two current loops behave approximately as independent first-order subsystems, consistent with the control-oriented assumptions used in Chapter 5 [23], [8].

From a design point of view, the PI gains used in the two loops are derived from the local inductances and stator resistance extracted from the FEM-based LUT model. The current loop is therefore not tuned from a purely analytical constant-parameter motor, but from a local control-oriented representation of the nonlinear machine.

The current control is the core electrical regulator of the drive, because it receives the dq current references generated by the upper control layers, compares them with the measured dq currents, the rotating-frame coupling, and produces the voltage commands required to compensate for the difference and also to impose the desired electrical state on the motor.

6.5 Speed Control Loop Implementation

The outer speed-control loop is implemented through the VelocityController block. Its role is to regulate the mechanical speed of the motor and to generate the q-axis current reference required by the inner current loop. In the adopted cascaded FOC structure, this loop is slower than the dq current regulation and provides the torque demand needed to track the external speed reference [23], [24].

The VelocityController block receives the measured mechanical speed wm and the reference speed wmref. The difference between these two quantities defines the speed error, which is processed by a PI controller. The output of this regulator is the torque reference Tm*, representing the electromagnetic torque that the drive must produce in order to reduce the speed error. The corresponding Simulink implementation of the speed-control stage is shown in Figure 6.9.

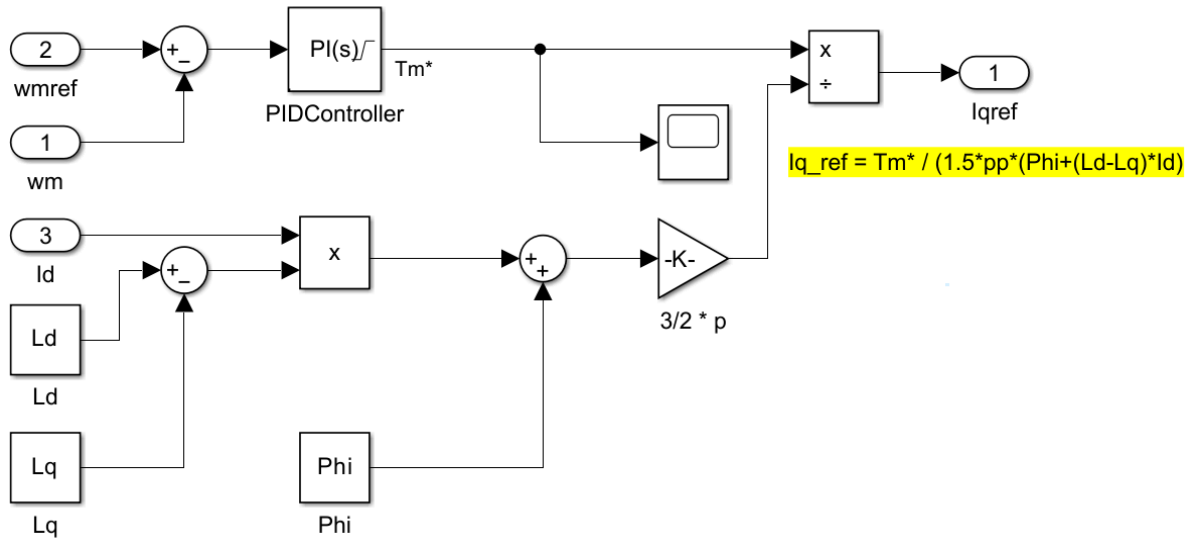

Figure 6.9 - Internal structure of the VelocityController block, including the PI speed regulator and the conversion from torque reference to q-axis current reference based on the local parameters Ld, Lq, and Φ.

In the implemented structure, the speed controller does not output the q-axis current reference directly. Instead, the torque request generated by the PI regulator is converted into `Iqref` through a control-oriented torque relation based on the local machine parameters extracted from the LUT model [17], [18]. More specifically, the conversion includes the quantity

$\phi + (L_d - L_q)i_d$ together with the factor $\frac{3}{2}p$, so that the q-axis current reference is computed consistently with the local electromechanical model used for controller design.

A relevant feature of this implementation is that the conversion from torque reference to current reference depends explicitly on the d-axis current and on the local quantities L_d, L_q, and Φ. The speed loop is therefore not based on a fixed global torque constant, but on a local control-oriented representation of the machine. This is consistent with the LUT-based motor model's nonlinearity and allows the outer loop to remain better aligned with the plant's actual electromagnetic behavior at the selected operating condition.

The resulting Iqref signal is then passed to the lower control layer, where it is regulated, along with the d-axis current reference, by the current controllers. In this way, the speed loop closes the electromechanical part of the drive, while the current loop makes sure that the corresponding electrical state is actually imposed on the motor. This structure guarantees a clear separation between speed regulation and electrical current regulation.

The PI determines the torque demand from the mechanical speed error, while the subsequent conversion stage translates that torque demand into the appropriate q-axis current reference using the local machine model. This makes the speed loop fully consistent with the control-oriented parameter set derived in Chapter 5 and with the nonlinear LUT-based plant used in the simulation.

6.6 MTPA and Flux-Weakening Reference Generation

The generation of the d-axis current reference is implemented using dedicated lookup-table-based logic that combines control-oriented MTPA and flux-weakening information into a single block. In the Simulink model, this function is implemented using a two-dimensional lookup table that receives the motor speed and the q-axis current reference generated by the speed loop as inputs. Its output is the d-axis current reference used by the inner current controller. The corresponding implementation of the d-axis current-reference generation logic is shown in Figure 6.10.

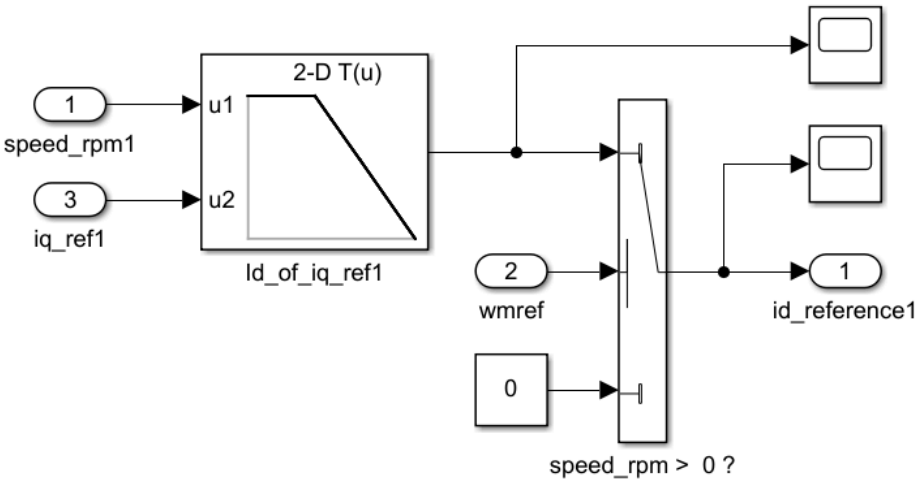

Figure 6.10 - Implementation of the d-axis current-reference generation through a two-dimensional lookup table and switching logic based on the speed-reference signal.

From a functional point of view, this block provides the current-reference scheduling required to move the operating point across the drive's different regions. For low- and medium-speed operation, the reference follows the control-oriented trajectory derived from the LUT data, consistently with the MTPA logic introduced in the previous chapters [18], [19], [20]. As the operating point moves toward higher speed, the same scheduling mechanism allows more negative d-axis current values to be selected, thereby enabling flux weakening [15], [16], [25], [26].

In the implemented model, the d-axis current reference is therefore not generated analytically online from the motor equations. Instead, it is read from a precomputed two-dimensional table that maps the operating condition to the corresponding current reference. This choice is fully consistent with the LUT-based philosophy adopted throughout the thesis: the reference-generation logic itself is derived from offline processed data, rather than from an idealized analytical approximation.

After the lookup-table evaluation, the resulting d-axis reference is passed through a switching stage. The switch is controlled by the reference-speed signal and uses a zero threshold. As a result, when the speed reference is active, the d-axis current reference provided by the table is passed to the current-control loop; when the speed reference is zero or inactive, the output is forced to zero. This prevents the controller from imposing a non-zero d-axis current reference when the machine is not commanded to run.

This implementation has two advantages. First, it keeps the reference generation simple and numerically robust, since the d-axis current command is obtained directly from precomputed data. Second, it guarantees that the controller behavior remains coherent with the reference trajectories derived offline from the processed FEM dataset. In this way, the same framework is used both for the motor model and for the current-reference generation applied in the controller.

Together with the q-axis current reference produced by the speed loop, this block completes the reference-generation part of the FOC implementation. It provides the link between the upper control logic and the dq current regulators, while ensuring a smooth transition between the MTPA region and the flux-weakening region of operation [18], [19], [20], [15], [16], [25], [26].

6.7 Test-Signal Generation and Injection

In addition to the plant and controller subsystems, the complete simulation framework includes a dedicated signal-generation stage used to impose the reference-speed trajectory and the corresponding load-torque profile. This function is implemented through the TestSignals subsystem, which provides the external inputs of the closed-loop model.

The role of this subsystem is to convert predefined test profiles into the internal signals required by the Simulink architecture. In particular, the generated outputs are grouped into the TestSigs bus, which is then passed to the Controller and Plant subsystems. In this way, the same closed-loop model can be exercised under different validation scenarios without modifying the internal control structure.

For the WLTP-based simulations, the input of the subsystem is the prescribed vehicle-speed profile. This signal is processed to generate two quantities. First, the speed reference applied to the controller is obtained by converting the vehicle speed into the corresponding motor mechanical speed through the transmission ratio and wheel radius. In the implemented model, this conversion is expressed as

$$\omega_{m,ref} = \frac{vG}{r_{wheel}} \qquad (62)$$

where v is the vehicle speed, G is the transmission ratio, and r_{wheel} is the wheel radius.

Second, the subsystem generates the resistant torque signal applied to the plant. In the present implementation, this is obtained from a simplified road-load model including rolling resistance and aerodynamic drag. The rolling-resistance contribution is represented by a constant force term

$$F_{roll} = \mu mg \qquad (63)$$

while the aerodynamic drag is computed as

$$F_{aero} = \frac{1}{2}\rho C_d A_r v^2 \qquad (64)$$

where μ is the rolling-resistance coefficient, m is the vehicle mass, g is the gravitational acceleration, ρ is the air density, Cd is the aerodynamic drag coefficient, and Ar is the frontal area. The total resistant force is then written as:

$$F_{res} = F_{roll} + F_{aero} \qquad (65)$$

and converted into the equivalent motor-side resistant torque through:

$$T_r = F_{res} \frac{r_{wheel}}{G} \qquad (66)$$

This is the torque signal applied to the plant during the WLTP-based simulations.

The corresponding Simulink implementation is shown in Figure 6.11.

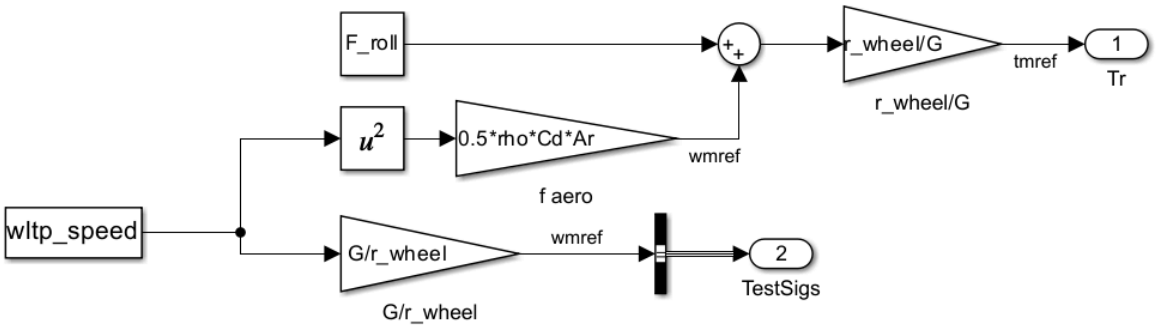

Figure 6.11 - TestSignals subsystem used for WLTP-based simulations: generation of motor-speed reference and equivalent resistant torque from the prescribed vehicle-speed profile.

The block receives the WLTP speed signal as input, computes the aerodynamic term from the square of the speed, adds it to the rolling-resistance contribution, converts the total resistant force into the equivalent motor-side torque Tr, and generates the speed-reference signal that is routed through the TestSigs bus. This structure makes it possible to test the controller under

drive-cycle-based operating conditions while conserving a clear separation between the test generation stage and the internal drive model.

For the step-response test discussed later in Section 7.2, a simpler input configuration is used. In that case, the speed reference is imposed directly through a Step block, while the applied resistant torque is set equal to zero. The step test, therefore, represents the most basic reference-tracking condition, whereas the WLTP-based cases introduce a more realistic force through the TestSignals subsystem.

Chapter 7 - Simulation Results and Validation

7.1 Validation Approach and Test Scenarios

The purpose of the simulations is to assess the closed-loop speed-tracking capability of the implemented drive model under both idealized and more realistic conditions. In the results, only the motor-speed response is reported, since this is the most direct indicator of the overall reference-following performance of the implemented control structure.

The validation is organized around four simulation scenarios. The first is a conventional speed-step response test used to evaluate the closed-loop system's basic behavior, including tracking capability, overshoot, and steady-state error. The other three scenarios are based on segments of the Worldwide harmonized Light vehicles Test Procedure (WLTP) speed profile and are used to assess the controller under progressively more demanding time-varying operating conditions [32]. The WLTP is a standardized driving-cycle framework developed to represent vehicle operation under more realistic speed variations than simplified laboratory tests or idealized step references [32].

In the present work, the WLTP-based tests are not used as a complete homologation-oriented vehicle assessment, but as dynamic speed-reference profiles suitable for evaluating the behavior of the implemented motor control system under more representative conditions [2], [32].

In principle, the validation plan included simulating the full WLTP speed profile as a single continuous test. In practice, however, the complete simulation repeatedly exceeded available memory. For this reason, the WLTP-based assessment was divided into three separate simulation runs corresponding to the Medium-, High-, and Extra-High-speed phases. This choice made the simulations computationally manageable while still preserving the possibility of evaluating the controller over increasingly demanding portions of the drive cycle. This decision is also supported by the absence of a thermal model for the losses' analysis.

The WLTP-based reference signals and the step reference signal used in the following simulations are generated through the TestSignals subsystem described in Section 6.7.

The results are presented in the same order. The step test is discussed first, since it provides the clearest interpretation of the basic closed-loop behavior. The three WLTP segments are then analyzed separately, so that the controller response can be assessed under progressively wider speed ranges and more dynamic reference variations.

In summary, the objective of the chapter is not to provide a complete vehicle-level certification study, but to verify whether the implemented LUT-based control model remains stable and able to follow the imposed speed reference under increasingly demanding test conditions.

7.2 Speed Step Response

The first validation scenario is a conventional speed step test, used to assess the basic closed-loop behavior of the implemented drive model. In the context of electric-drive control, this type of test is useful because it provides an immediate evaluation of the dynamic response of the cascaded control structure in terms of rise behavior, overshoot, settling trend, and steady-state tracking [23], [24], [29], [30].

In the present case, the speed reference is changed from zero to 3000 rpm, and the measured motor speed is compared directly with the imposed reference. The resulting response is shown in Figure 7.1.

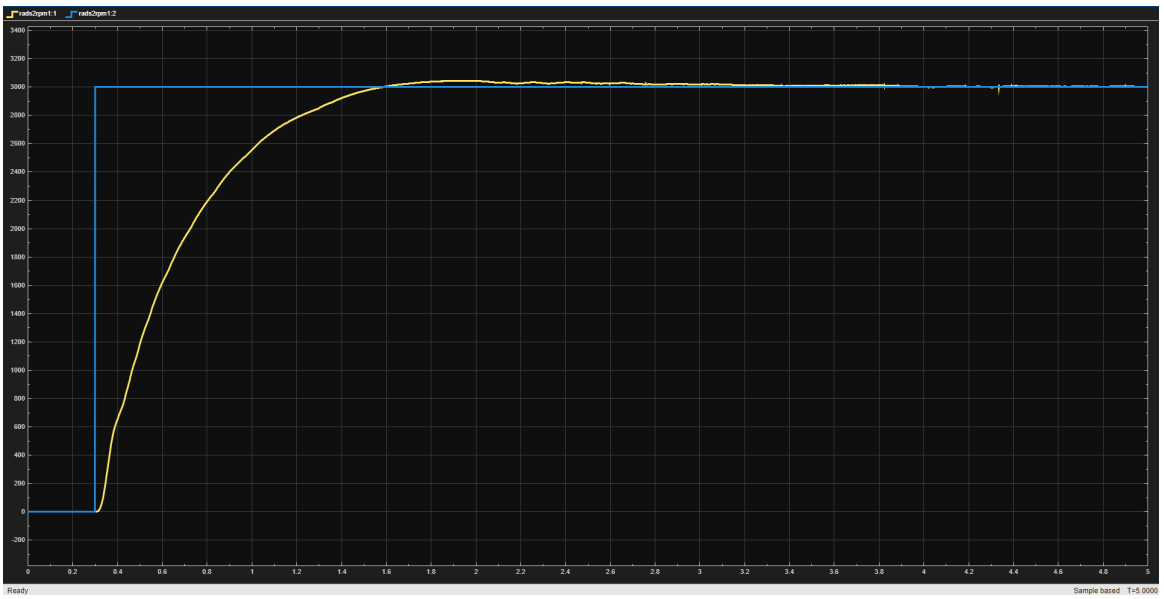

Figure 7.1 - Motor-speed response under the step tracking test: measured speed and reference speed.

Figure 7.1 shows that the implemented controller is able to track the imposed speed command with stable overall behavior. After the step is applied, the measured speed rises smoothly toward the reference without oscillatory transient behavior and reaches the target operating region progressively. The response is relatively slow compared with an aggressively tuned speed loop, but it remains well controlled throughout the transient, apart from a limited overshoot that is visible once the reference is reached. After this peak, the measured speed converges back toward the imposed value and remains close to it for the rest of the simulation, with negligible fluctuations. This indicates that the selected controller tuning provides a stable response and acceptable steady-state tracking, even if the transient is intentionally not particularly fast.

From a control point of view, this test is mainly useful as a preliminary validation of the complete cascaded FOC implementation. In particular, it confirms that the outer speed controller, the inner current loop, and the LUT-based plant model interact correctly under a basic reference change, without introducing numerical instability or loss of control.

7.3 WLTP Medium Phase

After the preliminary step test, the controller is assessed under a more realistic time-varying reference by using the Medium-speed phase of the WLTP profile. This test is intended to verify whether the implemented speed-control structure remains capable of following a non-constant reference trajectory under operating conditions that are more representative than an idealized step command [2], [32].

The corresponding result is shown in Figure 7.2, where the measured motor speed is compared with the imposed reference over the considered WLTP Medium-phase segment.

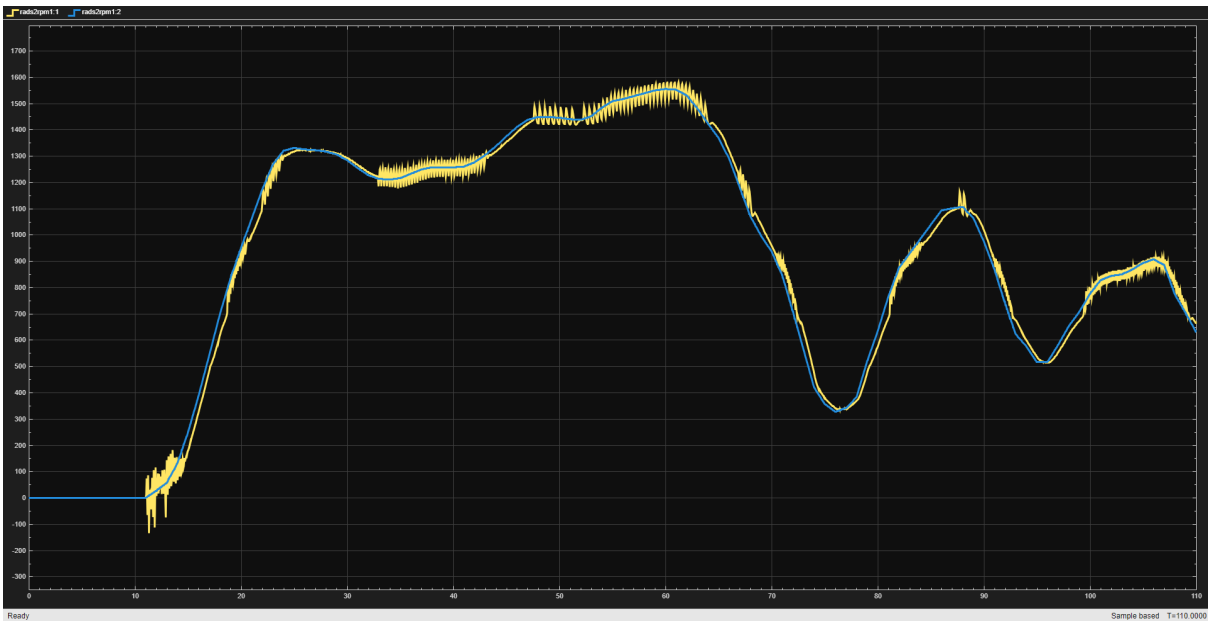

Figure 7.2 - Motor-speed response during the WLTP Medium-phase test: measured speed and reference speed.

Figure 7.2 shows that the implemented controller is overall able to follow the reference trajectory with stable behavior throughout the test. The measured speed reproduces the main trend of the imposed profile over the full simulation interval, including the initial acceleration, the intermediate plateau regions, the subsequent speed reduction, and the final sequence of lower-speed variations.

At the same time, the result also highlights undesired effects that were not visible to the same extent in the step test. In particular, a noticeable oscillatory ripple appears around the reference during several portions of the profile, especially in the medium-speed plateau regions. In addition, small local delays and moderate deviations are visible during some of the more pronounced accelerations and decelerations. These effects do not compromise stability, but they show that the tracking quality is less ideal once the controller is subjected to a continuously varying reference.

This result can still be considered positive because the controller remains stable throughout the full Medium-phase test, and the measured speed continues to follow the reference.

The Medium-phase result, therefore, acts as an intermediate validation stage between the idealized step test and the more demanding WLTP segments discussed in the following sections. It confirms that the implemented framework can handle drive-cycle-inspired operating conditions while also making the current tuning's first tracking limitations visible.

7.4 WLTP High Phase

The next validation scenario considers the High-speed phase of the WLTP profile. Compared with the Medium phase, this test imposes a broader speed range and more demanding reference variations, and is therefore useful to assess whether the implemented speed-control structure preserves acceptable tracking behaviour under more severe dynamic conditions [2], [32]. The corresponding result is shown in Figure 7.3, where the measured motor speed is compared with the imposed reference over the selected WLTP High-phase segment.

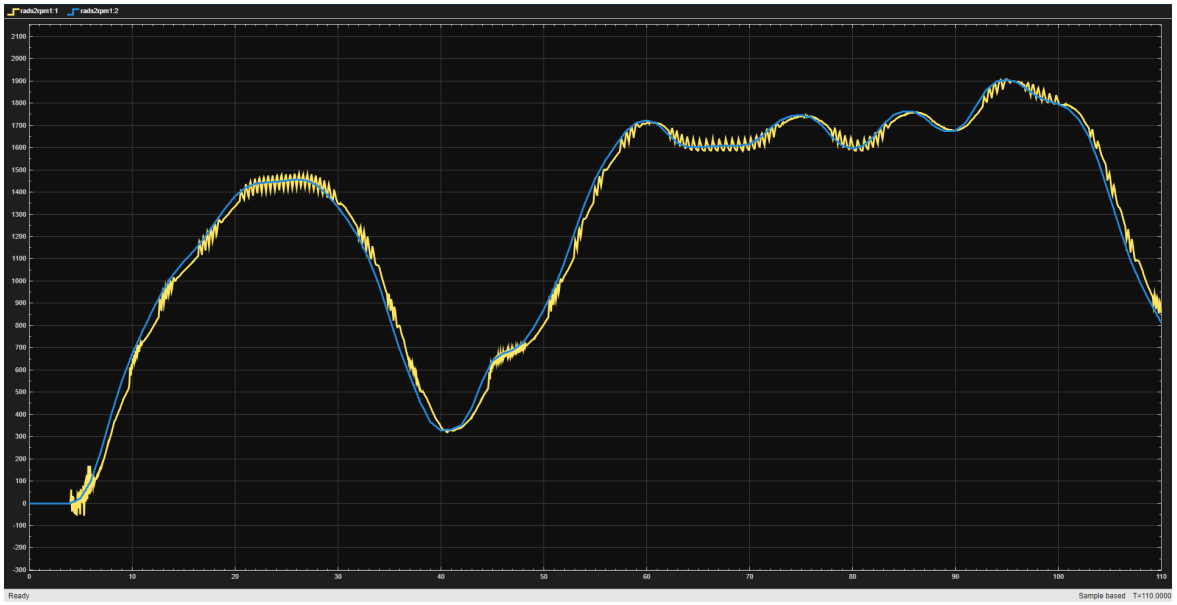

Figure 7.3 - Motor-speed response during the WLTP High-phase test: measured speed and reference speed.

Figure 7.3 shows that the controller follows the overall reference trajectory with stable behavior throughout the test. The measured speed reproduces the main trend of the imposed profile over the full simulation interval, including the initial acceleration, the intermediate speed reductions, the subsequent higher-speed operating region, and the final deceleration.

Despite the presence of the same undesirable effects seen during the previous simulation, the closed-loop response remains stable and does not show loss of control or divergence.

This result is therefore important for two reasons. First, it confirms that the implemented drive model can operate correctly under a more severe reference-speed profile than the previous tests. Second, it reveals that, in this operating region, the current control structure is

still affected by oscillatory behavior and limited transient mismatch, which will be included in Chapter 8.2 about the implementation's limitations.

7.5 WLTP Extra-High Phase

The final validation scenario considers the Extra-High-speed phase of the WLTP profile. Among the WLTP-based tests reported in this chapter, this is the most demanding one because it involves the highest speed levels and the widest reference variations. For this reason, it provides the most severe assessment of the speed-tracking capability of the implemented LUT-based drive model under the adopted simulation conditions [2], [32].

The corresponding result is shown in Figure 7.4, where the measured motor speed is compared with the imposed reference over the selected Extra-High-phase segment.

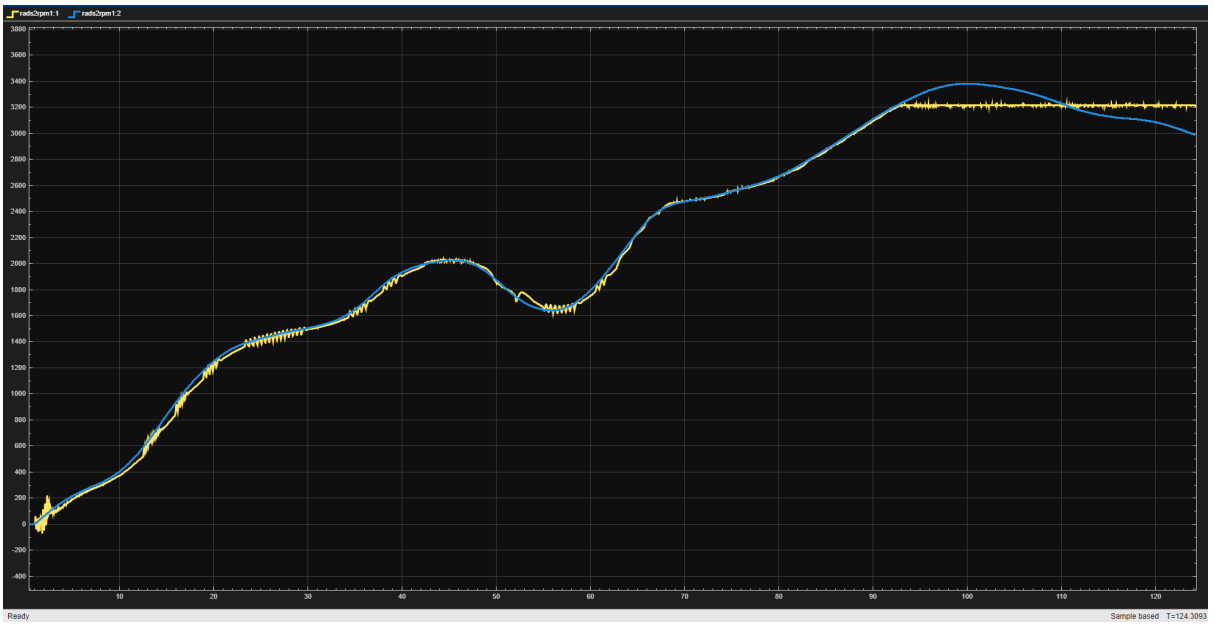

Figure 7.4 - Motor-speed response during the WLTP Extra-High-phase test: measured speed and reference speed.

The figure shows that the implemented controller is still able to follow the overall reference trajectory with stable behavior during most of the test. The main deviation appears in the final high-speed region, where the measured speed tends to remain close to an upper operating value while the reference continues to vary. Within the implemented control structure, this behavior is consistent with the effect of the saturation and limiting constraints active in the

upper operating region, where the achievable speed trajectory is restricted by the available voltage and current margins.

The result can be interpreted as a controlled limitation of the achievable operating speed under the most demanding part of the imposed profile, rather than as a loss of closed-loop stability.

Even with this limitation, the result remains important. The closed-loop system preserves stability over the duration of the full Extra-High-phase test, and the controller is able to reproduce the overall speed evolution correctly over a wide operating range before the final high-speed restriction becomes dominant. The test, therefore, confirms that the implemented LUT-based drive model can be exercised under the most demanding reference-speed conditions considered in this work, while also revealing the expected effect of the control and operating constraints in the upper speed region.

Chapter 8 - Conclusions and Future Work

8.1 Summary of Contributions

This thesis presented a complete LUT-based modeling workflow for electric-drive simulation, from raw FEM motor-characterization data to a closed-loop Simulink implementation based on field-oriented control. The main contribution of the work is not limited to the final simulation model itself, but includes the full intermediate processing chain required to make FEM data usable in a control-oriented simulation environment.

The first contribution of the thesis is the definition and implementation of the preprocessing pipeline discussed in Chapter 4, which converts raw FEM outputs into lookup tables compatible with the Simscape Electrical FEM-Parameterized PMSM block discussed in Chapter 6.2. Taken together, these steps bridge the gap between electromagnetic characterization data and simulation-ready motor models.

The second contribution is the extraction of the control parameters directly from the processed LUT data, as discussed in Chapter 5.

The third contribution is the complete Simulink drive model, integrating the LUT-based motor representation with a cascaded FOC architecture, as discussed in Chapter 6. The final model includes the plant subsystem, the inverter and measurement structure, the current and speed control loops, and the LUT-based reference generation used for MTPA and flux-weakening operation. In this way, the thesis demonstrates how non-linear FEM-derived motor data can be embedded into a simulation framework suitable for closed-loop control analysis.

The fourth contribution is the validation of the implemented framework through a structured set of simulation tests focused on speed tracking. The reported results include a conventional speed step test and three WLTP-based operating segments corresponding to the Medium-, High-, and Extra-High-speed phases. These results showed that the model is able to reproduce stable closed-loop behavior over progressively more strict conditions, while highlighting the effect of control and operating constraints in the upper speed region.

This confirms that the overall LUT-based simulation environment is suitable for control-oriented studies and provides an effective alternative to simplified linear machine models when higher electromagnetic fidelity is required.

8.2 Limitations

Despite the achieved results, the present work also has a number of limitations that should be acknowledged clearly.

A first limitation concerns the scope of the reported validation results. The final simulation assessment presented in this thesis is intentionally restricted to speed tracking, since this was the most robust and informative system-level quantity within the available development time and computational power limitations. Internal electrical variables such as current and torque were used during model construction and debugging, but they are not discussed in the final results chapter in the same detail.

A second limitation concerns the computational burden of the adopted simulation framework. Although the LUT-based model is much lighter than direct FEM simulation, the closed-loop Simulink implementation remains significantly more demanding than a classical linear dq model. This became especially evident in the WLTP-based tests, where the full cycle could not be simulated as a single continuous run within the available environment and had to be divided into separate Medium-, High-, and Extra-High-phase simulations because of memory limitations.

A further limitation emerges in the upper operating region, where the Extra-High WLTP test shows that the achievable speed trajectory is restricted by the control and operating constraints active at high speed. As a result, the final test confirms stable behavior in the upper region, but also shows that tracking accuracy becomes more limited when the imposed reference approaches the most demanding operating conditions considered in this work.

Another limitation is that the procedure was developed using two complementary motor datasets, each of which was useful for a different purpose. The Toyota Prius dataset served as the main application-oriented reference for the implementation stage, whereas the synchronous reluctance dataset was particularly useful for illustrating some preprocessing stages more clearly. This dual-dataset strategy strengthened the generality of the workflow,

but it also means that the final thesis does not present a single perfectly uniform case study across every chapter.

Finally, the controller design remains based on a local control-oriented approximation extracted from the nonlinear LUT data around selected operating conditions. This is effective for implementation and simulation, but it does not fully exploit the possibility of gain adaptation or operating-point-dependent controller scheduling across the entire machine domain.

8.3 Future Directions

Several natural extensions of the present work can be identified.

A first development direction is to extend the validation campaign to a more complete set of reported outputs, including dq currents, torque, voltage commands, and flux-weakening trajectories, in addition to speed. A second natural step is to perform a full continuous WLTP simulation without splitting the cycle into separate phases, which would require a more computationally optimized implementation of the current framework. Another important direction is the improvement of controller behavior in the upper operating region, where the present results show the influence of high-speed control and operating constraints on the achievable tracking performance.

A natural next step would be to embed the model in a broader vehicle-level simulation. This means coupling it with a longitudinal vehicle model, drivetrain elements, power supply, and battery models so that the analysis can cover a full traction-system scenario under standard driving cycles rather than being limited to speed-reference tracking.

The motor model itself also leaves room for improvement. Incorporating iron losses, thermal effects, and efficiency estimation — along with additional electromagnetic quantities from FEM — would push the LUT-based framework beyond control-oriented simulation and into the territory of performance and energy-consumption analysis.

A further improvement would be the introduction of adaptive or scheduled controller tuning across the operating domain. Since the machine behavior is strongly nonlinear, a controller whose gains vary consistently with the operating point could exploit the available LUT information more completely than the fixed local tuning adopted here.

Finally, the preprocessing methodology developed in this thesis could be generalized further and packaged into a more automated software tool. In this form, it could become a reusable interface between FEM-based motor characterization environments and Simulink drive models, lowering manual conditioning effort and improving reproducibility throughout different machines and datasets.

References

[1] C. C. Chan, “The state of the art of electric, hybrid, and fuel cell vehicles,” Proceedings of the IEEE, vol. 95, no. 4, pp. 704–718, 2007.

[2] M. Zeraoulia, M. E. H. Benbouzid, and D. Diallo, “Electric motor drive selection issues for HEV propulsion systems: A comparative study,” IEEE Transactions on Vehicular Technology, vol. 55, no. 6, pp. 1756–1764, 2006.

[3] G. Pellegrino, A. Vagati, P. Guglielmi, and B. Boazzo, “Comparison of induction and PM synchronous motor drives for EV application including design examples,” IEEE Transactions on Industry Applications, vol. 48, no. 6, pp. 2322–2332, 2012.

[4] N. Bianchi and S. Bolognani, “Magnetic models of saturated IPM motors based on finite element analysis,” in Conference Record of the 1998 IEEE Industry Applications Conference. Thirty-Third IAS Annual Meeting, 1998.

[5] B. Štumberger, G. Štumberger, D. Dolinar, A. Hamler, and M. Trlep, “Evaluation of saturation and cross-magnetization effects in interior permanent-magnet synchronous motor,” IEEE Transactions on Industry Applications, vol. 39, no. 5, pp. 1264–1271, 2003.

[6] N. Bianchi, Electrical Machine Analysis Using Finite Elements. Boca Raton, FL, USA: CRC Press, 2005.

[7] Altair Engineering, Flux 2024 User’s Guide. Altair Engineering Inc., 2024.

[8] L. Harnefors and H.-P. Nee, “Model-based current control of AC machines using the internal model control method,” IEEE Transactions on Industry Applications, vol. 34, no. 1, pp. 133–141, 1998.

[9] MathWorks, “FEM-Parameterized PMSM - Permanent Magnet Synchronous Motor Defined in Terms of Magnetic Flux Linkage,” Simscape Electrical Documentation, 2026.

[10] MathWorks, “PMSM Field-Oriented Control - Permanent magnet synchronous machine field-oriented control,” Simscape Electrical Documentation, 2026.

[11] S. Kim, J. Hong, and K. Nam, “MTPA control of an IPM machine based on signal injection considering inductance saturation,” IEEE Transactions on Power Electronics, vol. 28, no. 1, pp. 488–497, 2013.

[12] T. Sun, J. Wang, and X. Chen, “MTPA control for IPMSM drives based on virtual signal injection,” IEEE Transactions on Power Electronics, vol. 30, no. 9, pp. 5036–5045, 2015.

[13] G. D. Foo and M. F. Rahman, “Sensorless sliding-mode MTPA control of an IPM synchronous motor drive using a sliding-mode observer and HF signal injection,” IEEE Transactions on Industrial Electronics, vol. 57, no. 4, pp. 1270–1278, 2010.

[14] J.-W. Jung, V. Q. Leu, T. D. Do, E.-K. Kim, and H. H. Choi, “Fuzzy adaptive maximum-torque-per-ampere control of an interior permanent magnet synchronous motor for electric vehicle applications,” IEEE Transactions on Transportation Electrification, vol. 7, no. 3, pp. 1305–1315, 2021.

[15] S. Bolognani, S. Calligaro, and R. Petrella, “Adaptive flux-weakening controller for interior permanent magnet synchronous motor drives,” IEEE Journal of Emerging and Selected Topics in Power Electronics, vol. 2, no. 2, pp. 236–248, 2014.

[16] N. Bedetti, S. Calligaro, and R. Petrella, “Analytical design and autotuning of adaptive flux-weakening voltage regulation loop in IPMSM drives,” IEEE Transactions on Industry Applications, vol. 56, no. 1, pp. 301–313, 2020.

[17] P. Pillay and R. Krishnan, “Modeling of permanent magnet motor drives,” IEEE Transactions on Industry Applications, vol. 24, no. 2, pp. 265–273, 1988.

[18] R. Krishnan, Permanent Magnet Synchronous and Brushless DC Motor Drives. Boca Raton, FL, USA: CRC Press, 2010.

[19] Z. Q. Zhu, “Torque ripple in fractional-slot permanent magnet machines,” IET Electric Power Applications, vol. 6, no. 9, pp. 561–582, 2012.

[20] M. Martinez, D. Reigosa, D. Fernandez, and F. Briz, “Comparative Analysis of High Frequency Signal Injection Based Torque Estimation Methods for SPMSM, IPMSM and SynRM,” Energies, vol. 13, no. 3, art. no. 592, 2020.

[21] R. H. Park, “Two-reaction theory of synchronous machines - Generalized method of analysis - Part I,” Transactions of the American Institute of Electrical Engineers, vol. 48, no. 3, pp. 716–727, 1929.

[22] E. Clarke, Circuit Analysis of A-C Power Systems, vol. 1. New York, NY, USA: Wiley, 1943.

[23] D. W. Novotny and T. A. Lipo, Vector Control and Dynamics of AC Drives. Oxford, U.K.: Oxford University Press, 1996.

[24] W. Leonhard, Control of Electrical Drives, 3rd ed. Berlin, Germany: Springer, 2001.

[25] N. Bianchi and S. Bolognani, “Parameters and volt-ampere ratings of a synchronous motor drive for flux-weakening applications,” IEEE Transactions on Power Electronics, vol. 12, no. 5, pp. 895–903, 1997.

[26] A. M. El-Refaie and T. M. Jahns, “Optimal flux weakening in surface PM machines using fractional-slot concentrated windings,” IEEE Transactions on Industry Applications, vol. 41, no. 3, pp. 790–800, 2005.

[27] G. De Boni, L. Mantione, M. Minervini, and L. Frosini, “Look-Up Table Based Reduced Order Model of Synchronous Motors for Digital Twin Applications,” in Proc. IEEE Workshop on Electrical Machines Design, Control and Diagnosis (WEMDCD), 2025.

[28] MathWorks, “scatteredInterpolant,” MATLAB Documentation, 2026.

[29] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, Feedback Control of Dynamic Systems, 7th ed. Harlow, U.K.: Pearson, 2014.

[30] K. J. Åström and T. Hägglund, Advanced PID Control. Research Triangle Park, NC, USA: ISA, 2006.

[31] D. G. Holmes and T. A. Lipo, Pulse Width Modulation for Power Converters: Principles and Practice. Hoboken, NJ, USA: Wiley-IEEE Press, 2003.

[32] United Nations Economic Commission for Europe (UNECE), United Nations Global Technical Regulation No. 15: Worldwide harmonized Light vehicles Test Procedure (WLTP), Amendment 6, ECE/TRANS/180/Add.15/Amend.6, 2021.

Appendix - Annotated MATLAB Pipeline Script

```
%% ========================================================
%  UNIFIED FEM -> SIMULINK LUT PIPELINE
%
%  Change the filename below to select the motor dataset.
%
%  Final outputs:
%        ID_PEAK, IQ_PEAK, theta_m_deg, FD, FQ, T
%% ========================================================
%% ========================================================
%  CLEAR WORKSPACES
%% ========================================================
evalin('base','clearvars');
mdl = bdroot;
if ~isempty(mdl)
   try
       mws = get_param(mdl,'ModelWorkspace');
       clear(mws);
   catch
   end
end
clearvars -except filename make_debug_plots;
clc;
%% ========================================================
%  SETTINGS
%% ========================================================
% filename = 'Prius_emotor_data_LuT_DQFrame_IdIqRotorAngleInputs.mat';
filename = 'emotor_data_LuT_DQFrame_IdIqRotorAngleInputs.mat';
```

```
S = load(filename);
make_debug_plots = true;
fprintf('Loaded file: %s\n', filename);
%% ============================================================
% NORMALIZE SCALAR MOTOR PARAMETERS FROM LOADED FILE
% ============================================================
% If the MAT file was loaded into a struct, expose fields locally.
% Example:
%   S = load(filename);
% then use S.<field>
if exist('S','var') && isstruct(S)
   src = S;
else
   src = struct();
   vars_loaded = who;
   for ii = 1:numel(vars_loaded)
       try
           src.(vars_loaded{ii}) = eval(vars_loaded{ii});
       catch
       end
   end
end
% Resistance
if ~exist('R_phase','var')
   if isfield(src,'R_phase')
       R_phase = src.R_phase;
   elseif isfield(src,'Rs')
       R_phase = src.Rs;
   else
       error('Motor resistance not found: expected R_phase or Rs.');
```

```
    end
end
% End winding inductance
if ~exist('L_end_winding','var')
    if isfield(src,'L_end_winding')
        L_end_winding = src.L_end_winding;
    else
        warning('L_end_winding not found -> using 0');
        L_end_winding = 0;
    end
end
% Inertia
if ~exist('J_inertia','var')
    if isfield(src,'J_inertia')
        J_inertia = src.J_inertia;
    elseif isfield(src,'J')
        J_inertia = src.J;
    else
        error('Rotor inertia not found: expected J_inertia or J.');
    end
end
% Damping
if ~exist('b_damping_coef','var')
    if isfield(src,'b_damping_coef')
        b_damping_coef = src.b_damping_coef;
    elseif isfield(src,'Dm')
        b_damping_coef = src.Dm;
    else
        b_damping_coef = NaN;
    end
```

```
end
% Pole pairs
if ~exist('p','var')
   if isfield(src,'num_pole_pairs')
       p = src.num_pole_pairs;
   elseif isfield(src,'p')
       p = src.p;
   else
       error('Pole-pair count not found: expected num_pole_pairs or p.');
   end
end
% Force scalar shape
R_phase       = double(R_phase(1));
L_end_winding = double(L_end_winding(1));
J_inertia     = double(J_inertia(1));
b_damping_coef = double(b_damping_coef(1));
p             = double(p(1));
fprintf('\n--- Normalized scalar parameters ---\n');
fprintf('R_phase       = %.8g Ohm\n', R_phase);
fprintf('L_end_winding = %.8g H\n',   L_end_winding);
fprintf('J_inertia     = %.8g kg*m^2\n', J_inertia);
fprintf('b_damping_coef= %.8g N*m*s/rad\n', b_damping_coef);
fprintf('p             = %g\n', p);
required_core                                                              =
{'ID_PEAK','IQ_PEAK','ANGPOS_ROTOR_DEG','FLUX_D','FLUX_Q','TORQUE','num_pole_
pairs'};
for k = 1:numel(required_core)
      assert(isfield(S,  required_core{k}),  'Missing  required  field:  %s',
required_core{k});
end
% Raw exported axes
```

```
A = S.ID_PEAK(:).';

B = S.IQ_PEAK(:).';

theta_m_deg = S.ANGPOS_ROTOR_DEG(:).';

p = double(S.num_pole_pairs);

if numel(p) > 1, p = p(1); end

nA  = numel(A);

nB  = numel(B);

nTh = numel(theta_m_deg);

assert(numel(S.FLUX_D) == nA*nB*nTh, 'FLUX_D size mismatch');

assert(numel(S.FLUX_Q) == nA*nB*nTh, 'FLUX_Q size mismatch');

assert(numel(S.TORQUE) == nA*nB*nTh, 'TORQUE size mismatch');

%% ========================================================

%  OPTIONAL FIELDS

%% ========================================================

hasLdDynID  = isfield(S,'LD_DYN_vs_ID');

hasLdDynIQ  = isfield(S,'LD_DYN_vs_IQ');

hasLqDynID  = isfield(S,'LQ_DYN_vs_ID');

hasLqDynIQ  = isfield(S,'LQ_DYN_vs_IQ');

hasLdStatID = isfield(S,'LD_STAT_vs_ID');

hasLqStatIQ = isfield(S,'LQ_STAT_vs_IQ');

hasLossJ    = isfield(S,'LOSS_JOULE');

%% ========================================================

%  RESHAPE ALL AVAILABLE TABLES ON NATIVE EXPORTED AXES [nA, nB, nTh]

%% ========================================================

FD_native = reshape(S.FLUX_D, [nA, nB, nTh]);

FQ_native = reshape(S.FLUX_Q, [nA, nB, nTh]);

T_native  = reshape(S.TORQUE, [nA, nB, nTh]);

if hasLdDynID,   LdDyn_ID_native   = reshape(S.LD_DYN_vs_ID,   [nA, nB, nTh]);
end

if hasLdDynIQ,   LdDyn_IQ_native   = reshape(S.LD_DYN_vs_IQ,   [nA, nB, nTh]);
end
```

```
if hasLqDynID,   LqDyn_ID_native   = reshape(S.LQ_DYN_vs_ID,   [nA, nB, nTh]);
end
if hasLqDynIQ,   LqDyn_IQ_native   = reshape(S.LQ_DYN_vs_IQ,   [nA, nB, nTh]);
end
if hasLdStatID, LdStat_ID_native = reshape(S.LD_STAT_vs_ID, [nA, nB, nTh]);
end
if hasLqStatIQ, LqStat_IQ_native = reshape(S.LQ_STAT_vs_IQ, [nA, nB, nTh]);
end
if hasLossJ,     LossJ_native      = reshape(S.LOSS_JOULE,     [nA, nB, nTh]);
end
%% ============================================================
%  DATASET INTERPRETATION + AXIS DETECTION
%  Detect whether exported axes are already [id, iq] or swapped.
%  Expected source quadrant: id in [negative ... 0], iq in [0 ... positive]
%% ============================================================
if min(A) < 0 && max(A) <= 0 && min(B) >= 0
   id_vec = A;
   iq_vec = B;
   FD_raw = FD_native;
   FQ_raw = FQ_native;
   T_raw  = T_native;
   if hasLdDynID,  LdDyn_ID_raw  = LdDyn_ID_native;  end
   if hasLdDynIQ,  LdDyn_IQ_raw  = LdDyn_IQ_native;  end
   if hasLqDynID,  LqDyn_ID_raw  = LqDyn_ID_native;  end
   if hasLqDynIQ,  LqDyn_IQ_raw  = LqDyn_IQ_native;  end
   if hasLdStatID, LdStat_ID_raw = LdStat_ID_native; end
   if hasLqStatIQ, LqStat_IQ_raw = LqStat_IQ_native; end
   if hasLossJ,    LossJ_raw     = LossJ_native;     end
   axes_permuted = false;
   fprintf('Axis detection: exported A=id, B=iq.\n');
elseif min(B) < 0 && max(B) <= 0 && min(A) >= 0
   id_vec = B;
```

```
    iq_vec = A;
    FD_raw = permute(FD_native, [2 1 3]);
    FQ_raw = permute(FQ_native, [2 1 3]);
    T_raw  = permute(T_native,  [2 1 3]);
    if hasLdDynID,  LdDyn_ID_raw  = permute(LdDyn_ID_native,  [2 1 3]); end
    if hasLdDynIQ,  LdDyn_IQ_raw  = permute(LdDyn_IQ_native,  [2 1 3]); end
    if hasLqDynID,  LqDyn_ID_raw  = permute(LqDyn_ID_native,  [2 1 3]); end
    if hasLqDynIQ,  LqDyn_IQ_raw  = permute(LqDyn_IQ_native,  [2 1 3]); end
    if hasLdStatID, LdStat_ID_raw = permute(LdStat_ID_native, [2 1 3]); end
    if hasLqStatIQ, LqStat_IQ_raw = permute(LqStat_IQ_native, [2 1 3]); end
    if hasLossJ,    LossJ_raw     = permute(LossJ_native,     [2 1 3]); end
    axes_permuted = true;
    fprintf('Axis detection: exported B=id, A=iq. Arrays permuted to [id, iq,
theta].\n');
else
    % Already likely full quadrant, try direct interpretation first.
    id_vec = A;
    iq_vec = B;
    FD_raw = FD_native;
    FQ_raw = FQ_native;
    T_raw  = T_native;
    if hasLdDynID,  LdDyn_ID_raw  = LdDyn_ID_native;  end
    if hasLdDynIQ,  LdDyn_IQ_raw  = LdDyn_IQ_native;  end
    if hasLqDynID,  LqDyn_ID_raw  = LqDyn_ID_native;  end
    if hasLqDynIQ,  LqDyn_IQ_raw  = LqDyn_IQ_native;  end
    if hasLdStatID, LdStat_ID_raw = LdStat_ID_native; end
    if hasLqStatIQ, LqStat_IQ_raw = LqStat_IQ_native; end
    if hasLossJ,    LossJ_raw     = LossJ_native;     end
    axes_permuted = false;
    fprintf('Axis detection: non-1Q dataset, keeping exported axis order and
validating later.\n');
```

```
end
%% ============================================================
%  SORT BREAKPOINTS AND REORDER LUTS
%% ============================================================
[id_vec, idx_id] = sort(id_vec, 'ascend');
[iq_vec, idx_iq] = sort(iq_vec, 'ascend');
[theta_m_deg, idx_th] = sort(theta_m_deg, 'ascend');
reorder3 = @(X) X(idx_id, idx_iq, idx_th);
FD_raw = reorder3(FD_raw);
FQ_raw = reorder3(FQ_raw);
T_raw  = reorder3(T_raw);
if hasLdDynID,  LdDyn_ID_raw  = reorder3(LdDyn_ID_raw);  end
if hasLdDynIQ,  LdDyn_IQ_raw  = reorder3(LdDyn_IQ_raw);  end
if hasLqDynID,  LqDyn_ID_raw  = reorder3(LqDyn_ID_raw);  end
if hasLqDynIQ,  LqDyn_IQ_raw  = reorder3(LqDyn_IQ_raw);  end
if hasLdStatID, LdStat_ID_raw = reorder3(LdStat_ID_raw); end
if hasLqStatIQ, LqStat_IQ_raw = reorder3(LqStat_IQ_raw); end
if hasLossJ,    LossJ_raw     = reorder3(LossJ_raw);     end
assert(all(diff(id_vec) > 0), 'id axis is not strictly ascending');
assert(all(diff(iq_vec) > 0), 'iq axis is not strictly ascending');
assert(all(diff(theta_m_deg) > 0), 'theta axis is not strictly ascending');
%% ============================================================
%  FLUX CHECK (only needed for source 1Q datasets)
%  Expèected at (id=0, iq=0),
%       Fd should dominate the magnet contribution
%       while Fq should be near zero.
%% ============================================================
[~, id0] = min(abs(id_vec));
[~, iq0] = min(abs(iq_vec));
phi_D_00 = mean(FD_raw(id0, iq0, :), 'omitnan');
```

```
phi_Q_00 = mean(FQ_raw(id0, iq0, :), 'omitnan');

full_quadrant_source = any(id_vec < 0) && any(id_vec > 0) && any(iq_vec < 0)
&& any(iq_vec > 0);

flux_channels_swapped = false;

if ~full_quadrant_source && abs(phi_Q_00) > abs(phi_D_00)

   tmp = FD_raw; FD_raw = FQ_raw; FQ_raw = tmp;

   flux_channels_swapped = true;

    fprintf('Flux-channel correction applied: exported FLUX_Q interpreted as
lambda_d.\n');

else

   fprintf('No flux-channel swap applied.\n');

end

%% ========================================================

%  NaN FILLING (if needed)

%% ========================================================

need_nan_fill    =    any(isnan(FD_raw(:)))    ||    any(isnan(FQ_raw(:)))    ||
any(isnan(T_raw(:)));

if hasLdDynID,  need_nan_fill = need_nan_fill || any(isnan(LdDyn_ID_raw(:)));
end

if hasLdDynIQ,  need_nan_fill = need_nan_fill || any(isnan(LdDyn_IQ_raw(:)));
end

if hasLqDynID,  need_nan_fill = need_nan_fill || any(isnan(LqDyn_ID_raw(:)));
end

if hasLqDynIQ,  need_nan_fill = need_nan_fill || any(isnan(LqDyn_IQ_raw(:)));
end

if       hasLdStatID,       need_nan_fill       =       need_nan_fill       ||
any(isnan(LdStat_ID_raw(:))); end

if       hasLqStatIQ,       need_nan_fill       =       need_nan_fill       ||
any(isnan(LqStat_IQ_raw(:))); end

if hasLossJ,      need_nan_fill = need_nan_fill || any(isnan(LossJ_raw(:)));
end

if need_nan_fill
```

```
      fprintf('NaN filling required: applying slice-by-slice scattered
interpolation.\n');
   [IQ_grid, ID_grid] = meshgrid(1:numel(iq_vec), 1:numel(id_vec));
   FD_raw = fill_nan_2d_per_theta(FD_raw, ID_grid, IQ_grid);
   FQ_raw = fill_nan_2d_per_theta(FQ_raw, ID_grid, IQ_grid);
   T_raw  = fill_nan_2d_per_theta(T_raw,  ID_grid, IQ_grid);
     if hasLdDynID,    LdDyn_ID_raw    = fill_nan_2d_per_theta(LdDyn_ID_raw,
ID_grid, IQ_grid); end
     if hasLdDynIQ,    LdDyn_IQ_raw    = fill_nan_2d_per_theta(LdDyn_IQ_raw,
ID_grid, IQ_grid); end
     if hasLqDynID,    LqDyn_ID_raw    = fill_nan_2d_per_theta(LqDyn_ID_raw,
ID_grid, IQ_grid); end
     if hasLqDynIQ,    LqDyn_IQ_raw    = fill_nan_2d_per_theta(LqDyn_IQ_raw,
ID_grid, IQ_grid); end
     if hasLdStatID,  LdStat_ID_raw  = fill_nan_2d_per_theta(LdStat_ID_raw,
ID_grid, IQ_grid); end
     if hasLqStatIQ,  LqStat_IQ_raw  = fill_nan_2d_per_theta(LqStat_IQ_raw,
ID_grid, IQ_grid); end
     if hasLossJ,       LossJ_raw        = fill_nan_2d_per_theta(LossJ_raw,
ID_grid, IQ_grid); end
else
   fprintf('NaN filling not required.\n');
end
%% ========================================================
%  THETA TILING TO FULL ELECTRICAL PERIOD (if needed)
%% ========================================================
elec_period = 360 / p;
theta_span  = theta_m_deg(end) - theta_m_deg(1);
ratio       = elec_period / theta_span;
need_theta_tiling = ratio > 1.5;
junction_blending_applied = false;
fprintf('Theta span = %.6f deg | Electrical period = %.6f deg | ratio =
%.6f\n', ...
```

```
   theta_span, elec_period, ratio);
if need_theta_tiling
   assert(abs(ratio - 2) < 0.3, ...
          'Theta span does not look like half an electrical period. Check
rotor-angle definition.');
   dtheta = theta_m_deg(2) - theta_m_deg(1);
   theta_m_deg = [theta_m_deg, theta_m_deg(2:end) + theta_span];
   FD_raw = cat(3, FD_raw, FD_raw(:,:,2:end));
   FQ_raw = cat(3, FQ_raw, FQ_raw(:,:,2:end));
   T_raw  = cat(3, T_raw,  T_raw(:,:,2:end));
          if   hasLdDynID,      LdDyn_ID_raw      =   cat(3,   LdDyn_ID_raw,
LdDyn_ID_raw(:,:,2:end)); end
          if   hasLdDynIQ,      LdDyn_IQ_raw      =   cat(3,   LdDyn_IQ_raw,
LdDyn_IQ_raw(:,:,2:end)); end
          if   hasLqDynID,      LqDyn_ID_raw      =   cat(3,   LqDyn_ID_raw,
LqDyn_ID_raw(:,:,2:end)); end
          if   hasLqDynIQ,      LqDyn_IQ_raw      =   cat(3,   LqDyn_IQ_raw,
LqDyn_IQ_raw(:,:,2:end)); end
           if   hasLdStatID,    LdStat_ID_raw    =   cat(3,   LdStat_ID_raw,
LdStat_ID_raw(:,:,2:end)); end
           if   hasLqStatIQ,    LqStat_IQ_raw    =   cat(3,   LqStat_IQ_raw,
LqStat_IQ_raw(:,:,2:end)); end
        if   hasLossJ,          LossJ_raw             =   cat(3,   LossJ_raw,
LossJ_raw(:,:,2:end)); end
   fprintf('Theta tiling applied (2x).\n');
   %% ======================================================
   %  JUNCTION BLENDING
   %% ======================================================
   junc = round((numel(theta_m_deg) + 1)/2);
   jump_T  = max(abs(T_raw(:,:,junc) - T_raw(:,:,junc+1)), [], 'all');
   range_T = max(T_raw(:), [], 'omitnan') - min(T_raw(:), [], 'omitnan');
   tol_rel = 0.02;
```

```
    if jump_T > tol_rel * max(range_T, eps)
        n_blend = max(2, round(5 / max(dtheta, eps)));
        FD_raw = blend_junction_crossfade(FD_raw, n_blend);
        FQ_raw = blend_junction_crossfade(FQ_raw, n_blend);
        T_raw  = blend_junction_crossfade(T_raw,  n_blend);
        if hasLdDynID,  LdDyn_ID_raw  = blend_junction_crossfade(LdDyn_ID_raw,
n_blend); end
        if hasLdDynIQ,  LdDyn_IQ_raw  = blend_junction_crossfade(LdDyn_IQ_raw,
n_blend); end
        if hasLqDynID,  LqDyn_ID_raw  = blend_junction_crossfade(LqDyn_ID_raw,
n_blend); end
        if hasLqDynIQ,  LqDyn_IQ_raw  = blend_junction_crossfade(LqDyn_IQ_raw,
n_blend); end
                                    if     hasLdStatID,     LdStat_ID_raw     =
blend_junction_crossfade(LdStat_ID_raw, n_blend); end
                                    if     hasLqStatIQ,     LqStat_IQ_raw     =
blend_junction_crossfade(LqStat_IQ_raw, n_blend); end
         if hasLossJ,     LossJ_raw      = blend_junction_crossfade(LossJ_raw,
n_blend); end
        junction_blending_applied = true;
        fprintf('Junction blending applied.\n');
    else
        fprintf('Junction blending not required.\n');
    end
else
    fprintf('Theta tiling not required.\n');
    fprintf('Junction blending not required.\n');
end
%% ============================================================
%  FOUR-QUADRANT RECONSTRUCTION (if needed)
%% ============================================================
```

```
need_4q_reconstruction = ~(any(id_vec < 0) && any(id_vec > 0) && any(iq_vec < 0) && any(iq_vec > 0));
if need_4q_reconstruction
   fprintf('Four-quadrant reconstruction required.\n');
   [~, id0] = min(abs(id_vec));
   [~, iq0] = min(abs(iq_vec));
   % ---- Extend over negative iq ------------------------------------
   iq_pos = iq_vec(iq0+1:end);
   iq_neg = -fliplr(iq_pos);
   IQ_PEAK = [iq_neg, iq_vec];
   FD_iq = cat(2,  flip(FD_raw(:, iq0+1:end, :), 2), FD_raw);
   FQ_iq = cat(2, -flip(FQ_raw(:, iq0+1:end, :), 2), FQ_raw);
   T_iq  = cat(2, -flip(T_raw(:,  iq0+1:end, :), 2), T_raw);
   if hasLdDynID,  LdDyn_ID_iq  = cat(2, flip(LdDyn_ID_raw(:, iq0+1:end, :), 2),  LdDyn_ID_raw);  end
   if hasLdDynIQ,  LdDyn_IQ_iq  = cat(2, flip(LdDyn_IQ_raw(:, iq0+1:end, :), 2),  LdDyn_IQ_raw);  end
   if hasLqDynID,  LqDyn_ID_iq  = cat(2, flip(LqDyn_ID_raw(:, iq0+1:end, :), 2),  LqDyn_ID_raw);  end
   if hasLqDynIQ,  LqDyn_IQ_iq  = cat(2, flip(LqDyn_IQ_raw(:, iq0+1:end, :), 2),  LqDyn_IQ_raw);  end
   if hasLdStatID, LdStat_ID_iq = cat(2, flip(LdStat_ID_raw(:,iq0+1:end, :), 2), LdStat_ID_raw); end
   if hasLqStatIQ, LqStat_IQ_iq = cat(2, flip(LqStat_IQ_raw(:,iq0+1:end, :), 2), LqStat_IQ_raw); end
   if hasLossJ,    LossJ_iq     = cat(2, flip(LossJ_raw(:,iq0+1:end, :), 2), LossJ_raw);     end
   [~, iq0_4q] = min(abs(IQ_PEAK));
   FQ_iq(:, iq0_4q, :) = 0;
   T_iq(:,  iq0_4q, :) = 0;
   if iq0_4q > 1 && iq0_4q < numel(IQ_PEAK)
           FD_iq(:, iq0_4q, :) = 0.5 * (FD_iq(:, iq0_4q-1, :) + FD_iq(:, iq0_4q+1, :));
```

```
        if hasLdDynID,   LdDyn_ID_iq(:, iq0_4q, :)   = 0.5 * (LdDyn_ID_iq(:,
iq0_4q-1, :)  + LdDyn_ID_iq(:, iq0_4q+1, :));  end

        if hasLdDynIQ,   LdDyn_IQ_iq(:, iq0_4q, :)   = 0.5 * (LdDyn_IQ_iq(:,
iq0_4q-1, :)  + LdDyn_IQ_iq(:, iq0_4q+1, :));  end

        if hasLqDynID,   LqDyn_ID_iq(:, iq0_4q, :)   = 0.5 * (LqDyn_ID_iq(:,
iq0_4q-1, :)  + LqDyn_ID_iq(:, iq0_4q+1, :));  end

        if hasLqDynIQ,   LqDyn_IQ_iq(:, iq0_4q, :)   = 0.5 * (LqDyn_IQ_iq(:,
iq0_4q-1, :)  + LqDyn_IQ_iq(:, iq0_4q+1, :));  end

        if hasLdStatID, LdStat_ID_iq(:, iq0_4q, :) = 0.5 * (LdStat_ID_iq(:,
iq0_4q-1, :) + LdStat_ID_iq(:, iq0_4q+1, :)); end

        if hasLqStatIQ, LqStat_IQ_iq(:, iq0_4q, :) = 0.5 * (LqStat_IQ_iq(:,
iq0_4q-1, :) + LqStat_IQ_iq(:, iq0_4q+1, :)); end

         if hasLossJ,     LossJ_iq(:, iq0_4q, :)      = 0.5 * (LossJ_iq(:,
iq0_4q-1, :)     + LossJ_iq(:, iq0_4q+1, :));     end

   end

   % ---- Extend over positive id ---------------------------------------

   id_neg = id_vec(1:id0-1);

   id_pos = -fliplr(id_neg);

   ID_PEAK = [id_vec, id_pos];

   nPos = numel(id_pos);

   % Fd continuation using mild slope-limited extrapolation

   FD_pos = zeros(nPos, size(FD_iq,2), size(FD_iq,3));

   alpha = 0.2;

      slope_FD  =  (FD_iq(id0,:,:)  -  FD_iq(id0-1,:,:))  /  (id_vec(id0)  -
id_vec(id0-1));

   for kk = 1:nPos

       delta = id_pos(kk) - id_vec(id0);

       FD_pos(kk,:,:) = FD_iq(id0,:,:) + alpha * slope_FD * delta;

   end

   FD = cat(1, FD_iq, FD_pos);

   FQ = cat(1, FQ_iq, flip(FQ_iq(1:id0-1,:,:), 1));

   T  = cat(1, T_iq,  flip(T_iq(1:id0-1,:,:),  1));
```

```
            if    hasLdDynID,        LdDyn_ID    =    cat(1,    LdDyn_ID_iq,
flip(LdDyn_ID_iq(1:id0-1,:,:),  1)); end
            if    hasLdDynIQ,        LdDyn_IQ    =    cat(1,    LdDyn_IQ_iq,
flip(LdDyn_IQ_iq(1:id0-1,:,:),  1)); end
            if    hasLqDynID,        LqDyn_ID    =    cat(1,    LqDyn_ID_iq,
flip(LqDyn_ID_iq(1:id0-1,:,:),  1)); end
            if    hasLqDynIQ,        LqDyn_IQ    =    cat(1,    LqDyn_IQ_iq,
flip(LqDyn_IQ_iq(1:id0-1,:,:),  1)); end
             if    hasLdStatID,    LdStat_ID    =    cat(1,    LdStat_ID_iq,
flip(LdStat_ID_iq(1:id0-1,:,:), 1)); end
             if    hasLqStatIQ,    LqStat_IQ    =    cat(1,    LqStat_IQ_iq,
flip(LqStat_IQ_iq(1:id0-1,:,:), 1)); end
         if   hasLossJ,            LossJ              =   cat(1,   LossJ_iq,
flip(LossJ_iq(1:id0-1,:,:),     1)); end
else
   fprintf('Four-quadrant reconstruction not required.\n');
   ID_PEAK = id_vec;
   IQ_PEAK = iq_vec;
   FD = FD_raw;
   FQ = FQ_raw;
   T  = T_raw;
   if hasLdDynID,  LdDyn_ID = LdDyn_ID_raw;  end
   if hasLdDynIQ,  LdDyn_IQ = LdDyn_IQ_raw;  end
   if hasLqDynID,  LqDyn_ID = LqDyn_ID_raw;  end
   if hasLqDynIQ,  LqDyn_IQ = LqDyn_IQ_raw;  end
   if hasLdStatID, LdStat_ID = LdStat_ID_raw; end
   if hasLqStatIQ, LqStat_IQ = LqStat_IQ_raw; end
   if hasLossJ,    LossJ     = LossJ_raw;     end
end
%% ========================================================
%  ENFORCE POSITIVITY WHERE REQUIRED FOR INDUCTANCE TABLES
%% ========================================================
```

```
pos_eps = 1e-12;
if hasLdDynID,  LdDyn_ID = max(LdDyn_ID, pos_eps);  end
if hasLdDynIQ,  LdDyn_IQ = max(LdDyn_IQ, -inf);     end
if hasLqDynID,  LqDyn_ID = max(LqDyn_ID, -inf);     end
if hasLqDynIQ,  LqDyn_IQ = max(LqDyn_IQ, pos_eps);  end
if hasLdStatID, LdStat_ID = max(LdStat_ID, pos_eps); end
if hasLqStatIQ, LqStat_IQ = max(LqStat_IQ, pos_eps); end
%% ========================================================
%  PERIODIC CLOSURE (always required)
%% ========================================================
FD(:,:,end) = FD(:,:,1);
FQ(:,:,end) = FQ(:,:,1);
T(:,:,end)  = T(:,:,1);
if hasLdDynID,  LdDyn_ID(:,:,end)  = LdDyn_ID(:,:,1);  end
if hasLdDynIQ,  LdDyn_IQ(:,:,end)  = LdDyn_IQ(:,:,1);  end
if hasLqDynID,  LqDyn_ID(:,:,end)  = LqDyn_ID(:,:,1);  end
if hasLqDynIQ,  LqDyn_IQ(:,:,end)  = LqDyn_IQ(:,:,1);  end
if hasLdStatID, LdStat_ID(:,:,end) = LdStat_ID(:,:,1); end
if hasLqStatIQ, LqStat_IQ(:,:,end) = LqStat_IQ(:,:,1); end
if hasLossJ,    LossJ(:,:,end)     = LossJ(:,:,1);     end
%% ========================================================
% FINAL EXPORT NAMES FOR SIMULINK
%% ========================================================
ANGPOS_ROTOR_DEG = theta_m_deg;
ID_RMS = ID_PEAK / sqrt(2);
IQ_RMS = IQ_PEAK / sqrt(2);
Iq_max = 0.85 * max(abs(IQ_PEAK));
Id_max = 0.85 * max(abs(ID_PEAK));
% Optional machine parameters if present
if          isfield(S,'stator_phase_resistance'),          R_phase          =
double(S.stator_phase_resistance); end
```

```
if      isfield(S,'end_winding_inductance'),           L_end                 =
double(S.end_winding_inductance);  end

if isfield(S,'b_damping_coef'),           b_damp  = double(S.b_damping_coef);
end

if   isfield(S,'rotor_moment_inertia'),             J                       =
double(S.rotor_moment_inertia);    end

%% ============================================================

% FINAL VALIDATION

%% ============================================================

assert(all(diff(ID_PEAK) > 0), 'Final ID_PEAK not strictly ascending');

assert(all(diff(IQ_PEAK) > 0), 'Final IQ_PEAK not strictly ascending');

assert(all(diff(ANGPOS_ROTOR_DEG) > 0), 'Final ANGPOS_ROTOR_DEG not strictly
ascending');

assert(isequal(size(FD),           [numel(ID_PEAK),           numel(IQ_PEAK),
numel(ANGPOS_ROTOR_DEG)]), 'FD size mismatch');

assert(isequal(size(FQ),           [numel(ID_PEAK),           numel(IQ_PEAK),
numel(ANGPOS_ROTOR_DEG)]), 'FQ size mismatch');

assert(isequal(size(T),               [numel(ID_PEAK),        numel(IQ_PEAK),
numel(ANGPOS_ROTOR_DEG)]), 'T size mismatch');

assert(all(isfinite(FD(:))), 'FD contains non-finite values');

assert(all(isfinite(FQ(:))), 'FQ contains non-finite values');

assert(all(isfinite(T(:))),  'T contains non-finite values');

assert(max(abs(FD(:,:,end) - FD(:,:,1)), [], 'all') == 0, 'FD is not exactly
periodic');

assert(max(abs(FQ(:,:,end) - FQ(:,:,1)), [], 'all') == 0, 'FQ is not exactly
periodic');

assert(max(abs(T(:,:,end)   - T(:,:,1)),   [], 'all') == 0, 'T is not exactly
periodic');

fprintf('\n================ FINAL SUMMARY ================\n');

fprintf('File                          : %s\n', filename);

fprintf('Pole pairs                    : %d\n', p);

fprintf('Axes permuted                 : %d\n', axes_permuted);

fprintf('Flux channels swapped         : %d\n', flux_channels_swapped);
```

```
fprintf('NaN filling applied          : %d\n', need_nan_fill);
fprintf('Theta tiling applied         : %d\n', need_theta_tiling);
fprintf('Junction blending applied    : %d\n', junction_blending_applied);
fprintf('4Q reconstruction applied    : %d\n', need_4q_reconstruction);
fprintf('Final ID range [A]           : [%.6f, %.6f] (%d pts)\n', ID_PEAK(1),
ID_PEAK(end), numel(ID_PEAK));
fprintf('Final IQ range [A]           : [%.6f, %.6f] (%d pts)\n', IQ_PEAK(1),
IQ_PEAK(end), numel(IQ_PEAK));
fprintf('Final theta range [deg]           : [%.6f, %.6f] (%d pts)\n',
ANGPOS_ROTOR_DEG(1), ANGPOS_ROTOR_DEG(end), numel(ANGPOS_ROTOR_DEG));
fprintf('FD / FQ / T size                   : [%d %d %d]\n', size(FD,1),
size(FD,2), size(FD,3));
fprintf('================================================\n');
%% ========================================================
% OPTIONAL DEBUG PLOTS
%% ========================================================
if make_debug_plots
   k_vis = round(numel(ANGPOS_ROTOR_DEG)/2);
   figure('Name','Final electromagnetic LUTs');
   tiledlayout(1,3,'TileSpacing','compact','Padding','compact');
   nexttile; surf(ID_PEAK, IQ_PEAK, FD(:,:,k_vis).', 'EdgeColor','none');
      title(sprintf('FD @ theta = %.2f deg', ANGPOS_ROTOR_DEG(k_vis)));
xlabel('i_d [A]'); ylabel('i_q [A]'); zlabel('FD [Wb]'); view([-40 30]); grid
on; box on;
   nexttile; surf(ID_PEAK, IQ_PEAK, FQ(:,:,k_vis).', 'EdgeColor','none');
      title(sprintf('FQ @ theta = %.2f deg', ANGPOS_ROTOR_DEG(k_vis)));
xlabel('i_d [A]'); ylabel('i_q [A]'); zlabel('FQ [Wb]'); view([-40 30]); grid
on; box on;
   nexttile; surf(ID_PEAK, IQ_PEAK, T(:,:,k_vis).', 'EdgeColor','none');
      title(sprintf('T @ theta = %.2f deg', ANGPOS_ROTOR_DEG(k_vis)));
xlabel('i_d [A]'); ylabel('i_q [A]'); zlabel('T [Nm]'); view([-40 30]); grid
on; box on;
end
```

```
%% ============================================================
% MODEL PARAMETERS
% ============================================================
% Inverter
Vdc  = 800;
Vmax = Vdc/2;
fc   = 10e3;
Ts   = 1/fc/100;
% Mechanical damping
% If you want to force a thesis-tuning value, keep Dm_fixed = true
Dm_fixed = true;
if Dm_fixed
   Dm = 0.02;
else
        if  exist('b_damping_coef','var')  &&  ~isempty(b_damping_coef)  &&
isfinite(b_damping_coef)
       Dm = b_damping_coef;
   else
          warning('b_damping_coef not found/invalid -> using fallback Dm =
0.02');
       Dm = 0.02;
   end
end
% Required variables check
reqVars                                                                =
{'ID_PEAK','IQ_PEAK','T','R_phase','p','J_inertia','L_end_winding',...
          'LdDyn_ID_raw','LqDyn_IQ_raw'};
for ii = 1:numel(reqVars)
   if ~exist(reqVars{ii},'var')
       error('Required variable "%s" not found in workspace.', reqVars{ii});
   end
```

```
end
% ------------------------------------------------------------
% Rated point for local Ld/Lq evaluation
% NOTE:
% The values below are Prius-specific legacy targets.
% They are kept for compatibility, but if not available in the
% current dataset the code falls back to the nearest available point.
% ------------------------------------------------------------
id_target = +11.785;
iq_target = +23.570;
iq_target_2 = +29.460;
[~, id_idx] = min(abs(ID_PEAK - id_target));
[~, iq_idx] = min(abs(IQ_PEAK - iq_target));
[~, iq_idx2] = min(abs(IQ_PEAK - iq_target_2));
id_rated = ID_PEAK(id_idx);
iq_rated = IQ_PEAK(iq_idx);
% Choose inductance maps consistent with the final active domain
if exist('LdDyn_ID','var') && exist('LqDyn_IQ','var') ...
            && size(LdDyn_ID,1) == numel(ID_PEAK) && size(LdDyn_ID,2) ==
numel(IQ_PEAK) ...
            && size(LqDyn_IQ,1) == numel(ID_PEAK) && size(LqDyn_IQ,2) ==
numel(IQ_PEAK)
   Ld_map_for_ctrl = LdDyn_ID;
   Lq_map_for_ctrl = LqDyn_IQ;
      fprintf('Using processed inductance maps for controller parameter
extraction.\n');
elseif exist('LdDyn_ID_raw','var') && exist('LqDyn_IQ_raw','var') ...
        && size(LdDyn_ID_raw,1) == numel(ID_PEAK) && size(LdDyn_ID_raw,2) ==
numel(IQ_PEAK) ...
        && size(LqDyn_IQ_raw,1) == numel(ID_PEAK) && size(LqDyn_IQ_raw,2) ==
numel(IQ_PEAK)
   Ld_map_for_ctrl = LdDyn_ID_raw;
```

```
    Lq_map_for_ctrl = LqDyn_IQ_raw;
        fprintf('Using raw inductance maps (already aligned with final domain).\n');
else
    error(['No inductance maps aligned with final ID/IQ domain. ', ...
           'You must reconstruct/export inductance LUTs consistently with the final 4Q domain.']);
end
Ld = mean(squeeze(Ld_map_for_ctrl(id_idx, iq_idx, :)), 'omitnan') + L_end_winding;
Lq = mean(squeeze(Lq_map_for_ctrl(id_idx, iq_idx, :)), 'omitnan') + L_end_winding;
fprintf('\n--- Inductance at selected operating point ---\n');
fprintf('id = %.6f A, iq = %.6f A\n', id_rated, iq_rated);
fprintf('Ld = %.4f mH\n', Ld*1000);
fprintf('Lq = %.4f mH\n', Lq*1000);
if isfinite(Ld) && isfinite(Lq) && Ld > 0
    fprintf('Lq/Ld saliency ratio = %.2f\n', Lq/Ld);
else
    fprintf('Lq/Ld saliency ratio = invalid\n');
end
if isnan(Ld) || ~isfinite(Ld) || Ld <= 0
    warning('Ld invalid -> using fallback 5.6e-3 H');
    Ld = 5.6e-3;
end
if isnan(Lq) || ~isfinite(Lq) || Lq <= 0
    warning('Lq invalid -> using fallback 8.9e-3 H');
    Lq = 8.9e-3;
end
%% ========================================================
% PI CURRENT CONTROLLERS
```

```
% ========================================================

wc_c  = 100;

Kp_id = Ld * wc_c;

Ki_id = R_phase * wc_c;

Kp_iq = Lq * wc_c;

Ki_iq = R_phase * wc_c;

%% ========================================================

% Kt FROM LUT

% ========================================================

T_mtpa  = mean(squeeze(T(id_idx, iq_idx , :)), 'omitnan');

T_mtpa2 = mean(squeeze(T(id_idx, iq_idx2, :)), 'omitnan');

dIq = IQ_PEAK(iq_idx2) - IQ_PEAK(iq_idx);

if abs(dIq) < eps

    warning('Kt calculation invalid because selected iq points coincide ->
using fallback later');

   Kt = NaN;

else

   Kt = (T_mtpa2 - T_mtpa) / dIq;

end

% Phi for VelocityController block

Phi = Kt/(1.5*p) - (Ld - Lq)*id_rated;

fprintf('T_mtpa = %.3f Nm   T_mtpa2 = %.3f Nm\n', T_mtpa, T_mtpa2);

fprintf('Kt     = %.5f Nm/A   Phi = %.5f Wb\n', Kt, Phi);

if isnan(Phi) || ~isfinite(Phi)

   Phi = 0.05;

   warning('Phi invalid -> fallback to 0.05 Wb');

end

if isnan(Kt) || ~isfinite(Kt) || Kt <= 0

   Kt = 1.5*p*Phi;

   warning('Kt invalid -> fallback from Phi');

end
```

```
%% ============================================================
% PI SPEED CONTROLLER
% ============================================================
wn_s   = 1.5;
zeta_s = 1.5;
Kp_s = (2*zeta_s*wn_s*J_inertia - Dm) / Kt;
Ki_s = (wn_s^2 * J_inertia) / Kt;
if ~isfinite(Kp_s) || ~isfinite(Ki_s)
   error('Invalid speed-controller gains computed.');
end
fprintf('\n--- PI Gains ---\n');
fprintf('Kp_id = %g   Ki_id = %g\n', Kp_id, Ki_id);
fprintf('Kp_iq = %g   Ki_iq = %g\n', Kp_iq, Ki_iq);
fprintf('Kp_s  = %g   Ki_s  = %g\n', Kp_s,  Ki_s);
fprintf('Phi   = %.5f Wb\n', Phi);
fprintf('Kt    = %.5f Nm/A\n', Kt);
%% ============================================================
% FLUX-WEAKENING / CURRENT REFERENCE TABLES
% ============================================================
bp_speed_rpm = [0 1000 2000 2400 3000 4000 6000];
bp_iq        = [0 1 2 3 4];
% Prius-specific legacy value retained for compatibility.
% If you want this block to become motor-adaptive later, this is one of the
% parameters that should be externalized or derived separately.
Param_BaseSpeed = 2390;
speed_max = max(bp_speed_rpm);
if speed_max <= Param_BaseSpeed
   k = 0;
else
   k = id_rated / (speed_max - Param_BaseSpeed);
```

```
end
IdLUT = zeros(numel(bp_speed_rpm), numel(bp_iq));
for i = 1:numel(bp_speed_rpm)
   if bp_speed_rpm(i) <= Param_BaseSpeed
       IdLUT(i,:) = id_rated;
   else
       IdLUT(i,:) = id_rated - k*(bp_speed_rpm(i) - Param_BaseSpeed);
   end
end
IqLUT = repmat(bp_iq(:)', numel(bp_speed_rpm), 1);
%% ============================================================
% LOAD WLTP
% ============================================================
% Choose one scenario only:
WLTP_case = 'Medium';   % 'Low' | 'Medium' | 'High' | 'ExtraHigh'
switch lower(WLTP_case)
   case 'low'
       S = load('WLTP_Low.mat');
   case 'medium'
       S = load('WLTP_Medium.mat');
   case 'high'
       S = load('WLTP_High.mat');
   case 'extrahigh'
       S = load('WLTP_ExtraHigh.mat');
   otherwise
       error('Unknown WLTP_case. Use Low, Medium, High, or ExtraHigh.');
end
if isfield(S,'wltp_speed')
   time     = S.wltp_speed(:,1);
   speed_ms = S.wltp_speed(:,2);
```

```
elseif isfield(S,'time') && isfield(S,'speed_ms')
   time     = S.time(:);
   speed_ms = S.speed_ms(:);
else
   error('WLTP file does not contain expected variables.');
end
r_wheel = 0.317;
G       = 3.267;
omega_mech = speed_ms * G / r_wheel;
omega_elec = omega_mech * p;
m   = 1400;
g   = 9.81;
mu  = 0.015;
rho = 1.225;
Cd  = 0.26;
Ar  = 2.10;
F_roll = mu * m * g;
F_aero = 0.5 * rho * Cd * Ar * speed_ms.^2;
T_res  = (r_wheel / G) .* (F_roll + F_aero);
wltp_speed = [time, speed_ms];
wltp_omega = [time, omega_mech];
wltp_Tres  = [time, T_res];
%% ============================================================
% EXPORT TO BASE WORKSPACE
% ============================================================
vars = who;
for k_export = 1:length(vars)
   assignin('base', vars{k_export}, eval(vars{k_export}));
end
disp('PMSM FEM DQ LUT (with theta) and model parameters loaded into
workspace');
```

```
%% ============================================================

% LOCAL FUNCTIONS

%% ============================================================

function X = fill_nan_2d_per_theta(X, ID_grid, IQ_grid)

   for kk = 1:size(X,3)

       M = X(:,:,kk);

       nan_mask = isnan(M);

       if ~any(nan_mask(:))

           continue;

       end

       valid_mask = ~nan_mask;

       xv = ID_grid(valid_mask);

       yv = IQ_grid(valid_mask);

       zv = M(valid_mask);

       if numel(zv) < 3

            error('Not enough valid points to fill NaNs in theta slice %d.',
kk);

       end

           F_nat  =  scatteredInterpolant(double(xv),  double(yv),  double(zv),
'natural', 'none');

                           M(nan_mask)    =    F_nat(double(ID_grid(nan_mask)),
double(IQ_grid(nan_mask)));

       nan_mask2 = isnan(M);

       if any(nan_mask2(:))

            F_near = scatteredInterpolant(double(xv), double(yv), double(zv),
'nearest', 'nearest');

                           M(nan_mask2)   =   F_near(double(ID_grid(nan_mask2)),
double(IQ_grid(nan_mask2)));

       end

       X(:,:,kk) = M;

   end

end
```

```
function X = blend_junction_crossfade(X, n_blend)
   nTh = size(X,3);
   junc = round((nTh + 1)/2);
   left_idx  = max(1, junc - n_blend + 1):junc;
   right_idx = (junc + 1):min(nTh, junc + n_blend);
   n = min(numel(left_idx), numel(right_idx));
   left_idx  = left_idx(end-n+1:end);
   right_idx = right_idx(1:n);
   for kk = 1:n
       beta = 0.5 * (1 - cos(pi*(kk-1)/max(n-1,1)));
       L = X(:,:,left_idx(kk));
       R = X(:,:,right_idx(kk));
       B = (1 - beta).*L + beta.*R;
       X(:,:,left_idx(kk))  = B;
       X(:,:,right_idx(kk)) = B;
   end
end
```