THESIS

SIMULATION AND HARDWARE VALIDATION OF METHODS FOR SYNCHRONIZATION OF CENTRAL-CONVERTER MULTI-MOTOR ELECTRIC ACTUATION SYSTEMS

Submitted by

Zane P. Miller

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2023

Master's Committee:

Advisor: James Cale

Edwin Chong
William Fairbank

ABSTRACT

SIMULATION AND HARDWARE VALIDATION OF METHODS FOR SYNCHRONIZATION OF CENTRAL-CONVERTER MULTI-MOTOR ELECTRIC ACTUATION SYSTEMS

Replacement of previously hydraulic and pneumatic drives with power-electronic drive systems to reduce weight and maintenance requirements is a current target of research in the aerospace industry. This includes electrification of thrust reverser actuation systems (TRAS), which redirect thrust produced by the aircraft's engines to aid with deceleration upon landing, reducing wear on the brakes. However, one challenge of developing an electromagnetic TRAS (EM-TRAS) is the requirement of speed and position synchronization of all motors in the system, despite unequal torque loading from differing wind forces. Use of a single ("central") power electronic converter to power a set of induction machines in parallel could potentially lower cost and weight requirements compared to the use of separate converters, but such a central-converter, multi-motor (CCMM) architecture requires some form of compensation for load torque differences. Previous research presented a synchronization methodology using closed-loop feedback control of variable stator resistances in parallel with each induction machine. This thesis builds on this research by presenting an alternative methodology that instead applies closed-loop feedback control to smaller-scale auxiliary converters for each motor line, coupled to the induction machines using transformers to apply adjustments to the stator voltage. This new methodology achieves similar synchronization performance with better energy efficiency, lowering power requirements for its use compared to the external resistance methodology. The author's contributions to construction of a testbed for aerospace actuation system research are also presented in this thesis, with applications including hardware validation of the external resistance CCMM EM-TRAS implementation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

The aerospace industry is a constant target for innovation, with researchers always seeking improvements to the efficiency, performance, and safety of aircraft. This includes the development of More Electric Aircraft (MEA): by replacing non-propulsive mechanical (e.g., hydraulic and pneumatic) systems with electrical systems, benefits can be obtained such as reduced weight and cost, improved performance, and more comprehensive diagnostics. One system in particular targeted for electrification is the thrust reverser actuation system (TRAS), which diverts the engine's thrust to reverse the force applied on the aircraft, reducing the time and length of runway required to slow down after landing and wear on the brakes. As illustrated in Figure 1.1, the exhaust is redirected by the TRAS to leave the nacelle in a forward direction, producing a backward thrust force.

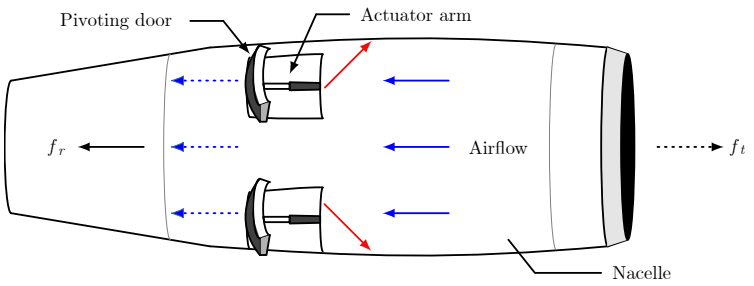


Figure 1.1: Basic diagram of a TRAS, showing normal (blue, dotted) and reversed (red, solid) airflow within the nacelle and thrust forces produced.

Operation of an electromechanical thrust reverser actuation system (EM-TRAS) requires electric drives and motors for supplying the force/torque necessary for deploying and stowing the thrust reversers. Different portions of the system would experience different external forces in operation, so the conceptually simplest method for keeping the speed and position of all motors in the system synchronized is to use a separate converter and drive for each motor, allowing for full individual control in an architecture known as distributed converter multi-motor (DCMM). This thesis builds on research exploring an alternative architecture, central converter multi-motor (CCMM), where a

single central converter is used to power all of the motors and the drive conditions the power input to each motor to achieve synchronization. A diagram of a CCMM EM-TRAS is shown in Figure 1.2. Use of CCMM architecture could reduce the size, weight, and cost of the EM-TRAS, making it an attractive option. [1]

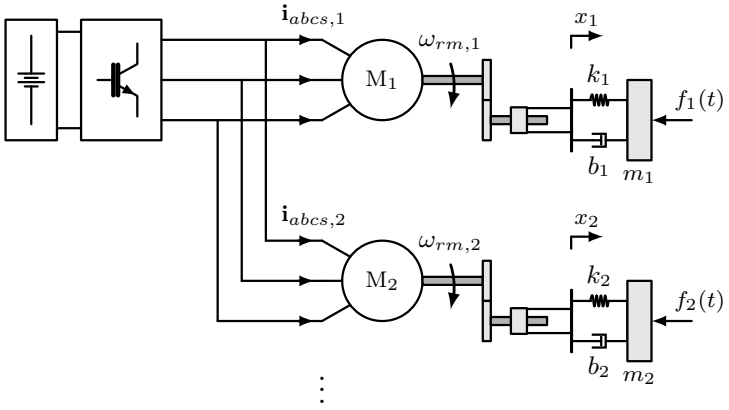


Figure 1.2: Notional depiction of CCMM architecture for EM-TRAS.

The implementation presented in [1] uses external stator resistance controlled by PWM switching to achieve speed and position synchronization. Disadvantages of this method include that rapidly switching the external resistance on and off produces large transient voltages that can damage equipment if not mitigated and a significant portion of power provided by the central converter must be dissipated in the external resistance to achieve synchronization (see Chapter 4). Better efficiency and performance could potentially be achieved through the use of external impedance including an inductive or capacitive component in addition to resistance. However, existing methods for electronic variable impedance control investigated by this researcher appear to be few in number, with some publicly documented only in patents that describe the method in terms too vague for replication, and one method requiring a separate voltage converter controlled using inline current readings [2], introducing complexity that may negate the advantages of CCMM architecture.

In the first part of this thesis, a novel CCMM implementation using low-power auxiliary converters for speed and position synchronization is presented. Chapter 2 explains the methodology for this implementation, including the principle of operation, the expected transient and steady-

state behavior, and a system diagram of a potential implementation. Chapter 3 presents results of simulations investigating this implementation using MATLAB and compares its performance and efficiency to that of the implementation presented in [1].

Chapter 4 constitutes the second part of this thesis, which details the author's contribution to a testbed designed for hardware emulation of various aerospace systems. The primary initial purpose of this testbed is hardware validation of the resistor-based CCMM EM-TRAS implementation presented in [1]. Design and construction of this testbed is explained in more detail in [3].

Plans and potential for future work building on the research presented in this thesis are detailed in Chapter 5. This includes nonlinear optimization of the novel CCMM implementation using a more detailed CCMM model and hardware validation of the implementation using the testbed. Finally, key findings and the author's specific contributions to the research presented here are summarized in Chapter 6.

Chapter 2

Background

This chapter begins with an overview of induction machines (IMs) along with the external stator resistance method for speed and position synchronization, referenced from [1]. Then, the novel auxiliary converter method is presented along with a full diagram of the circuitry required for implementation.

2.1 Induction Machines

An induction machine (IM) can be operated as a motor or a generator, though this thesis primarily focuses on motor operation. An IM contains stationary stator windings that magnetically couple with rotor windings inside the rotor when alternating current is applied, causing rotation. Roller bearings mounted on the shaft allow it to rotate freely while keeping a fixed gap distance between the stator and rotor. In this thesis, it is assumed for simplicity that the windings are sinusoidally distributed, the machine parameters are electrically symmetric, and the rotor windings are short-circuited at their ends (a "squirrel-cage" rotor).

The "electrical" angular velocity ω_r is related to the (actual) mechanical angular velocity as $\omega_r = ({}^{P}\!/_{2})\omega_{rm}$ where P is the number of magnetic poles. The electrical angular position θ_r at time t is defined as

$$\theta_r(t) = \int_0^t \omega_r(\tau)\, d\tau + \theta_r(0), \tag{2.1}$$

where $\theta_r(0)$ is the initial electrical angular position. Voltage equations for the IM are expressed as follows:

$$\begin{aligned} \mathbf{v}_{abcs} &= \mathbf{r}_s \mathbf{i}_{abcs} + p\boldsymbol{\lambda}_{abcs} \\ \mathbf{v}_{abcr} &= \mathbf{r}_r \mathbf{i}_{abcr} + p\boldsymbol{\lambda}_{abcr}, \end{aligned} \tag{2.2}$$

where $\mathbf{v}_{abcs(r)}$, $\mathbf{i}_{abcs(r)}$, and $\boldsymbol{\lambda}_{abcs(r)}$ represent the stator (motor) voltage, current, and flux linkages, respectively, while $\mathbf{r}_s$ and $\mathbf{r}_r$ are the stator and rotor resistance matrices

$$\mathbf{r}_s = \begin{bmatrix} r_s & 0 & 0 \\ 0 & r_s & 0 \\ 0 & 0 & r_s \end{bmatrix}, \quad \mathbf{r}_r = \begin{bmatrix} r_r & 0 & 0 \\ 0 & r_r & 0 \\ 0 & 0 & r_r \end{bmatrix}, \tag{2.3}$$

where r_s and r_r are the resistances of each stator and rotor winding, assumed to be equal for each phase (but generally $r_s \neq r_r$).

The flux linkages in (2.2) can be expressed as

$$\begin{aligned} \boldsymbol{\lambda}_{abcs} &= \mathbf{L}_s \mathbf{i}_{abcs} + \mathbf{L}_{sr} \mathbf{i}_{abcr} \\ \boldsymbol{\lambda}_{abcr} &= \mathbf{L}_r \mathbf{i}_{abcr} + \mathbf{L}_{sr}^{\top} \mathbf{i}_{abcs}, \end{aligned} \tag{2.4}$$

where the self-inductance matrices $\mathbf{L}_s$, $\mathbf{L}_r$ are

$$\begin{aligned} \mathbf{L}_s &= \begin{bmatrix} L_{ls} + L_{ms} & -\frac{1}{2}L_{ms} & -\frac{1}{2}L_{ms} \\ -\frac{1}{2}L_{ms} & L_{ls} + L_{ms} & -\frac{1}{2}L_{ms} \\ -\frac{1}{2}L_{ms} & -\frac{1}{2}L_{ms} & L_{ls} + L_{ms} \end{bmatrix}, \\ \mathbf{L}_r &= \begin{bmatrix} L_{lr} + L_{mr} & -\frac{1}{2}L_{mr} & -\frac{1}{2}L_{mr} \\ -\frac{1}{2}L_{mr} & L_{lr} + L_{mr} & -\frac{1}{2}L_{mr} \\ -\frac{1}{2}L_{mr} & -\frac{1}{2}L_{mr} & L_{lr} + L_{mr} \end{bmatrix} \end{aligned} \tag{2.5}$$

where L_{ls} and L_{lr} are the stator and rotor leakage inductances, and L_{ms} and L_{mr} are the stator and rotor magnetizing inductances. The mutual inductance matrix is

$$\mathbf{L}_{sr} = L_{sr} \begin{bmatrix} \cos\theta_r & \cos(\theta_r + \frac{2\pi}{3}) & \cos(\theta_r - \frac{2\pi}{3}) \\ \cos(\theta_r - \frac{2\pi}{3}) & \cos\theta_r & \cos(\theta_r + \frac{2\pi}{3}) \\ \cos(\theta_r + \frac{2\pi}{3}) & \cos(\theta_r - \frac{2\pi}{3}) & \cos\theta_r \end{bmatrix}, \tag{2.6}$$

where L_{sr} is the maximum mutual inductance between stator and rotor windings. To eliminate the angular dependence in the machine equations arising from (2.6), transformation of the equations to a rotating reference frame is employed. Transformation of the stationary variables to $qd0$ variables in the arbitrary rotating reference frame is achieved by the transformation

$$\mathbf{f}_{qd0s} = \mathbf{K}_s \mathbf{f}_{abcs}, \tag{2.7}$$

where $\mathbf{f}$ is any transformed quantity and the matrix $\mathbf{K}_s$ is defined as

$$\mathbf{K}_s(\theta) = \frac{2}{3} \begin{bmatrix} \cos\theta & \cos(\theta - \frac{2\pi}{3}) & \cos(\theta + \frac{2\pi}{3}) \\ \sin\theta & \sin(\theta - \frac{2\pi}{3}) & \sin(\theta + \frac{2\pi}{3}) \\ 1/2 & 1/2 & 1/2 \end{bmatrix}, \tag{2.8}$$

with the arbitrary reference frame position angle θ defined from the reference frame speed ω as

$$\theta(t) = \int_0^t \omega(\tau)\, d\tau + \theta(0). \tag{2.9}$$

Transformation of the rotor variables to $qd0$ variables in the arbitrary rotating reference frame is achieved by the transformation

$$\mathbf{f}_{qd0r} = \mathbf{K}_r \mathbf{f}_{abcr}, \tag{2.10}$$

where the matrix $\mathbf{K}_r$ is defined as

$$\mathbf{K}_r(\beta) = \mathbf{K}_s(\theta - \theta_r). \tag{2.11}$$

For ease of notation, variables are referred based on the turns ratio between stator and rotor windings. The voltage, flux linkages, currents, and electrical parameters of the rotor are referred to the

stator side as

$$\begin{gathered}\mathbf{v}'_{qdr} = \frac{N_s}{N_r}\mathbf{v}_{qdr}, \quad \boldsymbol{\lambda}'_{qdr} = \frac{N_s}{N_r}\boldsymbol{\lambda}_{qdr}, \quad \mathbf{i}'_{qdr} = \frac{N_r}{N_s}\mathbf{i}_{qdr}, \\ r'_r = \left(\frac{N_s}{N_r}\right)^2 r_r, \quad L'_{lr} = \left(\frac{N_s}{N_r}\right)^2 L_{lr},\end{gathered} \tag{2.12}$$

where N_s and N_r are the equivalent turns of the stator and rotor windings, respectively. After transformation of (2.2)–(2.4) to the arbitrary rotating reference frame using (2.7)–(2.11) and the referral of variables from (2.12) , the voltage equations in qd variables are expressed as

$$\begin{aligned}v_{qs} &= r_s i_{qs} + \omega\lambda_{ds} + p\lambda_{qs} \\ v_{ds} &= r_s i_{ds} - \omega\lambda_{qs} + p\lambda_{ds} \\ v_{qr} &= r'_r i'_{qr} + (\omega - \omega_r)\lambda'_{dr} + p\lambda'_{qr} \\ v_{dr} &= r'_r i'_{dr} - (\omega - \omega_r)\lambda'_{qr} + p\lambda'_{dr},\end{aligned} \tag{2.13}$$

and flux linkages in qd variables are expressed as

$$\begin{aligned}\lambda_{qs} &= L_{ls} i_{qs} + L_M(i_{qs} + i'_{qr}) \\ \lambda_{ds} &= L_{ls} i_{ds} + L_M(i_{ds} + i'_{dr}) \\ \lambda_{qr} &= L'_{lr} i'_{qr} + L_M(i_{qs} + i'_{qr}) \\ \lambda_{dr} &= L'_{lr} i'_{dr} + L_M(i_{qs} + i'_{qr}),\end{aligned} \tag{2.14}$$

where $L_M = (3/2)L_{ms}$. Note that in (2.13-2.14) and the remainder of this thesis, a balanced operation is assumed, i.e., $0s(r)$ variables are neglected. By inspection of (2.13-2.14) , the differential equations for the qd variables for the stator and rotor circuits can be represented by the equivalent circuits shown in Figure 2.1.

The electromagnetic torque produced by the induction machine can be expressed in qd variables [4] as

$$T_e = \frac{3}{2}\frac{P}{2}\frac{L_M}{L'_{rr}}\left(i_{qs}\lambda'_{dr} - i_{ds}\lambda'_{qr}\right), \tag{2.15}$$

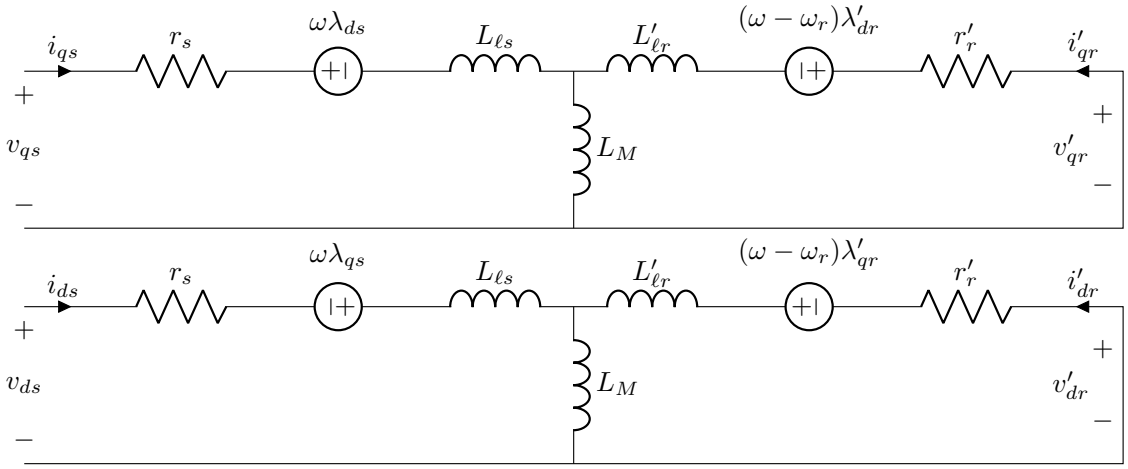


Figure 2.1: Induction machine equivalent $qd0$ circuits.

where $L'_{rr} = L_M + L'_{lr}$ is the rotor self-inductance. This leads to the mechanical motion equation

$$T_e - T_L = Jp\omega_{rm} + B_m\omega_{rm}, \tag{2.16}$$

where T_L is the mechanical load torque (defined in the opposite direction of T_e), J is the rotor's moment of inertia, and B_m is a loss coefficient representing windage and friction.

See [1] for background on the control strategies of compensated volts-per-hertz and indirect field-oriented control, which are used here for completeness of the comparison between the synchronization methodologies but not studied in detail.

2.2 External Stator Resistance Methodology

Induction machines are designed to operate under rated load at a speed slightly below the "synchronous" speed $\omega_r = \omega_e$. From (2.16), steady-state equilibrium of the motor is achieved when the difference between T_e and T_L is equal to the torque resulting from windage and friction and the derivative of T_e with respect to speed is negative [4]. To simplify explanation, this windage and friction term is often neglected, making the equilibrium point $T_e = T_L$.

As explained in [1], adding an external stator resistance to an IM decreases the slope of the equivalent T_e vs. speed curve near synchronous speed. Changing the shape of the curve in this way

moves the equilibrium point to a lower speed, as illustrated in Figure 2.2 for a 15 hp (11.19 kW) induction machine (parameters listed in Appendix A).

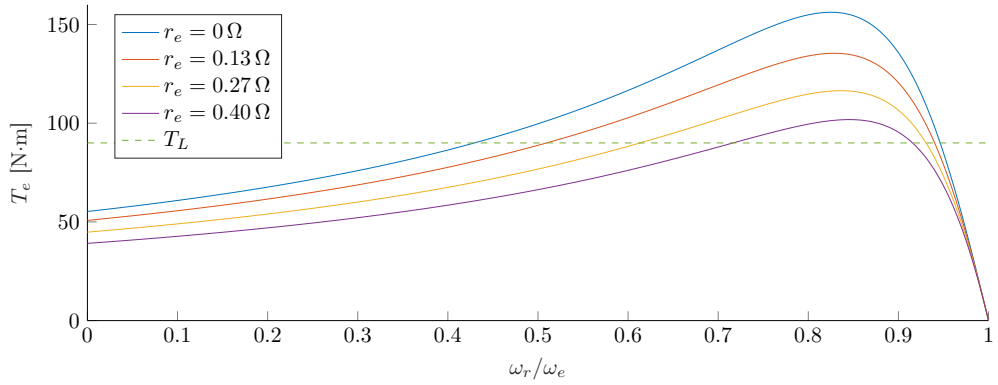


Figure 2.2: Torque vs. speed plot for selected external stator resistance values.

Note that the peak of the torque vs. speed curve moves to a higher speed as the external resistance is increased. The circuit used to implement this external stator resistance in [1] and this thesis is a power-electronic circuit with a bi-directional MOSFET arrangement shown in Figure 2.3.

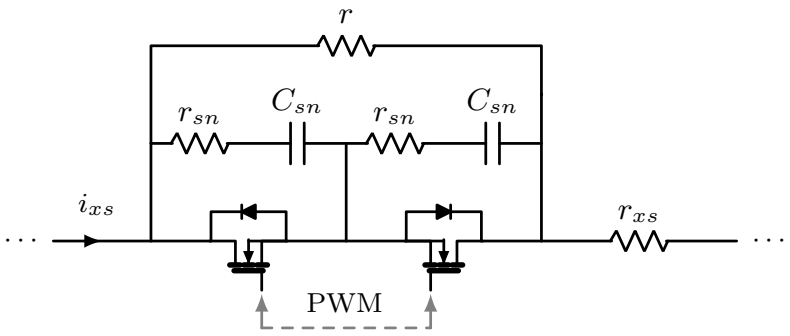


Figure 2.3: Circuit for achieving a desired average external stator resistance.

In Figure 2.3, r_s and C_s are the snubber resistance and capacitance across the back-to-back MOSFETs, i_{xs} and r_{xs} are the current and (nominal) resistance of stator phase x, and r is the 'base' external resistance. The MOSFETs receive the same PWM (fully on or fully off) signal;

the bi-directional arrangement allows current to flow in the stator in both directions. Neglecting snubber impedance, the circuit places either base resistance r or $0\,\Omega$ in series with the nominal stator resistance. By modulation of the PWM duty cycle, a desired external resistance can be achieved in a fast-average sense, where the fast-average of a signal $f(t)$ is defined as

$$\bar{f}(t) = \frac{1}{T_s}\int_0^{T_s} f(t)\,dt, \tag{2.17}$$

where the bar over $f(t)$ denotes fast-average, and $T_s = 1/f_s$ is the time period of the PWM switching cycle with frequency f_s.

The control strategy used in [1] and this thesis is a proportional-integral (PI) feedback control that updates the duty cycles of the external variable resistance circuits using the angular position differences of the IMs as inputs. In this control strategy, the motor that is experiencing the highest load torque (and therefore runs at the slowest speed without synchronization) is designated as the "primary" and used for the central converter's feedback control, while all other motors are designated as "secondary". This designation can be changed during operation to respond to varying load torques. In this thesis, values relating to the primary motor are denoted with a subscript p, while values relating to one of the secondary motors are denoted with a subscript s_i.

The difference in angular position between a secondary motor and the primary motor is denoted

$$\delta\theta_{rm,s_i}(t) = \theta_{rm,s_i}(t) - \theta_{rm,p}(t), \tag{2.18}$$

where

$$\begin{aligned}\theta_{rm,s_i}(t) &= \int_0^t \omega_{rm,s_i}(\tau)\,d\tau + \theta_{rm,s_i}(0),\\ \theta_{rm,p}(t) &= \int_0^t \omega_{rm,p}(\tau)\,d\tau + \theta_{rm,p}(0).\end{aligned} \tag{2.19}$$

The duty cycle for the external resistance for the primary motor, $D_{e,p}$, is held at the value it had when the motor was designated as primary (typically zero), while the duty cycles for the external

resistances for the secondary motors, D_{e,s_i} are updated by PI feedback as

$$D_{e,s_i} = K_{P,s_i}\delta\theta_{rm,s_i} + K_{I,s_i}\int \delta\theta_{rm,s_i}\, dt, \tag{2.20}$$

where K_{P,s_i} is the proportional gain and K_{I,s_i} is the integral gain of the PI controller for motor s_i.

2.3 Auxiliary Converter Methodology

The synchronization method introduced in this thesis instead changes the shape of the T_e vs. speed curve by adjusting the voltage applied across the IM. The effect of this on the equilibrium point is illustrated in Figure 2.4 for a 15 hp (11.19 kW) induction machine (parameters listed in Appendix A), with ΔV defined as the difference between the adjusted operating condition and the rated line-neutral RMS voltage.

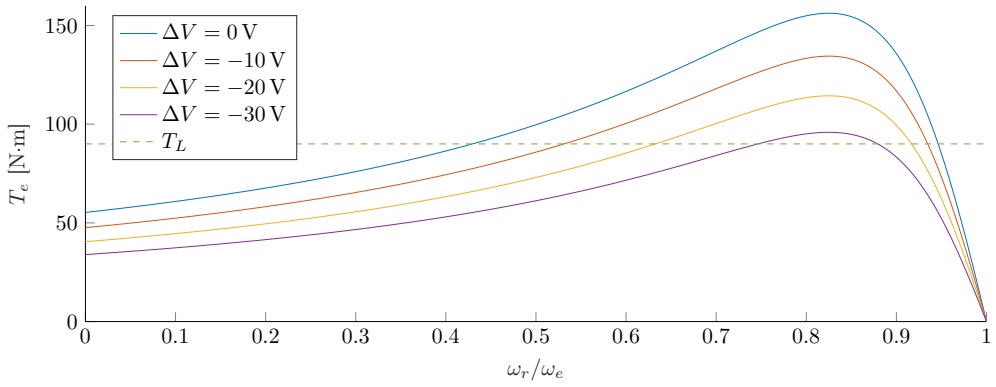


Figure 2.4: Torque vs. speed plot for selected IM input voltage adjustment values.

Note that unlike with applying an external resistance, the peak of the torque vs. speed curve remains at the same speed when adjusting the IM input voltage. The proposed implementation of this method is similar to the variable impedance implementation presented in [2], using an auxiliary converter connected to one side of a transformer, the other side being in series with one of the stator windings. However, the control strategy here is simpler, as the targeted effect is that of an ideal voltage source rather than a variable impedance, and only motor position readings are

necessary. The implementation is illustrated in Figure 2.5, which only displays one phase of the wiring for one motor for clarity.

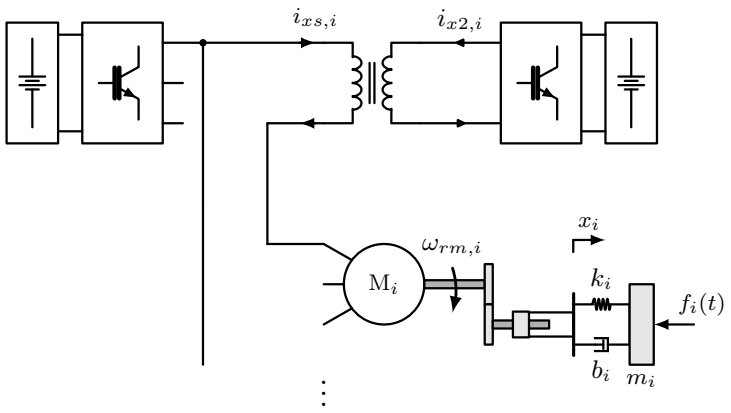


Figure 2.5: High-level per-phase, per-motor diagram of auxiliary converter implementation.

In this thesis, the side of each transformer in line with the stator is denoted the primary, while the side connected to the auxiliary converter is denoted the secondary. Note that this implementation requires independent access to both ends of the primary of each transformer. The secondary may be connected in a three-phase arrangement suitable for the auxiliary converter. (Figure 2.5 displays a neutral connection, but a delta-connected secondary should also be acceptable.)

The transformer is included for isolation between the converters and to allow for the use of a auxiliary converter running at a lower current than the IM, though this requires the converter to operate at a voltage higher than the required IM voltage adjustment. Since a power converter is used to lower the input voltage when slowing down an IM, it may be possible to recover some of the power diverted from the IM, effectively reducing the power requirement for the central converter. However, this requires the use of bidirectional auxiliary converters, which may be more expensive to build or require more complicated circuitry compared to a converter that cannot recover this power. Therefore, this thesis treats the power diverted by the auxiliary converters as "lost" for the purpose of efficiency calculations.

In a practical implementation, the auxiliary converter's maximum voltage referred across the transformer would be lower than the voltage the central converter is operating at, so the range of

torque differentials for which speed and position synchronization can be achieved is limited. An upper bound on this range is the requirement for steady-state speed synchronization. To examine the steady-state IM voltage adjustment needed for speed synchronization, we use the expression for the steady-state electromagnetic torque for a single-fed, three-phase symmetrical IM presented in [4]. Note that in the three-phase case, we use $L_M = \frac{3}{2}L_{ms}$ in place of L_{ms} and define $L_{ss} = L_{ls} + L_M$ and $L'_{rr} = L'_{lr} + L_M$. We denote the corresponding reactance X_M.

$$T_e = \frac{2(3/2)(P/2)(X_M^2/\omega_e)r'_r s|\tilde{V}_{as}|^2}{[r_s r'_r + s(X_M^2 - X_{ss}X'_{rr})]^2 + (r'_r X_{ss} + s r_s X'_{rr})^2}, \tag{2.21}$$

where the slip $s = (\omega_e - \omega_r)/\omega_e$. Suppose unequal load torques are applied to two IMs. We calculate the required voltage adjustment to retain equal slip using two methods. The first method uses one IM as a reference and reduces the voltage of the other IM to match the slip, which is required when the IMs are operated at rated voltage. The second method adjusts the voltages of both IM to reach an intermediate shared slip, which could be used if operating below rated voltage.

If the slips are equal, the torque can be treated as proportional to the square of the applied rms voltage. We denote the proportionality constant K.

$$K = \frac{2(3/2)(P/2)(X_M^2/\omega_e)r'_r s}{[r_s r'_r + s(X_M^2 - X_{ss}X'_{rr})]^2 + (r'_r X_{ss} + s r_s X'_{rr})^2} \tag{2.22}$$

Note that K is positive in motor mode ($s > 0$) and negative in generator mode ($s < 0$). For small values of s, we have

$$K \approx \frac{3(P/2)(X_M^2/\omega_e)}{r'_r(r_s^2 + X_{ss}^2)}s. \tag{2.23}$$

Therefore, when the slip is small, the torque produced is approximately proportional to s.

We neglect windage/friction for this analysis. Using the first method, IM 1 has applied line-neutral rms voltage V_{as1} and load torque T_1, while IM 2 has applied line-neutral rms voltage $V_{as1} + \Delta V$ and load torque $T_1 + \Delta T$.

$$T_1 = KV_{as1}^2, \qquad T_1 + \Delta T = K(V_{as1} + \Delta V)^2$$
$$\frac{KV_{as1}^2 + \Delta T}{K} = V_{as1}^2 + 2V_{as1}\Delta V + (\Delta V)^2$$
$$0 = -\frac{\Delta T}{K} + 2V_{as1}\Delta V + (\Delta V)^2$$
$$\Delta V = -V_{as1} \pm \sqrt{V_{as1}^2 + \frac{\Delta T}{K}}$$

This formula is only valid if $\Delta T = 0$ implies $\Delta V = 0$, so the + solution is required.

$$\Delta V = -V_{as1} + \sqrt{V_{as1}^2 + \frac{\Delta T}{K}} \tag{2.24}$$

Now we consider adjusting the voltages of both IMs. IM 1 has applied line-neutral rms voltage $V_{as} + (\Delta V/2)$ and load torque $T + (\Delta T/2)$, while IM 2 has applied line-neutral rms voltage $V_{as} - (\Delta V/2)$ and load torque $T - (\Delta T/2)$.

$$T + (\Delta T/2) = K(V_{as} + (\Delta V/2))^2, \qquad T - (\Delta T/2) = K(V_{as} - (\Delta V/2))^2$$
$$(K(V_{as} - (\Delta V/2))^2 + (\Delta T/2)) + \Delta T/2 = K(V_{as} + (\Delta V/2))^2$$
$$\Delta T = K[(V_{as} + (\Delta V/2))^2 - (V_{as} - (\Delta V/2))^2]$$
$$\Delta T = K[V_{as}^2 + V_{as}\Delta V + (\Delta V)^2/4 - (V_{as}^2 - V_{as}\Delta V + (\Delta V)^2/4)]$$
$$\Delta T = 2KV_{as}\Delta V$$

$$\Delta V = \frac{\Delta T}{2KV_{as}} \tag{2.25}$$

In both cases, the required rms voltage adjustment ΔV increases with the torque difference ΔT. Since K is approximately proportional to s, which increases with higher applied load torque, the required ΔV is lower when the base torque T is higher. Additionally, since the sign of K matches the sign of s, the required polarity of the applied voltage adjustment is reversed in generator mode. (Regardless of mode, the IM with the greater applied load torque magnitude must receive a higher voltage to synchronize.)

In this thesis, PI feedback control is used to achieve speed and position synchronization using the auxiliary converter method. This uses the same definition of primary and secondary motors and angular position used for the external resistor method's PI feedback control (2.18, 2.19). The auxiliary converters all command $v^*_{d2} = 0$ (so the commanded voltage is entirely in the q component) and, for the converters for secondary motor lines,

$$v^*_{q2,s_i} = K_{P,s_i}\delta\theta_{rm,s_i} + K_{I,s_i}\int \delta\theta_{rm,s_i}\, dt. \tag{2.26}$$

The primary motor line's auxiliary converter is disabled, with all high switches open and all low switches closed so that the converter acts like a short.

2.4 Diagram of Auxiliary Converter and Transformer

Figure 2.6 shows the full qd equivalent circuits for one motor of the auxiliary converter methodology, treating converter outputs as voltage sources.

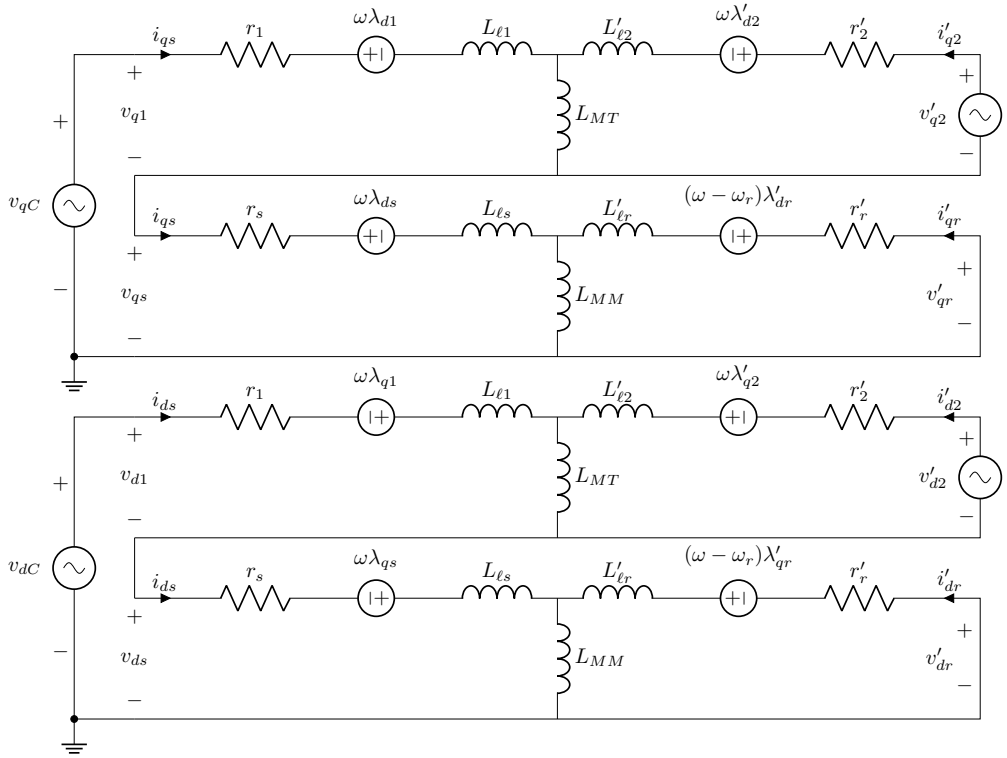


Figure 2.6: Auxiliary converter methodology equivalent $qd0$ circuits for one motor.

In Figure 2.6, v_{qC}, v_{dC} are the components of the voltage supplied by the central converter, $v_{q1}, v_{d1}, i_{qs}, i_{ds}, \lambda_{q1}, \lambda_{d1}$ are the components of the transformer primary voltage and flux linkage, $v'_{q2}, v'_{d2}, i'_{q2}, i'_{d2}, \lambda'_{q2}, \lambda'_{d2}$ are the components of the referred transformer secondary output voltage, current, and flux linkage (tied to the auxiliary converter), $v_{qs}, v_{ds}, i_{qs}, i_{ds}, \lambda_{qs}, \lambda_{ds}$ are the components of the stator winding voltage, current, and flux linkage, and $v'_{qr}, v'_{dr}, i'_{qr}, i'_{dr}, \lambda'_{qr}, \lambda'_{dr}$ are the components of the rotor winding voltage (zero for a squirrel-cage rotor), current, and flux linkage. Note that the transformer primary and the stator winding share the same current.

The transformer has primary resistance and leakage inductance $r_1, L_{\ell 1}$, referred secondary resistance and leakage inductance $r'_2, L'_{\ell 2}$, and three-phase magnetizing inductance L_{MT}. The motor has stator resistance and leakage inductance $r_s, L_{\ell s}$, referred rotor resistance and leakage inductance $r'_r, L'_{\ell r}$, and three-phase magnetizing inductance L_{MM}.

Figure 2.7 is a block diagram of the control structure of the auxiliary converter methodology.

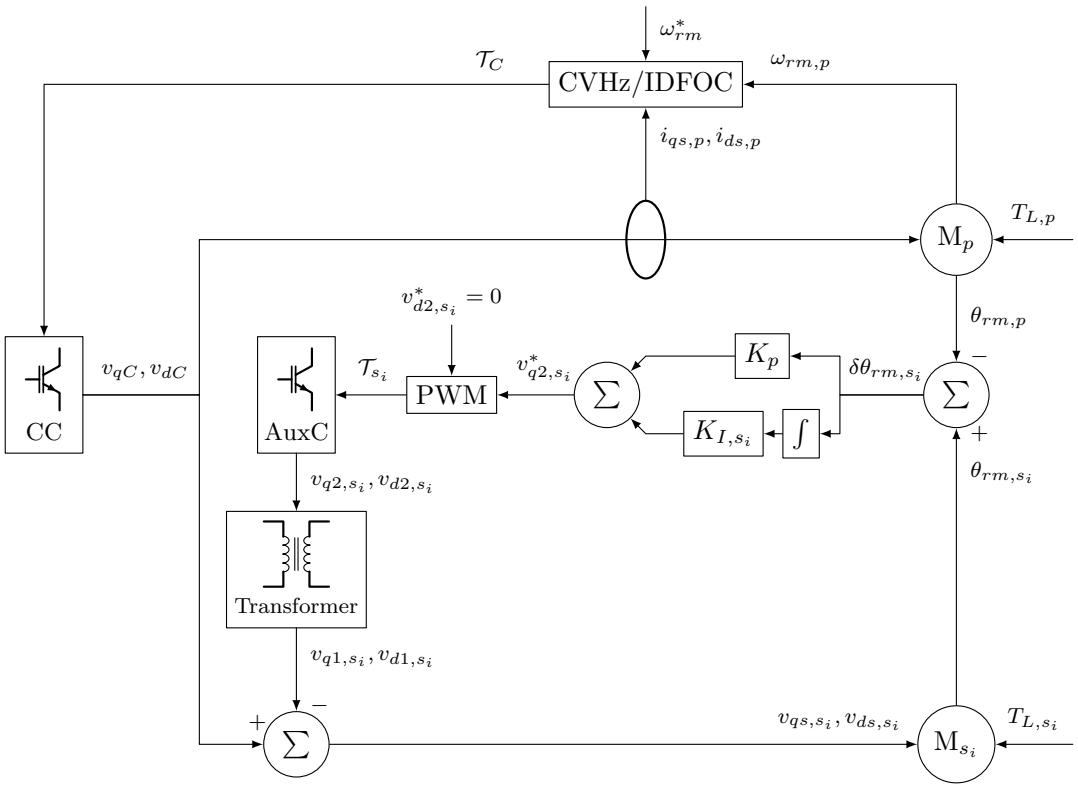


Figure 2.7: Auxiliary converter methodology block diagram, showing control for the primary motor and one secondary motor.

In Figure 2.7, the central converter is labeled as “CC” and the auxiliary converter is labeled as “AuxC”. The control strategy for the central converter and primary motor is represented as one block, as it can be changed without affecting the structure of the synchronization control. In a realistic implementation, a transformer and auxiliary converter would be included on every motor line, though the primary motor’s auxiliary converter would be disabled (acting as a short) until a new primary designation is assigned.

Chapter 3

Numerical Case Studies

This chapter describes and presents results of transient simulations used as case studies to validate the auxiliary converter methodology and compare its performance and efficiency to that of the external stator resistance methodology. Time-domain simulations were performed in MATLAB®, release R2022a. The central and auxiliary converters were modeled as six-switch, four-quadrant inverters with ideal switch states (fully on or fully off). The external resistor circuits were also modeled as switches in the same manner, neglecting snubbing. All case studies presented here include three actuation lines in the CCMM with identical 15 hp IMs used as drive motors. Motor 1 was selected to be the primary motor, with motors 2 and 3 acting as secondary motors. Parameters for the motors and control systems used in these studies are listed in Appendix A. Each case study explores the effect of varying the transformer turns ratio and auxiliary converter bus voltage.

For the auxiliary converter methodology, each motor line with transformer and auxiliary converter included was simulated using the following set of eight differential equations derived from Figure 2.6 and (2.13-2.16), where ω is an arbitrary reference frame speed:

$$p\lambda_{qC} = v_{qC} - (r_s + r_1)i_{qs} - \omega\lambda_{dC} \tag{3.1a}$$

$$p\lambda_{dC} = v_{dC} - (r_s + r_1)i_{ds} + \omega\lambda_{qC} \tag{3.1b}$$

$$p\lambda'_{qr} = v'_{qr} - r'_r i'_{qr} - (\omega - \omega_r)\lambda'_{dr} \tag{3.1c}$$

$$p\lambda'_{dr} = v'_{dr} - r'_r i'_{dr} + (\omega - \omega_r)\lambda'_{qr} \tag{3.1d}$$

$$p\lambda'_{q2} = v'_{q2} - r'_2 i'_{q2} - \omega\lambda'_{d2} \tag{3.1e}$$

$$p\lambda'_{d2} = v'_{d2} - r'_2 i'_{d2} + \omega\lambda'_{q2} \tag{3.1f}$$

$$p\omega_r = \frac{1}{J}\left(\frac{P}{2}(T_e - T_L) - B_m\omega_r\right) \tag{3.1g}$$

$$p\theta_r = \omega_r \tag{3.1h}$$

Using these differential equations requires the inductance matrix converting between current and flux linkage:

$$\begin{bmatrix} \lambda_{xC} \\ \lambda'_{xr} \\ \lambda'_{x2} \end{bmatrix} = \begin{bmatrix} L_{\ell s} + L_{\ell 1} + L_{MM} + L_{MT} & L_{MM} & L_{MT} \\ L_{MM} & L'_{\ell r} + L_{MM} & 0 \\ L_{MT} & 0 & L'_{\ell 2} + L_{MT} \end{bmatrix} \begin{bmatrix} i_{xs} \\ i'_{xr} \\ i'_{x2} \end{bmatrix}. \tag{3.2}$$

The secondary resistance and leakage inductance of the transformers ($r_2, L_{\ell 2}$) are assumed to scale linearly with the secondary-to-primary turns ratio N_2/N_1. Therefore, by the impedance referral relation $Z'_2 = (N_1/N_2)^2 Z_2$, the referred secondary resistance and leakage inductance of the transformers ($r'_2, L'_{\ell 2}$) are assumed to scale linearly with the primary-to-secondary turns ratio N_1/N_2 (i.e., the inverse of the secondary-to-primary turns ratio).

Efficiency calculations presented here consider electrical energy entering the converters, electrical energy supplied to the IMs, and mechanical energy transferred to the load. The energy required to generate the control signals is neglected, as it is relatively small and dependent on specifics of implementation.

3.1 Case Study 1a: Central Voltage Control, Ideal Switching

In this case study, all IMs were driven by the central converter using the CVHz strategy described in [1] in combination with both position synchronization schemes detailed in Chapter 2. Converter switching states for implementing the voltage control were generated using sine-triangle modulation with third harmonic injection (see [5] for implementation). All switches were modeled as ideal. In this study, the commanded speed of ω^*_{rm} = 1800 rpm was first applied at t = 0 s, with zero torque load applied to each motor. After reaching steady-state commanded rotor speed, at t = 2 s, torque loads of $T_{L,1} = 1.0T_{\text{rated}}$, $T_{L,2} = 0.8T_{\text{rated}}$, and $T_{L,3} = 0.7T_{\text{rated}}$ were then applied, where T_{rated} = 61.1 Nm was the torque rating of the (identical) IMs.

Under these conditions, the nominal voltage applied to motor 1 is $V_{as1} = \frac{240}{\sqrt{3}}$ V and the slip of motor 1 was observed to approach a steady-state value of $s \approx 0.0374$. Therefore, the proportion-

ality constant $K \approx 0.0033$ and the voltage adjustments expected for steady-state synchronization are

$$\begin{aligned} \Delta V_2 &= -V_{as1,1} + \sqrt{V_{as1,1}^2 + \frac{-0.2T_{\text{rated}}}{K}} \approx -14.3\,\text{V}, \\ \Delta V_3 &= -V_{as1,1} + \sqrt{V_{as1,1}^2 + \frac{-0.3T_{\text{rated}}}{K}} \approx -22.1\,\text{V}. \end{aligned} \tag{3.3}$$

Transient responses are shown in Figures 3.1-3.6.

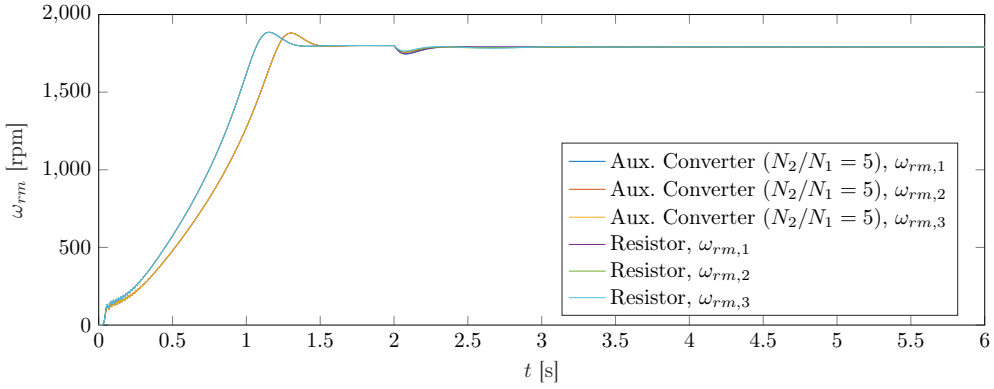


Figure 3.1: Mechanical rotor velocities including start-up, CVHz control, ideal switching.

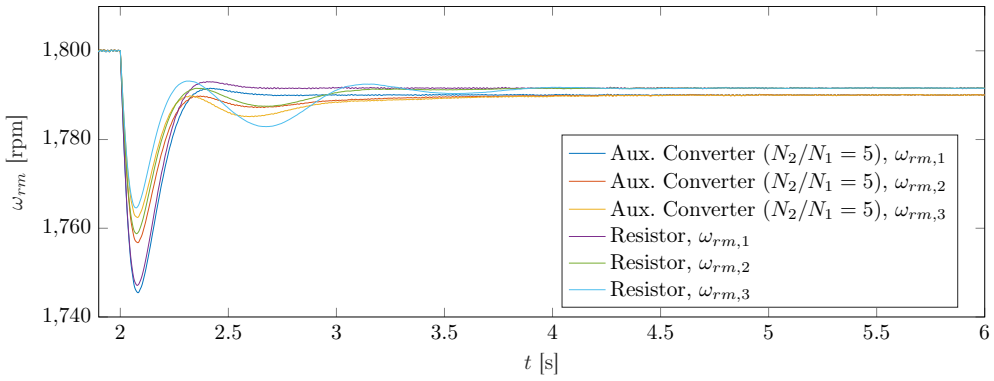


Figure 3.2: Mechanical rotor velocities (zoomed in), CVHz control, ideal switching.

Figure 3.1 shows mechanical rotor velocity for all motors including the initial start-up period, using the external resistance methodology and the auxiliary converter methodology with all converters at the same bus voltage and $N_2/N_1 = 5$. Figure 3.2 provides a zoomed-in view after the torque loads are applied. The auxiliary converter circuit slows down the acceleration a bit, tak-

ing about 0.2 s longer to stabilize with this methodology. Then, when load is applied, the speed varies slightly more with the auxiliary converter methodology compared to the external resistance methodology, settling with a slightly higher slip. However, more ringing is seen in the speed response with the external resistance methodology compared to the auxiliary converter methodology.

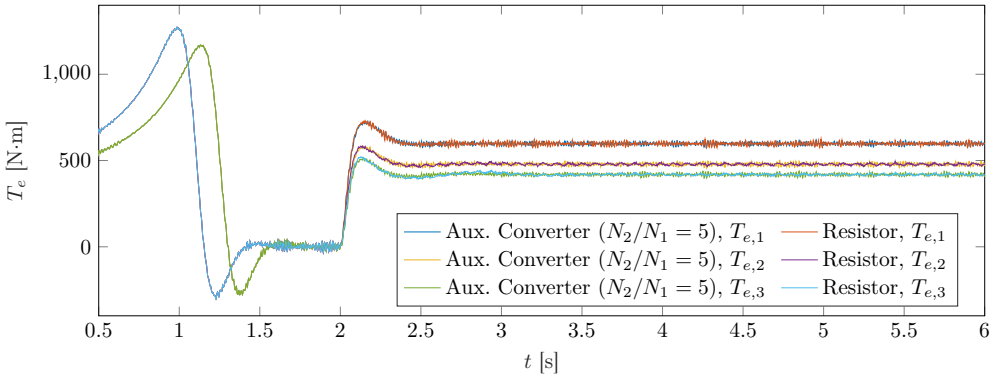
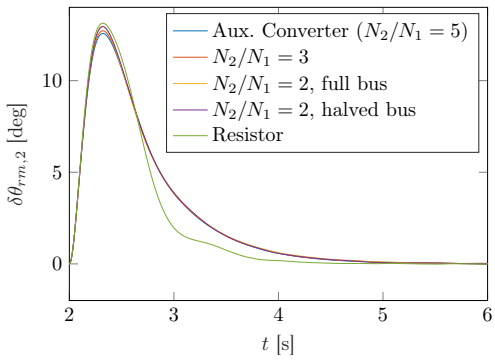
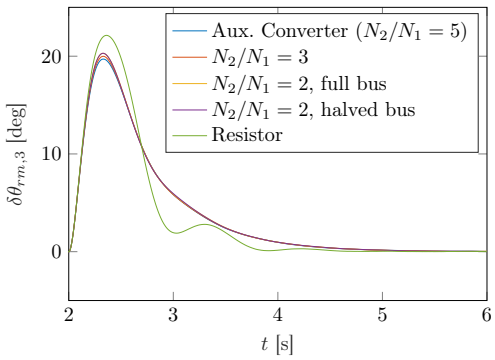


Figure 3.3: Electromagnetic torques from 0.5 s, CVHz control, ideal switching.

Figure 3.3 shows electromagnetic torques for all motors including part of the start-up period, using the external resistance methodology and the auxiliary converter methodology with all converters at the same bus voltage and $N_2/N_1 = 5$. The delay introduced by the auxiliary converter circuit is visible, but after the start-up period ends, the electromagnetic torque for each motor is nearly independent of the methodology. The torque responses for different motors separate at $t = 2$ s because of the different applied load torques.



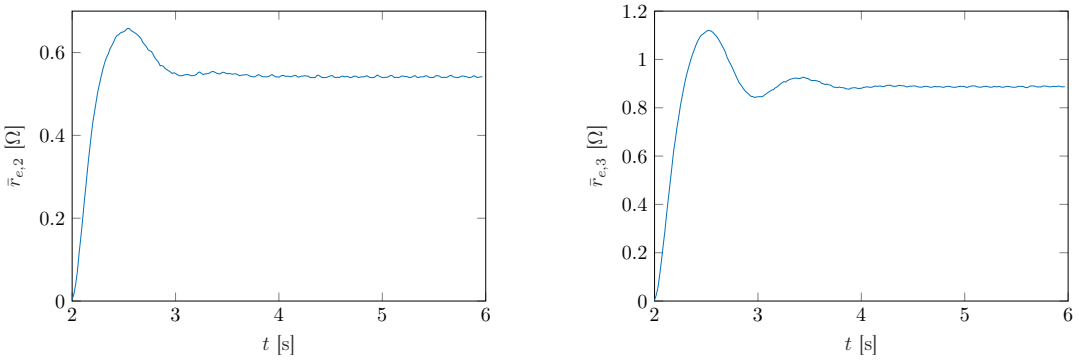
(a) Motor 2.



(b) Motor 3.

Figure 3.4: Individual angle errors, CVHz control, ideal switching.

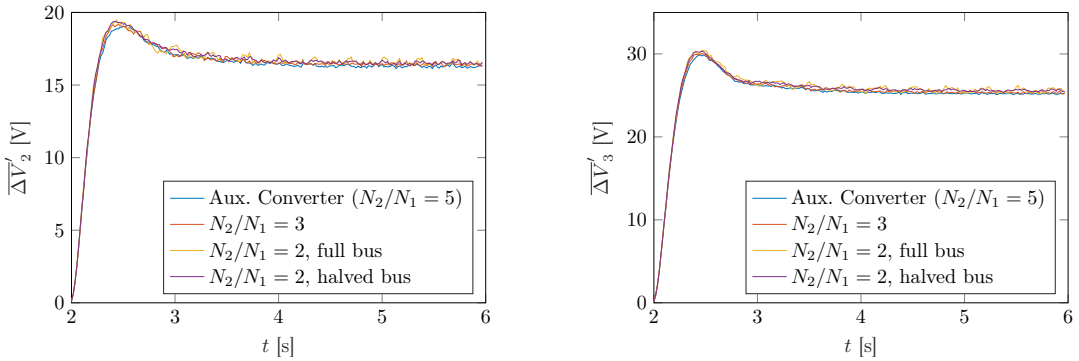
Individual rotor angle errors for the secondary motors, referenced from the primary motor, are shown in Figure 3.4. Compared to the external resistance methodology, the auxiliary converter methodology results in less overshoot and a smoother settling response, but a slightly longer settling time. The response is marginally better with a higher secondary-to-primary turns ratio for the transformer. Reducing the auxiliary converter bus voltage to half that of the central converter bus voltage has negligible impact on the response for $N_2/N_1 = 2$.



(a) Motor 2. **(b)** Motor 3.

Figure 3.5: Fast-average external resistance, CVHz control, ideal switching.

Figure 3.5 shows the fast-average external resistance (phase a) applied to each secondary motor line. Ringing is visible in the response, and a higher resistance is required when the load torque difference is higher.



(a) Motor 2. **(b)** Motor 3.

Figure 3.6: Fast-average auxiliary converter voltage, CVHz control, ideal switching.

Figure 3.6 shows the fast-average rms referred output voltage of the auxiliary converter for each secondary motor line. Less ringing occurs compared to the external resistance methodology, but a higher voltage is required when the load torque difference is higher. The output voltage is marginally lower for higher values of N_2/N_1, but the auxiliary converter bus voltage has no noticeable effect here. Note that the voltages here approach 16.5 V for motor 2 and 25.5 V for motor 3, about 15% higher than the predicted steady-state values.

Table 3.1: Efficiency analysis over first 4 seconds of operation, CVHz control, ideal switching.

Description	**Symbol**	**Ext. Resistor**	**Auxiliary Converter**			
Turns ratio	N_2/N_1	–	5	3	2	2
Auxiliary bus voltage	v_{xdc}	–	339 V	339 V	339 V	170 V
Central conv. energy in	E_{in}	137.8 kJ	134.9 kJ	134.9 kJ	134.9 kJ	135.0 kJ
Energy supp. to motors	E_{mot}	123.2 kJ	123.3 kJ	123.3 kJ	123.3 kJ	123.3 kJ
Mech. energy output	E_{T_m}	117.1 kJ	117.0 kJ	117.0 kJ	117.0 kJ	117.0 kJ
Electrical efficiency	η_e	89.40%	91.40%	91.37%	91.33%	91.31%
Mechanical efficiency	η_m	84.98%	86.78%	86.72%	86.66%	86.65%

Table 3.1 displays the energy flow measured at 3 points in the system for this case study for the first 4 seconds after load torque is applied: the electrical energy entering the central converter, the electrical energy supplied to the motors, and the mechanical energy supplied to the load by the motors. These are used to calculate values for electrical efficiency $\eta_e = E_{mot}/E_{in}$ and mechanical efficiency $\eta_m = E_{T_m}/E_{in}$. By both metrics, the auxiliary converter methodology is more energy efficient, as the central converter draws less energy while the energy output remains approximately the same. Efficiency is slightly improved by a higher secondary-to-primary turns ratio, while the auxiliary converter bus voltage appears to have no significant effect.

3.2 Case Study 1b: Central Voltage Control, Lossy Switching

In this modification of the previous case study, all switches were instead modeled with a voltage drop of 5 V in the direction of current flow, as described in [5]. The setup is otherwise identical. Transient responses are shown in Figures 3.7-3.12.

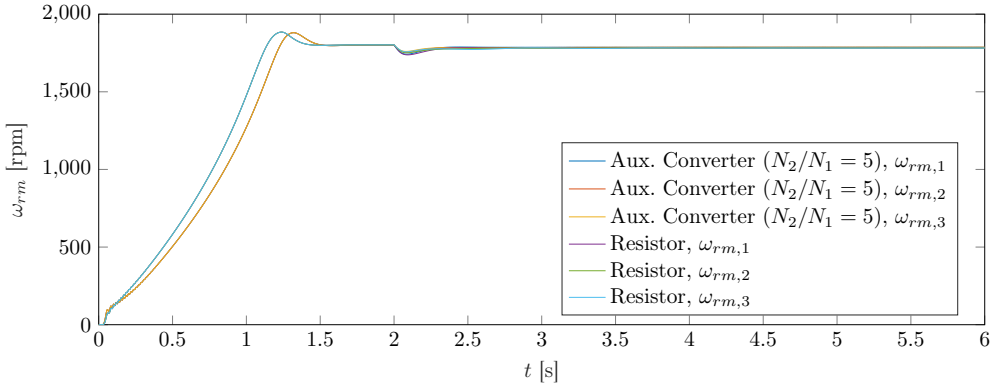


Figure 3.7: Mechanical rotor velocities including start-up, CVHz control, lossy switching.

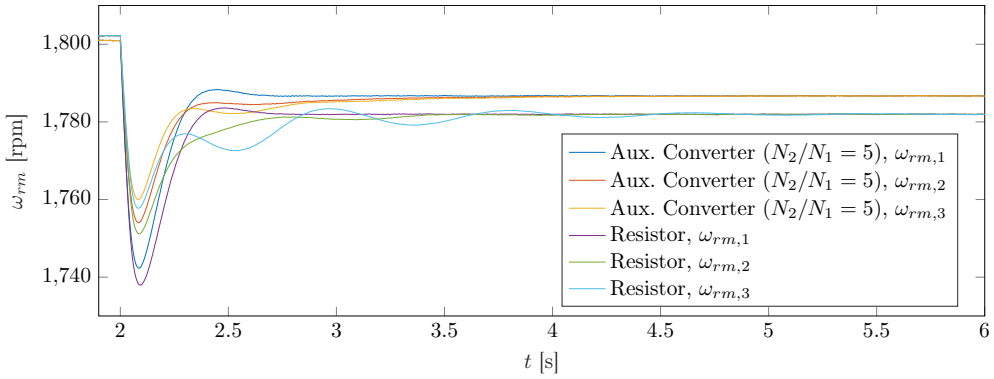


Figure 3.8: Mechanical rotor velocities (zoomed in), CVHz control, lossy switching.

Figure 3.7 shows mechanical rotor velocity for all motors including the initial start-up period, using the external resistance methodology and the auxiliary converter methodology with all converters at the same bus voltage and $N_2/N_1 = 5$, now with switching losses included. Figure 3.8 provides a zoomed-in view after the torque loads are applied. The start-up period is slightly longer in this case, and the slip is noticeably higher with a wider gap between the speeds for the different methodologies. The external resistance methodology now has more ringing in its response.

Figure 3.9 shows electromagnetic torques for all motors including part of the start-up period, using the external resistance methodology and the auxiliary converter methodology with all converters at the same bus voltage and $N_2/N_1 = 5$. The peak seen at about $t = 1$ s is lower, but the torque responses are not significantly changed beyond that point.

Individual rotor angle errors for the secondary motors, referenced from the primary motor, are shown in Figure 3.10. The introduction of switching losses increases overshoot and lengthens the

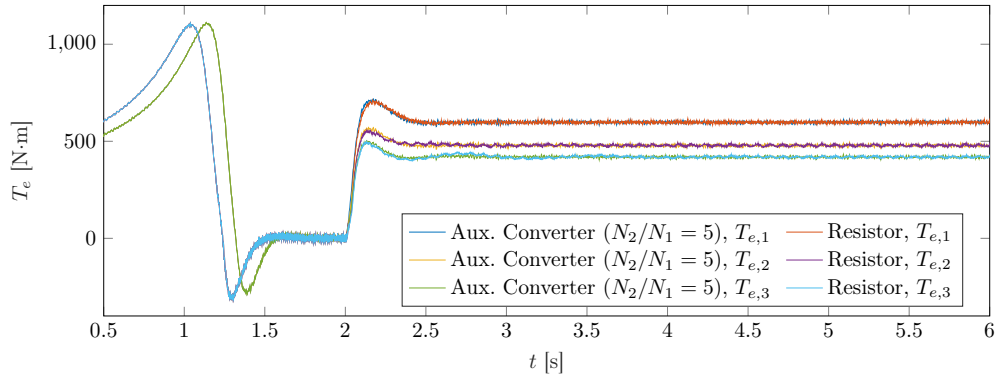
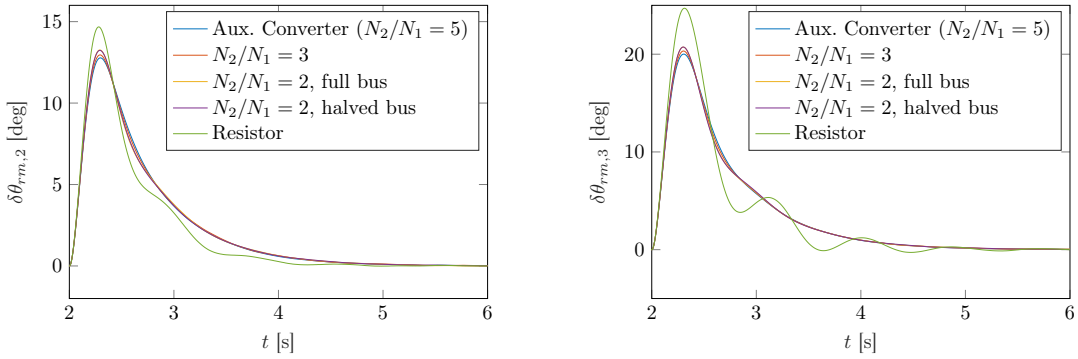


Figure 3.9: Electromagnetic torques from 0.5 s, CVHz control, lossy switching.



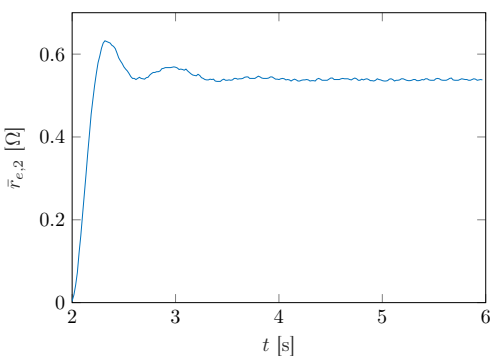
(a) Motor 2. **(b)** Motor 3.

Figure 3.10: Individual angle errors, CVHz control, lossy switching.

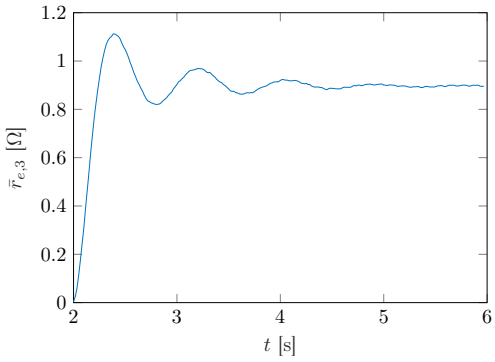
settling time, though this effect is more pronounced for the external resistance method than for the auxiliary converter method. The external resistance response also experiences more ringing in this case.

Figure 3.11 shows the fast-average external resistance (phase a) applied to each secondary motor line. The rise time is faster and ringing is more pronounced with the addition of switching losses, but peak and steady-state external resistance do not appear to have been affected significantly.

Figure 3.12 shows the fast-average rms referred output voltage of the auxiliary converter for each secondary motor line. As before, less ringing occurs compared to the external resistance methodology, but the voltage is now significantly dependent on the choice of turns ratio, with a higher secondary-to-primary turns ratio resulting in a lower voltage.

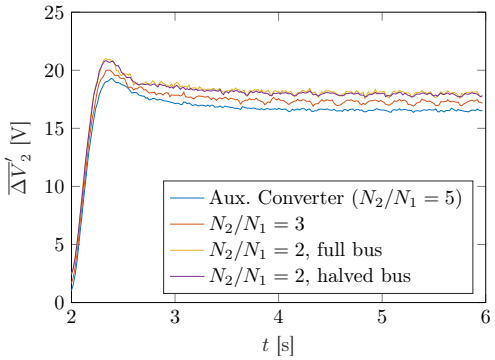


(a) Motor 2.

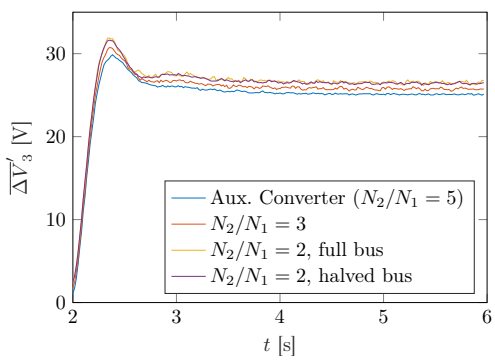


(b) Motor 3.

Figure 3.11: Fast-average external resistance, CVHz control, lossy switching.



(a) Motor 2.



(b) Motor 3.

Figure 3.12: Fast-average auxiliary converter voltage, CVHz control, lossy switching.

Table 3.2: Efficiency analysis over first 4 seconds of operation, CVHz control, lossy switching.

Description	**Symbol**	**Ext. Resistor**	**Auxiliary Converter**			
Turns ratio	N_2/N_1	–	5	3	2	2
Auxiliary bus voltage	v_{xdc}	–	339 V	339 V	339 V	170 V
Central conv. energy in	E_{in}	154.7 kJ	141.5 kJ	142.3 kJ	143.2 kJ	143.3 kJ
Energy supp. to motors	E_{mot}	123.8 kJ	123.5 kJ	123.5 kJ	123.5 kJ	123.5 kJ
Mech. energy output	E_{T_m}	116.5 kJ	116.8 kJ	116.7 kJ	116.7 kJ	116.7 kJ
Electrical efficiency	η_e	80.02%	87.29%	86.81%	86.23%	86.22%
Mechanical efficiency	η_m	75.32%	82.57%	82.06%	81.43%	81.43%

Table 3.2 displays the energy flow measured at 3 points in the system for this case study for the first 4 seconds after load torque is applied: the electrical energy entering the central converter, the electrical energy supplied to the motors, and the mechanical energy supplied to the load by the

motors. These are used to calculate values for electrical efficiency $\eta_e = E_{mot}/E_{in}$ and mechanical efficiency $\eta_m = E_{T_m}/E_{in}$. Efficiency is significantly lower with the addition of switching losses, but the gap between the methodologies has widened. The efficiency improvement from increasing the secondary-to-primary turns ratio is more significant here, but the effect of changing the auxiliary converter bus voltage is still negligible.

3.3 Case Study 2a: Central Current Control, Ideal Switching

In this case study, all IMs were driven by the central converter using the IDFOC strategy described in [1] in combination with both position synchronization schemes detailed in Chapter 2. Converter switching states for implementing the voltage control were again generated using sine-triangle modulation with third harmonic injection. All switches were modeled as ideal. The same speed commands and torque loading conditions used in the previous study were applied here.

Under these conditions, the nominal voltage applied to motor 1 is $V_{as1} = \frac{240}{\sqrt{3}}$ V and the slip of motor 1 was observed to approach a steady-state value of $s \approx 0.0333$. Therefore, the proportionality constant $K \approx 0.003$ and the voltage adjustments expected for steady-state synchronization are

$$\begin{aligned} \Delta V_2 &= -V_{as1,1} + \sqrt{V_{as1,1}^2 + \frac{-0.2T_{\text{rated}}}{K}} \approx -15.9\,\text{V}, \\ \Delta V_3 &= -V_{as1,1} + \sqrt{V_{as1,1}^2 + \frac{-0.3T_{\text{rated}}}{K}} \approx -24.7\,\text{V}. \end{aligned} \tag{3.4}$$

Transient responses are shown in Figures 3.13-3.18.

Figure 3.13 shows mechanical rotor velocity for all motors including the initial start-up period, using the external resistance methodology and the auxiliary converter methodology with all converters at the same bus voltage and $N_2/N_1 = 5$. Figure 3.14 provides a zoomed-in view after the torque loads are applied. All motors accelerate at a constant rate starting at about 0.6 s and stabilize at the same time with this methodology. Then, when load is applied, the speed varies marginally more with the auxiliary converter methodology compared to the external resistance methodology, but all motors settle back at 1800 rpm in both cases. As in the CVHz control case study, more

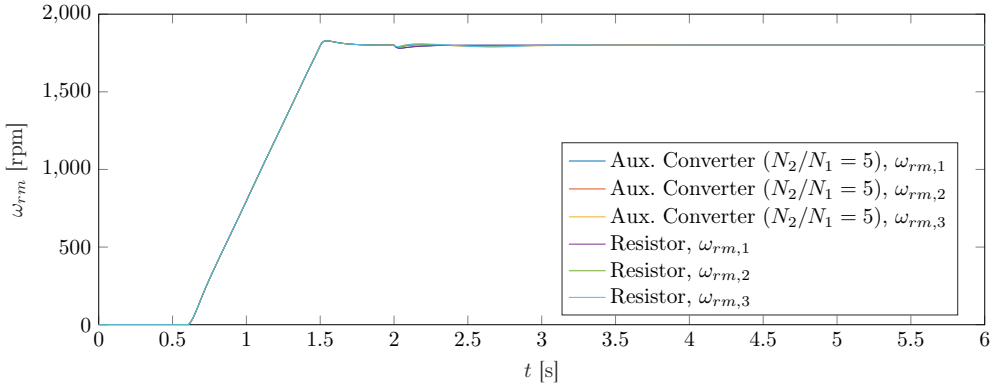


Figure 3.13: Mechanical rotor velocities including start-up, IDFOC control, ideal switching.

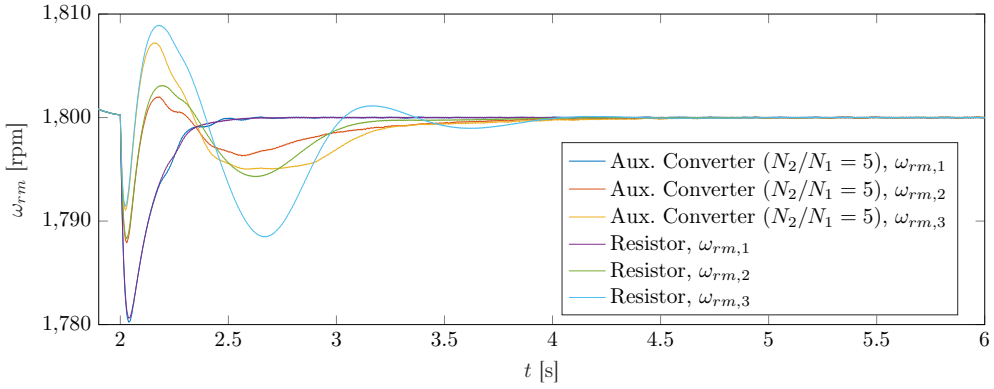


Figure 3.14: Mechanical rotor velocities (zoomed in), IDFOC control, ideal switching.

ringing is seen in the speed response with the external resistance methodology compared to the auxiliary converter methodology.

Figure 3.15 shows electromagnetic torques for all motors including part of the start-up period, using the external resistance methodology and the auxiliary converter methodology with all converters at the same bus voltage and $N_2/N_1 = 5$. The electromagnetic torque for each motor is virtually independent of the methodology until load is applied. The torque responses for different motors separate at $t = 2$ s because of the different applied load torques, with the ripple patterns looking slightly different. More torque ripple is present with IDFOC control than with CVHz control.

Individual rotor angle errors for the secondary motors, referenced from the primary motor, are shown in Figure 3.16. Compared to the external resistance methodology, the auxiliary converter

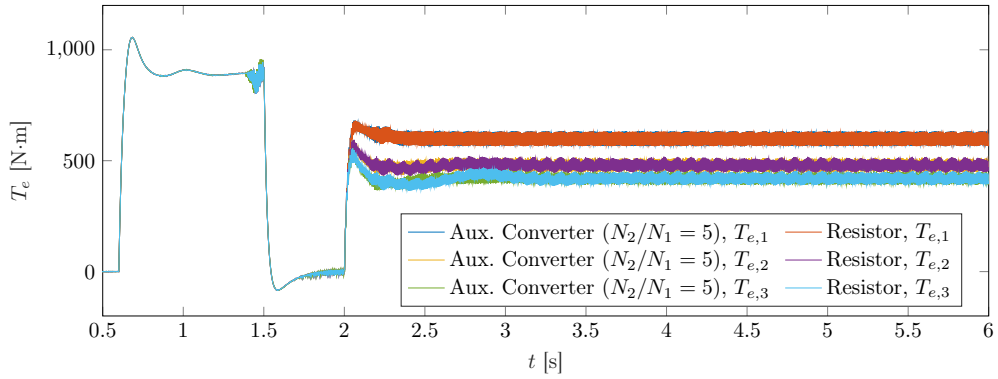
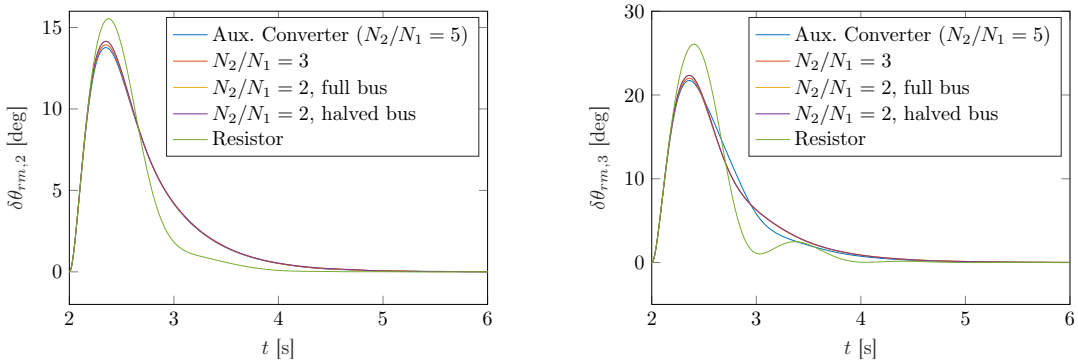


Figure 3.15: Electromagnetic torques from 0.5 s, IDFOC control, ideal switching.



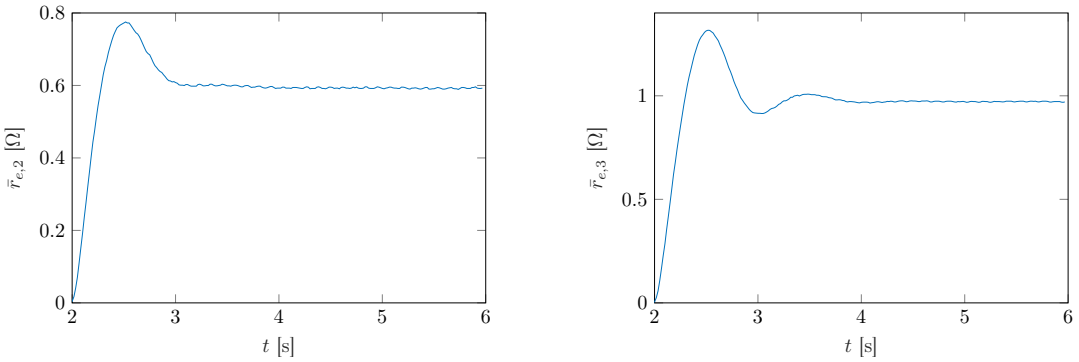
(a) Motor 2. **(b)** Motor 3.

Figure 3.16: Individual angle errors, IDFOC control, ideal switching.

methodology results in less overshoot and a smoother settling response, but a slightly longer settling time. The response is marginally better with a higher secondary-to-primary turns ratio for the transformer, though the shape is a bit different for motor 3 in the $N_2/N_1 = 5$ case (due to overmodulation). Reducing the auxiliary converter bus voltage to half that of the central converter bus voltage has negligible impact on the response for $N_2/N_1 = 2$.

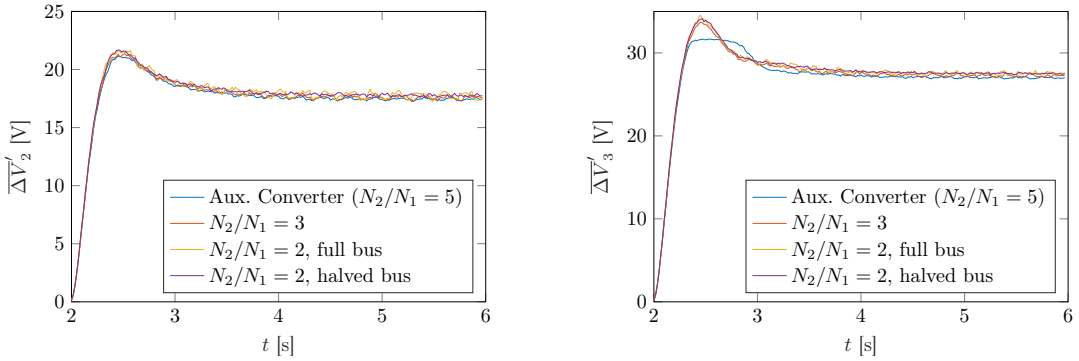
Figure 3.17 shows the fast-average external resistance (phase a) applied to each secondary motor line. Some ringing is visible in the response, and a higher resistance is required when the load torque difference is higher. The required resistance is slightly higher than in the CVHz control case study.

Figure 3.18 shows the fast-average rms referred output voltage of the auxiliary converter for each secondary motor line. Less ringing occurs compared to the external resistance methodology,



(a) Motor 2. **(b)** Motor 3.

Figure 3.17: Fast-average external resistance, IDFOC control, ideal switching.



(a) Motor 2. **(b)** Motor 3.

Figure 3.18: Fast-average auxiliary converter voltage, IDFOC control, ideal switching.

but a higher voltage is required when the load torque difference is higher, and this voltage is higher than in the CVHz case study. The output voltage is marginally lower for higher values of N_2/N_1, but the auxiliary converter bus voltage has no noticeable effect here. In the $N_2/N_1 = 5$ case, the peak commanded voltage exceeds the sine-triangle PWM maximum, leading to overmodulation: the output voltage still increases with the command, but fails to track it, leading to a shorter but wider peak. Note that the voltages here approach 17.5 V for motor 2 and 27.0 V for motor 3, about 10% higher than the predicted steady-state values.

Table 3.3 displays the energy flow measured at 3 points in the system for this case study for the first 4 seconds after load torque is applied: the electrical energy entering the central converter, the electrical energy supplied to the motors, and the mechanical energy supplied to the load by the motors. These are used to calculate values for electrical efficiency $\eta_e = E_{mot}/E_{in}$ and mechanical

Table 3.3: Efficiency analysis over first 4 seconds of operation, IDFOC control, ideal switching.

Description	**Symbol**	**Ext. Resistor**	**Auxiliary Converter**			
Turns ratio	N_2/N_1	–	5	3	2	2
Auxiliary bus voltage	v_{xdc}	–	339 V	339 V	339 V	170 V
Central conv. energy in	E_{in}	138.5 kJ	135.5 kJ	135.8 kJ	137.5 kJ	137.2 kJ
Energy supp. to motors	E_{mot}	123.8 kJ	122.5 kJ	122.8 kJ	124.3 kJ	124.0 kJ
Mech. energy output	E_{T_m}	117.7 kJ	117.7 kJ	117.7 kJ	117.7 kJ	117.7 kJ
Electrical efficiency	η_e	89.36%	90.41%	90.38%	90.41%	90.37%
Mechanical efficiency	η_m	84.99%	86.91%	86.67%	85.63%	85.81%

efficiency $\eta_m = E_{T_m}/E_{in}$. By both metrics, the auxiliary converter methodology is more energy efficient, as the central converter draws less energy while the energy output remains approximately the same. Efficiency is slightly improved by a higher secondary-to-primary turns ratio, while halving the auxiliary converter bus voltage appears to slightly reduce electrical efficiency while lightly improving mechanical efficiency.

3.4 Case Study 2b: Central Current Control, Lossy Switching

In this modification of the previous case study, all switches were instead modeled with a voltage drop of 5 V in the direction of current flow, as described in [5]. The setup is otherwise identical. Transient responses are shown in Figures 3.19-3.24.

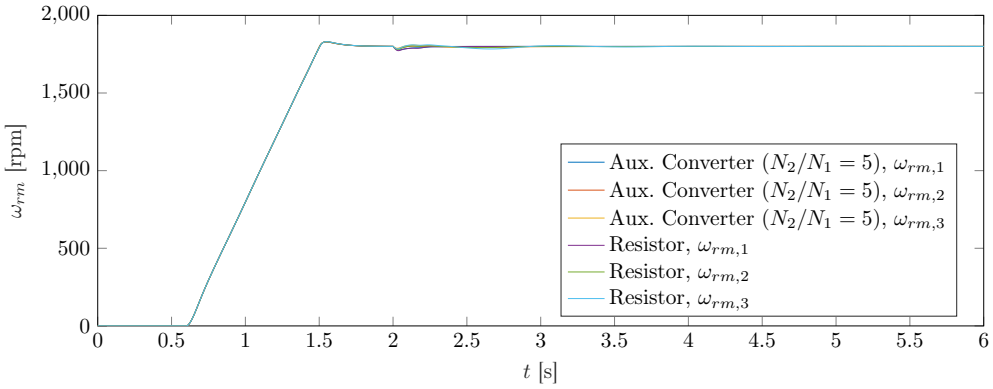


Figure 3.19: Mechanical rotor velocities including start-up, IDFOC control, lossy switching.

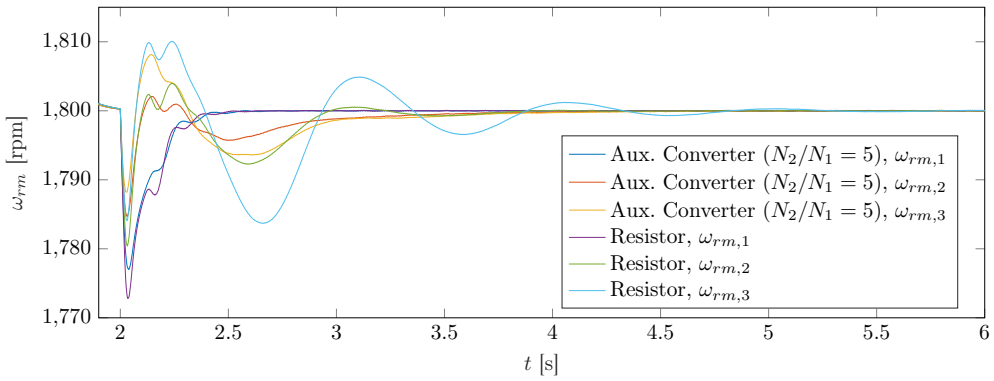


Figure 3.20: Mechanical rotor velocities (zoomed in), IDFOC control, lossy switching.

Figure 3.19 shows mechanical rotor velocity for all motors including the initial start-up period, using the external resistance methodology and the auxiliary converter methodology with all converters at the same bus voltage and $N_2/N_1 = 5$. Figure 3.20 provides a zoomed-in view after the torque loads are applied. The speed curves do not appear to be significantly affected by switching losses before load is applied. After load is applied, the speed experiences more ringing than without switching losses and the shape appears more chaotic at first, but all motors still settle back at 1800 rpm in both cases. More ringing is still seen in the speed response with the external resistance methodology compared to the auxiliary converter methodology.

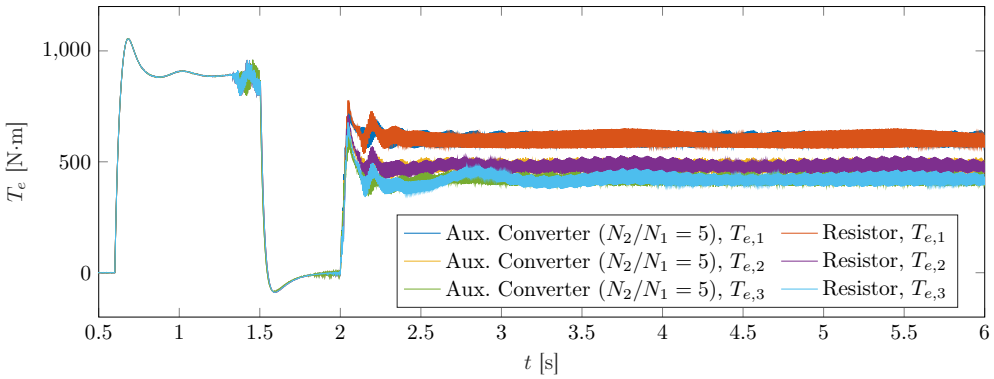
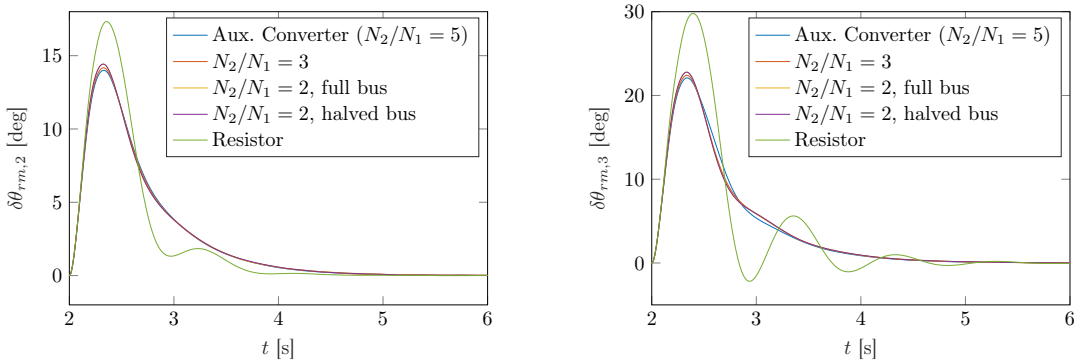


Figure 3.21: Electromagnetic torques from 0.5 s, IDFOC control, lossy switching.

Figure 3.21 shows electromagnetic torques for all motors including part of the start-up period, using the external resistance methodology and the auxiliary converter methodology with all con-

verters at the same bus voltage and $N_2/N_1 = 5$. A slight difference between the methodologies is now seen at about $t = 1.5$ s, before load is applied. The torque responses for different motors separate at $t = 2$ s because of the different applied load torques, with more ringing seen here than without switching losses.



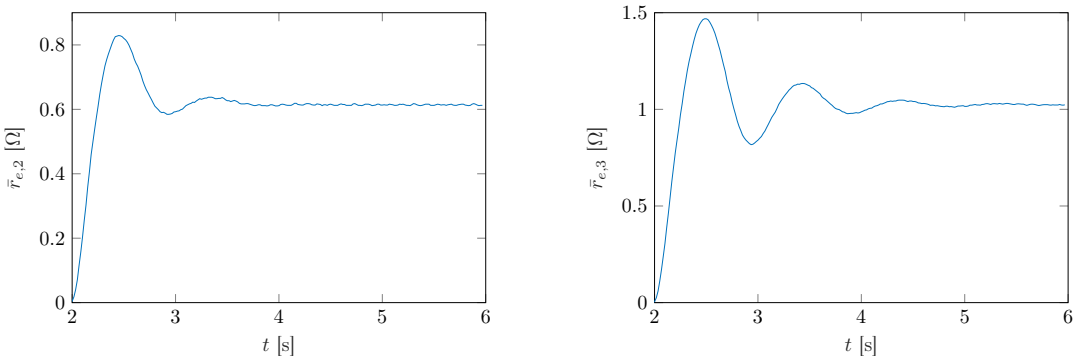
(a) Motor 2. **(b)** Motor 3.

Figure 3.22: Individual angle errors, IDFOC control, lossy switching.

Individual rotor angle errors for the secondary motors, referenced from the primary motor, are shown in Figure 3.22. Compared to the external resistance methodology, the auxiliary converter methodology results in less overshoot and a smoother settling response, but a slightly longer settling time. The response is very similar to that without switching losses, except the peak angle errors are higher and more ringing is seen in the plots for the external resistance methodology.

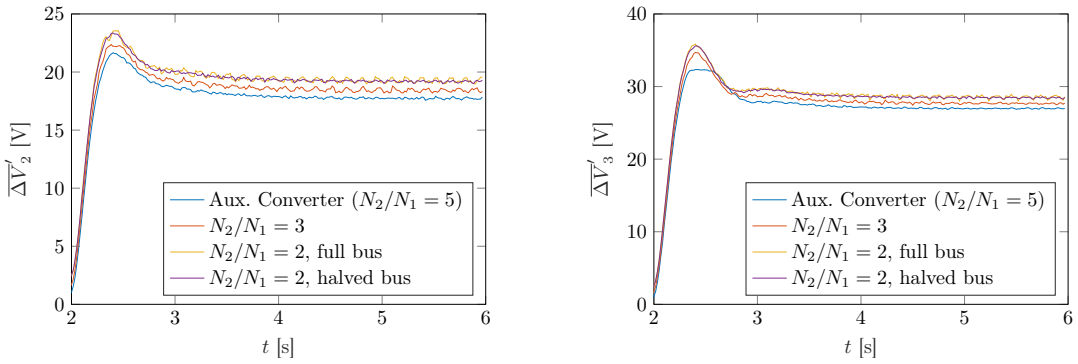
Figure 3.23 shows the fast-average external resistance (phase a) applied to each secondary motor line. More ringing is visible than without switching losses, and the peak and steady-state resistance are higher, with the resistance for motor 3 almost reaching the maximum of 1.5 Ω.

Figure 3.24 shows the fast-average rms referred output voltage of the auxiliary converter for each secondary motor line. Like with the CVHz case study, the addition of switching losses makes the voltage significantly dependent on the choice of turns ratio. As in the simulation without switching losses, overmodulation is seen for the $N_2/N_1 = 5$ case.



(a) Motor 2. **(b)** Motor 3.

Figure 3.23: Fast-average external resistance, IDFOC control, lossy switching.



(a) Motor 2. **(b)** Motor 3.

Figure 3.24: Fast-average auxiliary converter voltage, IDFOC control, lossy switching.

Table 3.4: Efficiency analysis over first 4 seconds of operation, IDFOC control, lossy switching.

Description	**Symbol**	**Ext. Resistor**	**Auxiliary Converter**			
Turns ratio	N_2/N_1	–	5	3	2	2
Auxiliary bus voltage	v_{xdc}	–	339 V	339 V	339 V	170 V
Central conv. energy in	E_{in}	153.8 kJ	142.2 kJ	143.6 kJ	145.9 kJ	145.1 kJ
Energy supp. to motors	E_{mot}	123.5 kJ	123.2 kJ	123.8 kJ	125.3 kJ	124.4 kJ
Mech. energy output	E_{T_m}	117.7 kJ	117.7 kJ	117.7 kJ	117.7 kJ	117.7 kJ
Electrical efficiency	η_e	80.34%	86.66%	86.26%	85.83%	85.73%
Mechanical efficiency	η_m	76.56%	82.78%	82.02%	80.67%	81.14%

Table 3.4 displays the energy flow measured at 3 points in the system for this case study for the first 4 seconds after load torque is applied: the electrical energy entering the central converter, the electrical energy supplied to the motors, and the mechanical energy supplied to the load by

the motors. These are used to calculate values for electrical efficiency $\eta_e = E_{mot}/E_{in}$ and mechanical efficiency $\eta_m = E_{T_m}/E_{in}$. As in the CVHz control case study, efficiency is significantly lowered by switching losses, but the gap between the methodologies has widened. The efficiency improvements from increasing the primary-to-secondary turns ratio and the mechanical efficiency improvement from lowering the auxiliary converter bus voltage are both more significant here.

3.5 Remarks

Overall, the auxiliary converter methodology was able to achieve synchronization performance similar to the external resistance methodology with less ringing and better energy efficiency. Performance could be further improved with a more optimal control algorithm. The required voltage adjustments for steady-state synchronization were consistently higher than expected, likely because of losses in the transformers, inductive loading causing a delay between the voltages applied on the motor lines, and the presence of harmonics in the converter outputs.

Use of transformers with higher secondary-to-primary turns ratios further improves efficiency, but this also limits the referred auxiliary converter output voltage and therefore the range of torque differentials for which speed and position synchronization can be achieved, so transformer selection must take this tradeoff into account. The selection of auxiliary converter bus voltage has little impact on performance when using CVHz control, but using a lower bus voltage slightly improves efficiency when using IDFOC control, which could be because harmonics are less present with a voltage command closer to the maximum. The effect of selection of these parameters on the presence of voltage and current harmonics in the system merits further investigation.

No benefit was observed from simulating in a reference frame synchronized with the electrical frequency or the motor velocity, as the switching effects still cause rapid voltage changes in all reference frames. Therefore, a stationary reference frame was chosen for simulation and arbitrary reference transformation functions were replaced with fixed stationary transform functions in the main simulation loop, except for where values in other reference frames were needed (e.g., deter-

mining commands for the central converter). This produced a small but significant reduction in simulation time, allowing for faster collection of data.

An aliasing effect was observed when the switching period for the external resistance or auxiliary converter was an integer multiple of the step time, producing a low resolution in the fast-average plots for external resistance and auxiliary converter voltage. This appears to be because changes in the duty cycle that do not cross a step time boundary have no effect on the samples collected at each step. This was resolved by changing the switching period to a slightly higher value with a much larger least common multiple with the step time.

The use of ideal voltage sources to simulate switching losses described in [5] had some peculiar effects, including steady-state operation above commanded speed seen in Figure 3.8 and the chaotic shape of the speed plots soon after load is applied in 3.20. This technique appears to be primarily intended for quantifying converter efficiency and may not be well-suited for simulating the effects of switching losses on synchronization performance, though the observed effects were not large. Simulating with a more detailed transient model of switching effects would likely be preferable for studying how the presence of switching losses affects the synchronization performance of the methods studied here.

Chapter 4

Testbed

This chapter discusses the author's contributions to a testbed for hardware emulation of various aerospace systems, including CCMM EM-TRAS implementations. The information presented here is summarized from [3]. The aerospace testbed provides a flexible environment to test different mechanical, electrical, and control architectures, is fully sensor-equipped, and includes high-speed data acquisition, allowing for anomaly detection and health prognostics investigation.

Figure 4.1 shows a functional block diagram of the testbed. It consists of three subsystems: the test stand, where the four actuation lines are mounted; the motor control cabinet, which controls the drive machines; and the supervisory control and data acquisition (SCADA) system, responsible for controlling the load machines and measuring high-speed data from the test stand.

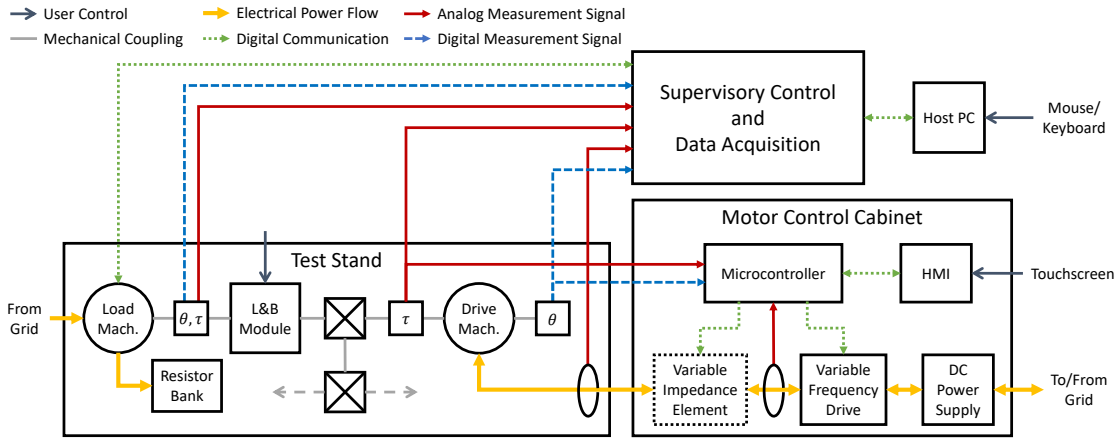


Figure 4.1: High-level functional block diagram of the aerospace testbed.

The test stand contains four actuation lines that can be mechanically coupled together by one-to-one three-way gearboxes. Each line consists of a drive machine, a load and braking module (L&B) that emulates mechanical anomalies such as jamming and sideload, a load machine responsible for emulating load torque, and several sensors, such as torque sensors (indicated by τ) and position sensors (indicated by θ). The motor control cabinet acts as the electrical drive of the

drive machines, including a custom-built variable frequency drive (VFD). It contains controller elements (HMI and microcontroller), a variable impedance element for each line responsible for position synchronization, and sensors to measure voltages and currents of each drive machine. The SCADA controls the load machines, acquires high-speed measurements from the test stand, and provides an interface for user commands. The author's contributions to the testbed include development of the SCADA system and HMI, along with construction of the testbed's electrical measurement equipment.

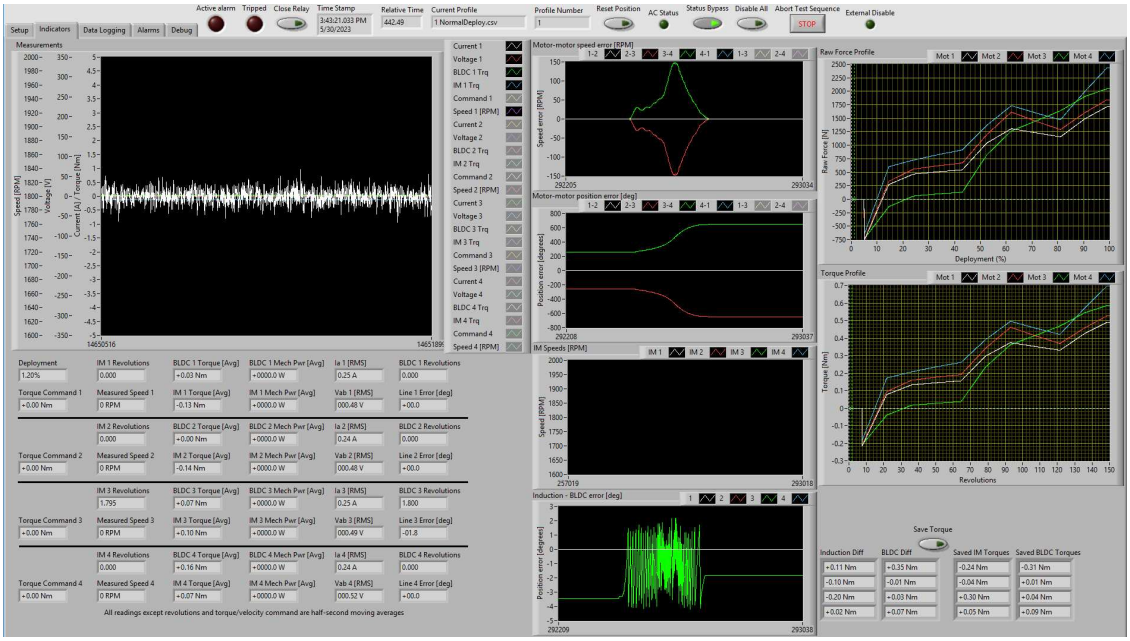


Figure 4.2: Screenshot of the "Indicators" tab of the LabVIEW SCADA user interface for the testbed.

SCADA is implemented for the testbed by a LabVIEW project consisting of an interface running on a Windows host PC, pictured in Figure 4.2, and a realtime program running on an NI PXI-1031 unit with a PXI-8016 controller. The PC interface allows for manual speed or torque control of the load motors or automated torque control based on loaded position-indexed torque profiles. Plots and data logging features for voltage, current, torque, speed, and position are also available on the PC interface, along with automated alarm and trip functionality. The PXI unit contains a PXI-6259 analog input module for voltage, current, and torque readings, a PXI-6509 digital

I/O module for load motor control, and a PXI-7833R FPGA module with digital inputs for speed and position measurement along with generation of PWM signals for the digital output module.

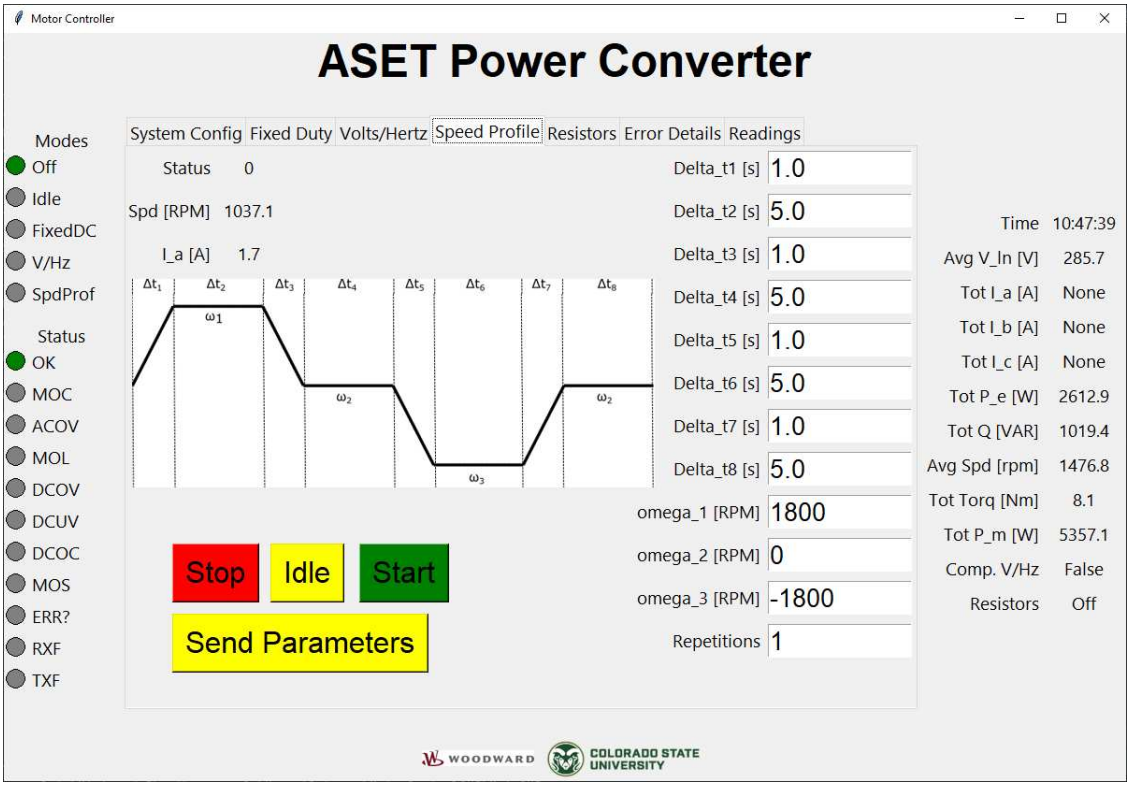


Figure 4.3: Screenshot of the "Speed Profile" tab of the HMI for the motor control cabinet.

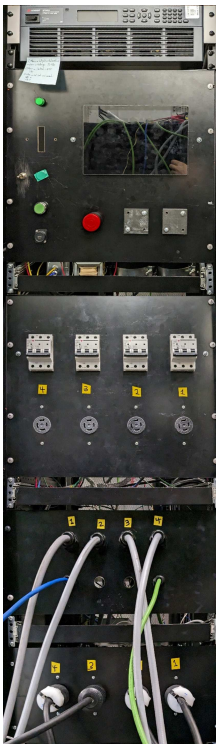
The HMI for the motor control cabinet is programmed in Python and runs on a Raspberry Pi with a touchscreen display. Using it, the user commands the microcontroller, chooses different operation modes, sets control parameters, and accesses the system status, which includes voltage, current, speed, and torque measurements along with fault conditions. Figure 4.3 shows a screenshot of the HMI. The communication protocol used is UART/SCI, an asynchronous serial protocol that allows bidirectional, simultaneous communication between the microcontroller and the HMI.

The testbed is equipped with sensors to measure mechanical variables (angular position and torque) and electrical variables (voltage and current). Each actuation line has two torque sensors measuring the torque at the shafts of the drive and load motors. The torque sensors provide an output voltage signal in the range ± 5 V corresponding to a measured torque in the range ± 20 Nm.

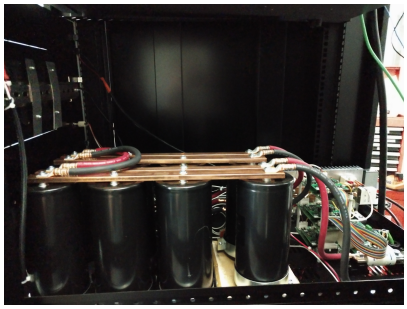
The driving-side torque signals are measured by the microcontroller, while both sides are measured by the SCADA. The torque sensors also provide an incremental encoder digital output with 60 pulses per revolution, used by the SCADA to quantify differences in speed and position between the load and driving sides of each line. Connected to the other end of the shaft of each driving machine is a 14-bit absolute encoder, which provides a high-resolution position measurement to the microcontroller and SCADA. Inside the cabinet are voltage and current sensors that measure the three-phase output voltage of the power converter and two phases of the currents of all the driving machines (the third current is calculated by the microcontroller based on no-neutral connection). Sensing boards are used to condition all of the sensors' output voltage signals to match the analog input range of the DSP (0 to 3.3 V) and to filter high-frequency components that could cause aliasing problems. Mounted on the testbed are voltage and current sensors that measure one phase of the line-to-line voltage and one phase of the current for each driving machine, feeding output voltage signals in the range ± 10 V into the SCADA. An optocoupler circuit is used to condition the encoder digital outputs from the load side torque sensors for the SCADA.

The test stand and motor control cabinet are pictured in Figure 4.4. In Figure 4.4a, the four drive motors are the induction motors on the right, the four load motors are the BLDC motors on the left, the gearboxes are visible in the middle, the 14-bit absolute encoders are on the far right, and the torque sensors are the red boxes connected to each of the eight motors. In Figure 4.4b, the DC power supply for the motor control cabinet is visible at the top. The screen below it displays the HMI, surrounded by physical controls and indicators: from the top left, a power status light, a protection bypass switch, a DC contactor status light, an enable button for the DC contactors, and an emergency stop button. The middle panel mounts breakers to control power to each motor and outlets that directly connect to the VFD output, while the lowest panel mounts outlets that connect to the VFD output through the variable impedance elements. The panel between the rows of outlets provides connections for the encoders and torque sensors.

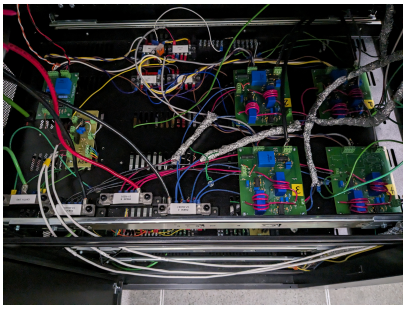
(a) Test stand.



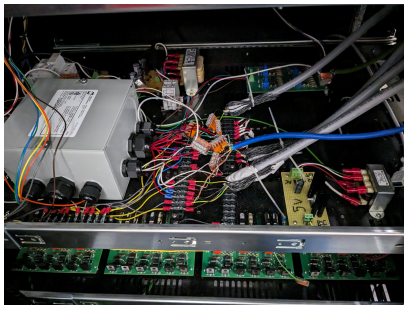
(b) Control cabinet.



(c) Upper rack.



(d) Sensor rack.



(e) Microcontroller rack.

Figure 4.4: Photographs of (a) test stand and (b,c,d,e) motor control cabinet.

The remaining subfigures display internal views of the motor control cabinet. Figure 4.4c shows the upper rack, which contains high-voltage capacitors allowing for rapid motor acceleration along with the custom VFD. Figure 4.4d shows the sensor rack, which contains voltage and current sensors for each motor and the DC bus as well as an auxiliary board conditioning additional readings for the microcontroller. Figure 4.4e shows the microcontroller rack, which contains the microcontroller rack, the board for conditioning torque signals, and some linear power supplies

used to power the encoders and sensors. Below the microcontroller rack is the bottom rack, which mounts the external resistors and their control boards.

The currently available variable impedance elements are boards that implement the external stator resistance synchronization method described in Chapter 2.2 and [1]. Hardware tests of this methodology have recently been completed, with abridged results presented in Figure 4.5.

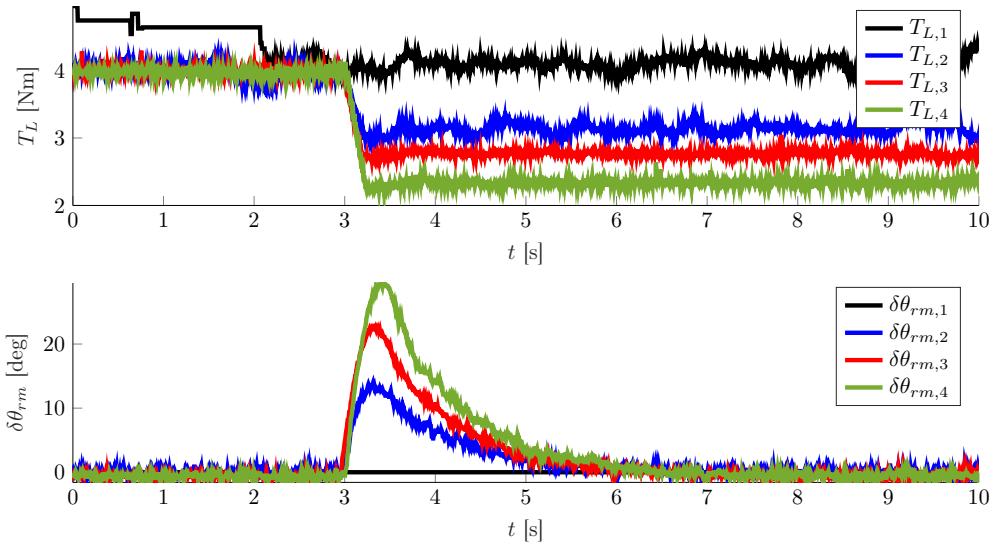


Figure 4.5: Hardware torque and position synchronization data collected from the textbed.

In the test used to generate Figure 4.5, CVHz control was used and all BLDC motors were set to apply net torque load (including windage and friction from the testbed) of 4 N·m to all IMs at first with the external resistance control keeping their positions synchronized, then the load commands were dropped by 0.8 N·m, 1.2 N·m, and 1.6 N·m for BLDC motors 2, 3, and 4 respectively. The recorded load torque reflected this change within about 0.25 s and remained fairly steady over time, while the position error rose to peaks of about 14°, 23°, and 30° for motors 2, 3, and 4 respectively before settling in about 3 s. This shows that the external resistance methodology works as expected for maintaining speed and position synchronization.

Chapter 5

Future Work

Research building on the work presented in this thesis will include the following:

- Further experimentation in simulation of the auxiliary converter methodology, including operation in generator mode, continuously varying load torques, and parameters of other motors and transformers that could realistically be used for this purpose, exploring the impact of changes to resistances and inductances.

- Investigation of the alternate auxiliary converter control strategy for operation below rated voltage, which allows an auxiliary converter to raise the voltage applied to its IM. This may achieve synchronization more quickly or more efficiently in some cases.

- Development of optimal control of both position synchronization methodologies (e.g., using H-infinity methods; see [6] for external resistance control optimization). For the auxiliary converter methodology, this could include use of a nonzero v_{d2}^* command, adding an intentional phase difference between central and auxiliary converters.

- Optimization of transformer parameters using comprehensive nonlinear models of the motors, converters, and transformers, with objectives of maximum electrical to mechanical transfer efficiency, minimum position error settling time and overshoot, and minimum cost of implementation. Obtaining a solution would require simulations performed on a computing cluster over a long period.

- Hardware validation of the auxiliary converter methodology using the testbed described in Chapter 4. This would require expansions to the testbed adding transformers and auxiliary converters in series with the drive motors, along with control capability for each auxiliary converter.

Chapter 6

Conclusions

Advances to the development of two methodologies for speed and position synchronization of induction machines in CCMM architecture used for applications such as EM-TRAS were presented in this thesis. The methodology introduced here, which uses auxiliary converters coupled through transformers to the lines supplying power to the IMs, has been shown in simulation to have better energy efficiency and some more desirable performance characteristics compared to the methodology presented in [1], which uses switched external resistance in series with the stators of the IMs. However, these improvements come at the expense of somewhat more complex requirements for equipment and control capability. The author also made significant contributions to the construction of a testbed used for hardware validation of the external resistance methodology, improving confidence that it would be feasible for application in a full-scale CCMM system. Both methodologies will likely have applications that are better suited for each due to their differing advantages and disadvantages, so investigation of both appears to be worthwhile.

It must be acknowledged that the simulations presented in this thesis constitute only preliminary analysis of the auxiliary converter methodology. One load torque profile was studied using a spread of transformer turns ratios and auxiliary converter bus voltages, but the resistance and inductance parameters of the transformers were chosen arbitrarily and conditions other than a step change from unloaded to positive load torque setpoints were not investigated. Simple PI feedback control was used with arbitrary parameter selection, and modeling and measurement uncertainties were not calculated. The author hopes that future study of both methodologies will provide a more complete picture of their feasibility and desirable use cases.

Bibliography

[1] C. d. A. Lima, J. Cale, and K. E. Shahroudi, "Rotor position synchronization in central-converter multi-motor electric actuation systems," *Energies*, vol. 14, no. 22, 2021. [Online]. Available: https://www.mdpi.com/1996-1073/14/22/7485

[2] M. Song, D. Li, X. Xiao, Y. Luo, J. Hu, and K. Yang, "Novel four-quadrant variable impedance and its application," *IET Power Electronics*, vol. 10, no. 15, pp. 2124–2132, 2017. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-pel.2017.0173

[3] C. d. A. Lima, Z. Miller, A. Riley, C. Lute, R. Bushue, B. Chan, A. Santos, S. Madcadi, J. Cale, M. Susman, and K. Shahroudi, "A novel electromechanical actuation testbed for emulation of aerospace actuation systems," in *33rd Aerospace Testing Seminar*, El Segundo, California (USA), May 2023, (accepted, to appear).

[4] P. C. Krause, O. Wasynczuk, and S. Pekarek, *Electromechanical Motion Devices*, 2nd ed. Wiley-IEEE Press, 2012.

[5] P. C. Krause, O. Wasynczuk, and S. D. Sudhoff, *Analysis of Electric Machinery and Drive Systems*, 2nd ed. Wiley-IEEE Press, 2002.

[6] C. d. A. Lima and J. Cale, "Control design for position synchronization in central converter multi-machine actuators," 2023.

Appendix A

Simulation Parameters

Table A.1: Parameters for 15 hp Induction Machine.

Description	Symbol	Value
Machine poles	P	4
Rated torque	T_{rated}	61.1 [N·m]
Stator resistance	r_s	0.06 [Ω]
Rotor resistance (referred)	r'_r	0.15 [Ω]
Stator leakage inductance	$L_{\ell s}$	1.17 [mH]
Rotor leakage inductance (referred)	$L'_{\ell r}$	1.14 [mH]
Magnetizing inductance	L_{MM}	33.4 [mH]
Windage and friction loss coefficient	B_m	5.41×10^{-4} [N·m·s]
Rotor inertia	J	0.45 [kg·m^2]

Table A.2: Parameters for Compensated Volts-per-Hertz Control.

Description	Symbol	Value
Low-pass filter time constant	τ_{LPF}	0.1 [s]
Slew-rate limiter minimum	$\alpha_{\min}$	-75.4 [rad/s^2]
Slew-rate limiter maximum	$\alpha_{\max}$	75.4 [rad/s^2]
Base rotor speed	ω_b	377 [rad/s]
Base voltage (rms)	V_b	139 [V]
Central converter DC input voltage	v_{dc}	339 [V]
PWM switching frequency	f_s	3 [kHz]
Switch voltage drop	v_{csw}	{0,5} [V]
Diode voltage drop	v_{cd}	{0,5} [V]

Table A.3: Parameters for Speed Control.

Description	Symbol	Value
Speed control proportional gain	K_{scp}	26.7 [N·m·s/rad]
Speed control integral gain	K_{sci}	8.33 [1/s]

Table A.4: Parameters for Indirect Field-oriented Control.

Description	Symbol	Value
Torque limiter upper max	$T_{e,\max}$	$2T_{rated}$
Torque limiter lower max	$T_{e,\min}$	$-2T_{rated}$
Hysteresis band tolerance	h_b	0.1 [A]
Converter DC input voltage	v_{dc}	339 [V]
Switch voltage drop	v_{csw}	{0,5} [V]
Diode voltage drop	v_{cd}	{0,5} [V]

Table A.5: Parameters for External Resistor Circuit and Control.

Description	Symbol	Value
Base resistance	r	1.5 [Ω]
PWM switching frequency	f_{rs}	4.988 [kHz]
Resistance control proportional gain	K_{rp}	30 [1/rad]
Resistance control integral gain	K_{ri}	60 [1/(rad·s)]
Switch voltage drop	v_{rsw}	{0,5} [V]
Diode voltage drop	v_{rd}	{0,5} [V]

Table A.6: Parameters for Auxiliary Converter Circuit and Control.

Description	Symbol	Value
Auxiliary converter DC input voltage	v_{xdc}	{339,170} [V]
PWM switching frequency	f_{xs}	4.988 [kHz]
Voltage control proportional gain	K_{xp}	80 [V/rad]
Voltage control integral gain	K_{xi}	120 [V/(rad·s)]
Switch voltage drop	v_{xsw}	{0,5} [V]
Diode voltage drop	v_{xd}	{0,5} [V]

Table A.7: Parameters for Auxiliary Converter Transformers.

Description	Symbol	Value
Secondary-to-primary turns ratio	N_2/N_1	{5,3,2}
Primary resistance	r_1	0.001 [Ω]
Secondary resistance (referred)	r'_2	0.01(N_1/N_2) [Ω]
Primary leakage inductance	$L_{\ell 1}$	0.1 [mH]
Secondary leakage inductance (referred)	$L'_{\ell 2}$	0.5(N_1/N_2) [mH]
Magnetizing inductance	L_{MT}	10 [mH]

Appendix B

MATLAB® Code for Simulation and Analysis

This appendix includes all code used for simulation of both synchronization methodologies and analysis of simulation data presented in this thesis. Code provided by Cláudio Lima used for simulations performed in [1] was used as a baseline.

B.1 CVHz Control, External Resistor Method

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Multirun_3IM_CVHZ_res.m
% Runs 3-motor simulation of resistor sync method using CVHz control
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear
close all
clc

% load k_hinf
% [Ak,Bk,Ck,Dk] = ssdata(k_hinf);

Kp = -30; % 1/rad
Ki = -60; % 1/(rad*s)
[Ak,Bk,Ck,Dk] = ssdata(tf([Kp Ki],[1 0]));

% Code currently only supports 3 identical motors
ParametersIM = ReturnParametersIM_15hp;

rs = ParametersIM.rs;
rr = ParametersIM.rr;
Lls = ParametersIM.Lls;
Llr = ParametersIM.Llr;
Lm = ParametersIM.Lm;
Ls = ParametersIM.Ls;
Lr = ParametersIM.Lr;
% sigma = 1 - Lm^2/(Ls*Lr);
% tau_s = Ls/rs;
% tau_r = Lr/rr;
% sigma_s = tau_s*sigma;
% sigma_r = tau_r*sigma;
% mu_s = 1/sigma_s;
% mu_r = 1/sigma_r;

% Incorporating mu_s inside the param structure
% ParametersIM.mu_s = mu_s;

ParamController.Ak = Ak;
ParamController.Bk = Bk;
ParamController.Ck = Ck;
ParamController.Dk = Dk;
ParamController.order = length(Ak);

% ParamController.rext_max = ParametersIM_2.rs*100;
ParamController.rext_max = 1.5;

% Parameters for switching losses
ParamSys.vswM = 0; % Main converter switch voltage
ParamSys.vdM = 0;  % Main converter diode voltage
ParamSys.vswR = 0; % Resistor board switch voltage
ParamSys.vdR = 0;  % Resistor board diode voltage

TL1_variation = 0; % Variation in load torque in pu
TL2_variation = -0.2;
TL3_variation = -0.3;

time_variation_TL1 = 2;
time_variation_TL2 = 2;
```

```
time_variation_TL3 = 2;

ParamSys.TL1_variation = TL1_variation;
ParamSys.TL2_variation = TL2_variation;
ParamSys.TL3_variation = TL3_variation;
ParamSys.time_variation_TL1 = time_variation_TL1;
ParamSys.time_variation_TL2 = time_variation_TL2;
ParamSys.time_variation_TL3 = time_variation_TL3;

num_states = 20 + 2*ParamController.order;
x0 = zeros(1,num_states); % Initial conditions for state variables
% load x0_save
% x0 = x0_save;

t_ini = 0;
t_end = 6;

deltat = 10e-6;
tspan = t_ini:deltat:t_end;
options = odeset('MaxStep', deltat, 'RelTol',1e-5, 'AbsTol', 1e-6);
% options = odeset();

tic
[t,y] = ode45(@(t,y) ThreeIM_1Converter_CVHz_res(t,y,ParametersIM, ...
    ParamSys, ParamController), tspan, x0, options);
toc

% Save state
% x0_save = y(end,:);
% save x0_save x0_save

%-------------------------------------------------------------------------%
%                          POST-PROCESSING
%-------------------------------------------------------------------------%

%-------------------------------------------------------------------------%
%                          PICKING UP THE STATES
%-------------------------------------------------------------------------%
%                          1ST MOTOR
%-------------------------------------------------------------------------%
lambda_ds_1   =   y(:,1);  % Flux ds                                  [Wb]
lambda_qs_1   =   y(:,2);  % Flux qs                                  [Wb]
lambda_dr_1   =   y(:,3);  % Flux dr                                  [Wb]
lambda_qr_1   =   y(:,4);  % Flux qr                                  [Wb]
wr_1          =   y(:,5);  % Rotor speed (electric)                [rad/s]
theta_r_1     =   y(:,6);  % Rotor position (electric)               [rad]
%-------------------------------------------------------------------------%
%              COMPENSATED VOLTS-PER-HERZ CONTROL
%-------------------------------------------------------------------------%
R_theta_e = y(:,7);    % Electrical angular position [rad]
% y_LPF   = y(:,8);    % Output of the low-pass filter
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
%                            2ND MOTOR
%-------------------------------------------------------------------------%
lambda_ds_2   =   y(:,9);   % Flux ds                                  [Wb]
lambda_qs_2   =   y(:,10);  % Flux qs                                  [Wb]
lambda_dr_2   =   y(:,11);  % Flux dr                                  [Wb]
lambda_qr_2   =   y(:,12);  % Flux qr                                  [Wb]
wr_2          =   y(:,13);  % Rotor speed (electric)                [rad/s]
theta_r_2      =  y(:,14);  % Rotor position (electric)               [rad]
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
%                            3RD MOTOR
%-------------------------------------------------------------------------%
lambda_ds_3   =   y(:,15);  % Flux ds                                  [Wb]
lambda_qs_3   =   y(:,16);  % Flux qs                                  [Wb]
lambda_dr_3   =   y(:,17);  % Flux dr                                  [Wb]
lambda_qr_3   =   y(:,18);  % Flux qr                                  [Wb]
wr_3          =   y(:,19);  % Rotor speed (electric)                [rad/s]
theta_r_3      =  y(:,20);  % Rotor position (electric)               [rad]
%-------------------------------------------------------------------------%

%-------------------------------------------------------------------------%
%          EXTRACTING OUTPUT STRUCTURES FROM DIFF EQ FUNCTION
%-------------------------------------------------------------------------%
[~, IM1_out, IM2_out, IM3_out, Sw_out] = cellfun(@(t,y) ...
    ThreeIM_1Converter_CVHz_res(t, y, ParametersIM, ParamSys, ...
    ParamController), num2cell(t), num2cell(y,2),'uni',0);

R_vqsC = zeros(numel(t),1);
R_vdsC = zeros(numel(t),1);
```

```
140
141 R_TL_1 = zeros(numel(t),1);
142 R_Te_1 = zeros(numel(t),1);
143 theta_a_1 = zeros(numel(t),1);
144 R_vqs_1 = zeros(numel(t),1);
145 R_vds_1 = zeros(numel(t),1);
146
147 R_TL_2 = zeros(numel(t),1);
148 R_Te_2 = zeros(numel(t),1);
149 theta_a_2 = zeros(numel(t),1);
150 R_vqs_2 = zeros(numel(t),1);
151 R_vds_2 = zeros(numel(t),1);
152 rext2_pwm = zeros(numel(t),1);
153 rext2_cont = zeros(numel(t),1);
154
155 R_TL_3 = zeros(numel(t),1);
156 R_Te_3 = zeros(numel(t),1);
157 theta_a_3 = zeros(numel(t),1);
158 R_vqs_3 = zeros(numel(t),1);
159 R_vds_3 = zeros(numel(t),1);
160 rext3_pwm = zeros(numel(t),1);
161 rext3_cont = zeros(numel(t),1);
162
163 R_S1 = zeros(numel(t),1);
164 R_S2 = zeros(numel(t),1);
165 R_S3 = zeros(numel(t),1);
166
167 for i=1:numel(t)
168     R_vqsC(i) = IM1_out{i}.vqsC;
169     R_vdsC(i) = IM1_out{i}.vdsC;
170
171     R_TL_1(i) = IM1_out{i}.TL;
172     R_Te_1(i) = IM1_out{i}.Te;
173 %    theta_a_1(i) = IM1_out{i}.theta_a;
174
175     R_vqs_1(i) = IM1_out{i}.vqs;
176     R_vds_1(i) = IM1_out{i}.vds;
177
178     R_TL_2(i) = IM2_out{i}.TL;
179     R_Te_2(i) = IM2_out{i}.Te;
180 %    theta_a_2(i) = IM2_out{i}.theta_a;
181
182     R_vqs_2(i) = IM2_out{i}.vqs;
183     R_vds_2(i) = IM2_out{i}.vds;
184
185     rext2_pwm(i) = IM2_out{i}.rext_pwm;
186     rext2_cont(i) = IM2_out{i}.rext_cont;
187
188     R_TL_3(i) = IM3_out{i}.TL;
189     R_Te_3(i) = IM3_out{i}.Te;
190 %    theta_a_3(i) = IM3_out{i}.theta_a;
191
192     R_vqs_3(i) = IM3_out{i}.vqs;
193     R_vds_3(i) = IM3_out{i}.vds;
194
195     rext3_pwm(i) = IM3_out{i}.rext_pwm;
196     rext3_cont(i) = IM3_out{i}.rext_cont;
197
198     R_S1(i) = Sw_out{i}.S(1);
199     R_S2(i) = Sw_out{i}.S(2);
200     R_S3(i) = Sw_out{i}.S(3);
201
202 end
203 %-------------------------------------------------------------------------%
204 clear y IM1_out IM2_out IM3_out Sw_out
205 %-------------------------------------------------------------------------%
206 %                    Auxiliary Equations for the IMs
207 %-------------------------------------------------------------------------%
208 % Transformation of electrical rotor speed and electrical angular position
209 % to mechanical rotor speed and mechanical angular position
210 % 1st IM
211 R_wm_1 =  wr_1/(ParametersIM.P/2);
212 R_theta_m_1 = theta_r_1/(ParametersIM.P/2);
213 % 2nd IM
214 R_wm_2 =  wr_2/(ParametersIM.P/2);
215 R_theta_m_2 = theta_r_2/(ParametersIM.P/2);
216 % 3rd IM
217 R_wm_3 =  wr_3/(ParametersIM.P/2);
218 R_theta_m_3 = theta_r_3/(ParametersIM.P/2);
219
220 % Transformation of fluxes to currents — DQ components
221 % 1ST MOTOR
```

```
[R_ids_1, R_iqs_1, R_idr_1, R_iqr_1] = Fluxes2Currents(lambda_ds_1, ...
    lambda_qs_1, lambda_dr_1, lambda_qr_1, ParametersIM);
% 2ND MOTOR
[R_ids_2, R_iqs_2, R_idr_2, R_iqr_2] = Fluxes2Currents(lambda_ds_2, ...
    lambda_qs_2, lambda_dr_2, lambda_qr_2, ParametersIM);
% 3RD MOTOR
[R_ids_3, R_iqs_3, R_idr_3, R_iqr_3] = Fluxes2Currents(lambda_ds_3, ...
    lambda_qs_3, lambda_dr_3, lambda_qr_3, ParametersIM);

% Calculating voltage actually reaching motor 2
R_vqs_2 = R_vqs_2 - (R_iqs_2 .* rext2_pwm);
R_vds_2 = R_vds_2 - (R_ids_2 .* rext2_pwm);

% Calculating voltage actually reaching motor 3
R_vqs_3 = R_vqs_3 - (R_iqs_3 .* rext3_pwm);
R_vds_3 = R_vds_3 - (R_ids_3 .* rext3_pwm);

% Clear variables not saved
clear lambda_ds_1 lambda_qs_1 lambda_dr_1 lambda_qr_1 theta_r_1 wr_1 ...
    lambda_ds_2 lambda_qs_2 lambda_dr_2 lambda_qr_2 theta_r_2 wr_2 ...
    lambda_ds_3 lambda_qs_3 lambda_dr_3 lambda_qr_3 theta_r_3 wr_3 ...
    i j tspan t_ini t_end x0 num_states deltat
%--------------------------------------------------------------------------%
```

```
function [y, IM1, IM2, IM3, Sw] = ThreeIM_1Converter_CVHz_res(t, u, ...
    ParametersIM, ParamSys, ParamController)
%--------------------------------------------------------------------------%
%                           STATES
%--------------------------------------------------------------------------%
%                           1ST MOTOR
%--------------------------------------------------------------------------%
lambda_ds_1 = u(1);  % Flux  ds                                      [Wb]
lambda_qs_1 = u(2);  % Flux  qs                                      [Wb]
lambda_dr_1 = u(3);  % Flux  dr                                      [Wb]
lambda_qr_1 = u(4);  % Flux  qr                                      [Wb]
wr_1        = u(5);  % Rotor speed (electric)                     [rad/s]
theta_r_1   = u(6);  % Rotor position (electric)                    [rad]
%--------------------------------------------------------------------------%
%              COMPENSATED VOLTS-PER-HERZ CONTROL
%--------------------------------------------------------------------------%
theta_e = u(7);    % Electrical angular position [rad]
y_LPF   = u(8);    % Output of the low-pass filter
%--------------------------------------------------------------------------%
%--------------------------------------------------------------------------%
%                             2ND MOTOR
%--------------------------------------------------------------------------%
lambda_ds_2 = u(9);   % Flux  ds                                      [Wb]
lambda_qs_2 = u(10);  % Flux  qs                                      [Wb]
lambda_dr_2 = u(11);  % Flux  dr                                      [Wb]
lambda_qr_2 = u(12);  % Flux  qr                                      [Wb]
wr_2        = u(13);  % Rotor speed (electric)                     [rad/s]
theta_r_2   = u(14);  % Rotor position (electric)                    [rad]
%--------------------------------------------------------------------------%
%--------------------------------------------------------------------------%
%                             3RD MOTOR
%--------------------------------------------------------------------------%
lambda_ds_3 = u(15);  % Flux  ds                                      [Wb]
lambda_qs_3 = u(16);  % Flux  qs                                      [Wb]
lambda_dr_3 = u(17);  % Flux  dr                                      [Wb]
lambda_qr_3 = u(18);  % Flux  qr                                      [Wb]
wr_3        = u(19);  % Rotor speed (electric)                     [rad/s]
theta_r_3   = u(20);  % Rotor position (electric)                    [rad]
%--------------------------------------------------------------------------%
%--------------------------------------------------------------------------%
%                RESISTANCE CONTROL OF MOTORS 2&3
%--------------------------------------------------------------------------%
%--------------------------------------------------------------------------%
%                         CONTROLLERS
%--------------------------------------------------------------------------%
n = ParamController.order;
% Automatic state-building of the controllers, based on order
xk2 = u(21:21+n-1);
xk2 = reshape(xk2,n,1);
xk3 = u(21+n:21+2*n-1);
xk3 = reshape(xk3,n,1);
%--------------------------------------------------------------------------%

%--------------------------------------------------------------------------%
%                           GET SYSPARAMS
%--------------------------------------------------------------------------%
vswM = ParamSys.vswM;
vdM = ParamSys.vdM;
```

```
vswR = ParamSys.vswR;
vdR = ParamSys.vdR;

TL1_delta  = ParamSys.TL1_variation;
time_delta_TL1 = ParamSys.time_variation_TL1;

TL2_delta  = ParamSys.TL2_variation;
time_delta_TL2 = ParamSys.time_variation_TL2;

TL3_delta  = ParamSys.TL3_variation;
time_delta_TL3 = ParamSys.time_variation_TL3;

if t > time_delta_TL1
    TL_1 = (1+TL1_delta)*ParametersIM.Tnom;
else
    %     TL_1 = ParametersIM.Tnom;
    TL_1 = 0;
end

if t > time_delta_TL2
    TL_2 = (1+TL2_delta)*ParametersIM.Tnom;
else
%     TL_2 = ParametersIM.Tnom;
    TL_2 = 0;
end

if t > time_delta_TL3
    TL_3 = (1+TL3_delta)*ParametersIM.Tnom;
else
%     TL_3 = ParametersIM.Tnom;
    TL_3 = 0;
end
%--------------------------------------------------------------------------%

%--------------------------------------------------------------------------%
%                          Arbitrary Reference Frame
%--------------------------------------------------------------------------%
%                           Stator reference
%--------------------------------------------------------------------------%
% % 1st IM
% wa_1 = 0;
% theta_a_1 = 0;
%
% % 2nd IM
% wa_2 = 0;
% theta_a_2 = 0;
%
% % 3rd IM
% wa_3 = 0;
% theta_a_3 = 0;
%--------------------------------------------------------------------------%

%--------------------------------------------------------------------------%
%                   Auxiliary Equations for the IMs
%--------------------------------------------------------------------------%
% Transformation of fluxes to currents — DQ components
% 1ST MOTOR
[ids_1, iqs_1, idr_1, iqr_1]  = Fluxes2Currents(lambda_ds_1, lambda_qs_1, ...
    lambda_dr_1, lambda_qr_1, ParametersIM);
% 2ND MOTOR
[ids_2, iqs_2, idr_2, iqr_2]  = Fluxes2Currents(lambda_ds_2, lambda_qs_2, ...
    lambda_dr_2, lambda_qr_2, ParametersIM);
% 3RD MOTOR
[ids_3, iqs_3, idr_3, iqr_3]  = Fluxes2Currents(lambda_ds_3, lambda_qs_3, ...
    lambda_dr_3, lambda_qr_3, ParametersIM);

% Electromagnetic and Load Torques
% 1ST MOTOR
Te_1 = 3/4*ParametersIM.P*(lambda_ds_1.*iqs_1 — lambda_qs_1.*ids_1);
% 2ND MOTOR
Te_2 = 3/4*ParametersIM.P*(lambda_ds_2.*iqs_2 — lambda_qs_2.*ids_2);
% 3RD MOTOR
Te_3 = 3/4*ParametersIM.P*(lambda_ds_3.*iqs_3 — lambda_qs_3.*ids_3);

% Transformation of electrical rotor speed and electrical angular position
% to mechanical rotor speed and mechanical angular position
% 1st IM
% wm_1 =  wr_1/(ParametersIM.P/2);
theta_m_1 = theta_r_1/(ParametersIM.P/2);
% 2nd IM
% wm_2 =  wr_2/(ParametersIM.P/2);
theta_m_2 = theta_r_2/(ParametersIM.P/2);
```

```
% 3rd IM
% wm_2 =  wr_2/(ParametersIM.P/2);
theta_m_3 = theta_r_3/(ParametersIM.P/2);

% Transforming stator and rotor currents from qd reference to abc variables
% % 1st IM
% [ias_1, ibs_1, ics_1] = qd2abcs(iqs_1, ids_1, theta_a_1);
% % 2nd IM
% [ias_2, ibs_2, ics_2] = qd2abcs(iqs_2, ids_2, theta_a_2);
% % 3rd IM
% [ias_3, ibs_3, ics_3] = qd2abcs(iqs_3, ids_3, theta_a_3);
% 1st IM
ias_1 = iqs_1;
[ibs_1, ics_1] = statbc(iqs_1, ids_1);
% 2nd IM
ias_2 = iqs_2;
[ibs_2, ics_2] = statbc(iqs_2, ids_2);
% 3rd IM
ias_3 = iqs_3;
[ibs_3, ics_3] = statbc(iqs_3, ids_3);

% Rotor Voltages (squirrel cage) -  DQ components
% 1st IM
vdr_1 = 0;
vqr_1 = 0;
% 2nd IM
vdr_2 = 0;
vqr_2 = 0;
% 3rd IM
vdr_3 = 0;
vqr_3 = 0;
%--------------------------------------------------------------------------%

%--------------------------------------------------------------------------%
%               Low-Pass Filter Definitions/Equations
%--------------------------------------------------------------------------%
% The signal x_corr = 3*P*(vqs_e_star*iqs_e - 2*rs*Is^2)/Ktv (Eq. 14.2-16
% from Krause) is filtered using a low-pass filter (LPF), accordingly to
% Krause's suggestion. In order to implement this filtering process, a new
% state, y, was created, from the diff. equation of a LPF:
% dy/dt = (x-y)/tau,
% where x is the filter input and y is the filter output, while tau is the
% filter time constant.
% The LPF output is the signal X_corr, used in the Volts-per-Hertz
% compensated control section down below.
% The input of the filter is set in the Volts-per-Hertz compensated control
% section down below, since, as seen above, it depends on vqs_e_star, which
% in turn is defined by the Volts-per-Herz compensated control equations.
%--------------------------------------------------------------------------%
% Time constant of the filter
tau_LPF = 0.1;

% Output of the LPF filter
X_corr = y_LPF;
%--------------------------------------------------------------------------%

%--------------------------------------------------------------------------%
%                  Compensated Volts-per-Herz Control
%--------------------------------------------------------------------------%

% Transforming from iabcs to iqds_e
[iqs_e_1, ids_e_1] = abcs2qd(ias_1, ibs_1, ics_1, theta_e);

% Rated voltage (line to neutral rms) and rated radian frequency
Vb = ParametersIM.Vs/sqrt(3);
wb = ParametersIM.f*2*pi;

% Commanded mechanical speed
wrm_star = DesiredSpeed(t);

% Mechanical commanded speed to electrical commanded speed
wr_star = wrm_star*ParametersIM.P/2;

% Eq 14.2-7 (Krause)
% Equation from Krause was wrong (commented version). The right one, after
% checking 2 times, with 2 different approachs, is the one used here!
% num_Ktv = 3*ParametersIM.P/2*ParametersIM.Lm^2*ParametersIM.rr*Vb^2;
% den_Ktv = ParametersIM.rr*2*(ParametersIM.rs^2+wb^2*ParametersIM.Ls^2);
num_Ktv = 3*ParametersIM.P*ParametersIM.Lm^2*Vb^2;
den_Ktv = ParametersIM.rr*2*(ParametersIM.rs^2+wb^2*ParametersIM.Ls^2);
Ktv = num_Ktv/den_Ktv;

```

```
% Eq. 14.2—12 (Krause)
Is = sqrt(iqs_e_1^2 + ids_e_1^2)/sqrt(2);

% Eq. 14.2—14 (Krause)
we = (wr_star + sqrt(max([0 wr_star^2+X_corr])))/2;

% Eq 14.2—4 (Krause)
Vs = (wr_star ~= 0)*Vb*sqrt((ParametersIM.rs^2+we^2*ParametersIM.Ls^2)/(ParametersIM.rs^2+wb^2*ParametersIM.Ls^2));

% Saturation on Vs
Vs = min(Vb,Vs);

% Placing the voltage arbitrarily in the q axis
vqs_e_star = Vs*sqrt(2);
vds_e_star = 0;

% Eq 14.2—16 (Krause)
x_corr = 3*ParametersIM.P*(vqs_e_star*iqs_e_1 — 2*ParametersIM.rs*Is^2)/Ktv;

% Input of the LPF filter
u_LPF = x_corr;

% Transforming vds_e_star and vqs_e_star to vabcs
% [vas_star, vbs_star, vcs_star] = qd2abcs(vqs_e_star,vds_e_star,theta_e);

% Converter (voltage control)
f_sw = 3e3; % inverter switching frequency
vdc = ParametersIM.Vs*sqrt(2); % vdc got by a 3—phase full—bridge rectifier
[S,~,~,~,~] = my3HarmonicSineTri_v2(t, theta_e, vqs_e_star, vds_e_star, vdc, f_sw);
IM = [(ias_1+ias_2+ias_3) (ibs_1+ibs_2+ibs_3) (ics_1+ics_2+ics_3)];
[vas, vbs, vcs] = my3PhaseConverterVoltLoss(S, vdc, vswM, vdM, IM);

% Ideal voltage source
% vas_1 = vas_star;
% vbs_1 = vbs_star;
% vcs_1 = vcs_star;

% Transforming vabcs to vds and vqs
[vqsC, vdsC] = abcs2qd(vas, vbs, vcs, theta_a_1);
%————————————————————————————————————————————————————————————————————————————%

%————————————————————————————————————————————————————————————————————————————%
%                      STATOR RESISTANCE CONTROL
%                       INDUCTION MOTORS 2 & 3
%————————————————————————————————————————————————————————————————————————————%

%————————————————————————————————————————————————————————————————————————————%
%                     CONTROLLER EQUATIONS
%————————————————————————————————————————————————————————————————————————————%
% Controller input vector uk
% For now, only the position error
Ak = ParamController.Ak;
Bk = ParamController.Bk;
Ck = ParamController.Ck;
Dk = ParamController.Dk;

uk2(1) = theta_m_1 — theta_m_2;
xk2_dot = Ak*xk2 + Bk*uk2;
yk2 = Ck*xk2 + Dk*uk2;

uk3(1) = theta_m_1 — theta_m_3;
xk3_dot = Ak*xk3 + Bk*uk3;
yk3 = Ck*xk3 + Dk*uk3;

% Controller output vector yk
% For PI controller, yk is in units of 1/rad
rext2_cont = yk2(1)*ParametersIM.rs;
rext2_cont = max(rext2_cont,0);
rext2_cont = min(rext2_cont,ParamController.rext_max);

rext3_cont = yk3(1)*ParametersIM.rs;
rext3_cont = max(rext3_cont,0);
rext3_cont = min(rext3_cont,ParamController.rext_max);
%————————————————————————————————————————————————————————————————————————————%

% Controlable external resistance via PWM
f_sw = 4.987654321e3;
r_full = ParamController.rext_max;
d2 = rext2_cont/r_full;
S_rext2 = myPWM(t, d2, f_sw);
rext2_pwm = S_rext2 * r_full;
```

```
d3 = rext3_cont/r_full;
S_rext3 = myPWM(t, d3, f_sw);
rext3_pwm = S_rext3 * r_full;

% PWM external resistance
rext2 = rext2_pwm;
rext3 = rext3_pwm;

% Ideal external resistance
% rext2 = rext2_cont;
% rext3 = rext3_cont;

% vqs_2 = vqs_1;
% vds_2 = vds_1;
% If external resistance is off, then current flows through switches. If
% these are non-ideal, then voltage drop is always 1 switch + 1 diode
vas_2 = vas - ~S_rext2*sign(ias_2)*(vswR+vdR);
vbs_2 = vbs - ~S_rext2*sign(ibs_2)*(vswR+vdR);
vcs_2 = vcs - ~S_rext2*sign(ics_2)*(vswR+vdR);
% Transforming vabcs to vds and vqs
[vqs_2, vds_2] = abcs2qd(vas_2, vbs_2, vcs_2, theta_a_2);

vas_3 = vas - ~S_rext3*sign(ias_3)*(vswR+vdR);
vbs_3 = vbs - ~S_rext3*sign(ibs_3)*(vswR+vdR);
vcs_3 = vcs - ~S_rext3*sign(ics_3)*(vswR+vdR);
% Transforming vabcs to vds and vqs
[vqs_3, vds_3] = abcs2qd(vas_3, vbs_3, vcs_3, theta_a_3);

% Apply this loss to motor line 1 too for parity. In a real system, both
% would be connected to a board in case torque differnce reverses
vas_1 = vas - sign(ias_1)*(vswR+vdR);
vbs_1 = vbs - sign(ibs_1)*(vswR+vdR);
vcs_1 = vcs - sign(ics_1)*(vswR+vdR);
% Transforming vabcs to vds and vqs
[vqs_1, vds_1] = abcs2qd(vas_1, vbs_1, vcs_1, theta_a_1);
%--------------------------------------------------------------------------%

%--------------------------------------------------------------------------%
%                       OUTPUT STRUCTURES
%--------------------------------------------------------------------------%
% Switching states
Sw.S = S;
Sw.S_rext2 = S_rext2;
Sw.S_rext3 = S_rext3;

% 1st motor
IM1.vqsC = vqsC;
IM1.vdsC = vdsC;
IM1.TL = TL_1;
IM1.Te = Te_1;
IM1.vqs = vqs_1;
IM1.vds = vds_1;
IM1.we = we;
IM1.theta_e = theta_e;
IM1.theta_a = theta_a_1;

% 2nd motor
IM2.TL = TL_2;
IM2.Te = Te_2;
IM2.vqs = vqs_2;
IM2.vds = vds_2;
IM2.theta_a = theta_a_2;
IM2.rext_pwm = rext2_pwm;
IM2.rext_cont = rext2_cont;

% 3rd motor
IM3.TL = TL_3;
IM3.Te = Te_3;
IM3.vqs = vqs_3;
IM3.vds = vds_3;
IM3.theta_a = theta_a_3;
IM3.rext_pwm = rext3_pwm;
IM3.rext_cont = rext3_cont;

%--------------------------------------------------------------------------%


%--------------------------------------------------------------------------%
%                       DIFF. STATES EQUATIONS
%--------------------------------------------------------------------------%
%                            1ST MOTOR
%--------------------------------------------------------------------------%
```

```
y(1) = vds_1 - ParametersIM.rs*ids_1 + wa_1 * lambda_qs_1;       % lambda_ds_1
y(2) = vqs_1 - ParametersIM.rs*iqs_1 - wa_1 * lambda_ds_1;       % lambda_qs_1
y(3) = vdr_1 - ParametersIM.rr*idr_1 + (wa_1-wr_1)*lambda_qr_1;  % lambda_dr_1
y(4) = vqr_1 - ParametersIM.rr*iqr_1 - (wa_1-wr_1)*lambda_dr_1;  % lambda_qr_1
y(5) = (1/ParametersIM.J)*((Te_1-TL_1)*ParametersIM.P/2 + ...
                                -ParametersIM.b*wr_1);           % wr_1
y(6) = wr_1;                                                     % theta_r_1
%-------------------------------------------------------------------------%
%              COMPENSATED VOLTS-PER-HERTZ CONTROL
%-------------------------------------------------------------------------%
y(7) = we;                      % Electrical angular position
y(8) = (u_LPF - y_LPF)/tau_LPF; % Output of the low-pass filter
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
%                           2ND MOTOR
%-------------------------------------------------------------------------%
y(9) = vds_2 - (ParametersIM.rs + rext2)*ids_2 + wa_2 * lambda_qs_2;      % lambda_ds_2
y(10) = vqs_2 - (ParametersIM.rs + rext2)*iqs_2 - wa_2 * lambda_ds_2;     % lambda_qs_2
y(11) = vdr_2 - ParametersIM.rr*idr_2 + (wa_2-wr_2)*lambda_qr_2; % lambda_dr_2
y(12) = vqr_2 - ParametersIM.rr*iqr_2 - (wa_2-wr_2)*lambda_dr_2; % lambda_qr_2
y(13) = (1/ParametersIM.J)*((Te_2-TL_2)*ParametersIM.P/2 + ...
                                -ParametersIM.b*wr_2);            % wr_2
y(14) = wr_2;                                                      % theta_r_2
%-------------------------------------------------------------------------%
%                           3RD MOTOR
%-------------------------------------------------------------------------%
y(15) = vds_3 - (ParametersIM.rs + rext3)*ids_3 + wa_3 * lambda_qs_3;     % lambda_ds_3
y(16) = vqs_3 - (ParametersIM.rs + rext3)*iqs_3 - wa_3 * lambda_ds_3;     % lambda_qs_3
y(17) = vdr_3 - ParametersIM.rr*idr_3 + (wa_3-wr_3)*lambda_qr_3; % lambda_dr_3
y(18) = vqr_3 - ParametersIM.rr*iqr_3 - (wa_3-wr_3)*lambda_dr_3; % lambda_qr_3
y(19) = (1/ParametersIM.J)*((Te_3-TL_3)*ParametersIM.P/2 + ...
                                -ParametersIM.b*wr_3);            % wr_3
y(20) = wr_3;                                                      % theta_r_3
%-------------------------------------------------------------------------%
%                RESISTANCE CONTROL OF 2ND & 3RD MOTOR
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
%                        CONTROLLER
%-------------------------------------------------------------------------%
% Automatic state-building diff. equations of the controller, based on its order
y(21:21+n-1) = xk2_dot;
y(21+n:21+2*n-1) = xk3_dot;
%-------------------------------------------------------------------------%

y = y';
end
```

B.2 CVHz Control, Auxiliary Converter Method

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Multirun_3IM_CVHZ_volt_xf.m
% Runs 3-motor simulation of auxiliary converter method using CVHz control
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear
close all
clc

% load k_hinf
% [Ak,Bk,Ck,Dk] = ssdata(k_hinf);

Kp = -80; % V/rad
Ki = -120; % V/(rad*s)
[Ak,Bk,Ck,Dk] = ssdata(tf([Kp Ki],[1 0]));

% Code currently only supports 3 identical motors
ParametersIM = ReturnParametersIM_15hp;

rs = ParametersIM.rs;
rr = ParametersIM.rr;
Lls = ParametersIM.Lls;
Llr = ParametersIM.Llr;
Lm = ParametersIM.Lm;
Ls = ParametersIM.Ls;
Lr = ParametersIM.Lr;
% sigma = 1 - Lm^2/(Ls*Lr);
% tau_s = Ls/rs;
% tau_r = Lr/rr;
% sigma_s = tau_s*sigma;
% sigma_r = tau_r*sigma;
```

```
% mu_s = 1/sigma_s;
% mu_r = 1/sigma_r;

% Incorporating mu_s inside the param structure
% ParametersIM.mu_s = mu_s;

ParamController.Ak = Ak;
ParamController.Bk = Bk;
ParamController.Ck = Ck;
ParamController.Dk = Dk;
ParamController.order = length(Ak);

ParamController.delv_max = 50;
ParamController.tstart = 2; % Time to turn on voltage drop converters

% Parameters for switching losses
ParamSys.vswM = 0; % Main converter switch voltage
ParamSys.vdM = 0;  % Main converter diode voltage
ParamSys.vswD = 0; % Drop converter switch voltage
ParamSys.vdD = 0;  % Drop converter diode voltage

time_variation_TL1 = 2;
time_variation_TL2 = 2;
time_variation_TL3 = 2;

ParamSys.time_variation_TL1 = time_variation_TL1;
ParamSys.time_variation_TL2 = time_variation_TL2;
ParamSys.time_variation_TL3 = time_variation_TL3;

TL1_variation = 0; % Variation in load torque in pu
ParamSys.TL1_variation = TL1_variation;

TL2_variation = -0.2;
ParamSys.TL2_variation = TL2_variation;

TL3_variation = -0.3;
ParamSys.TL3_variation = TL3_variation;

num_states = 26 + 2*ParamController.order;
x0 = zeros(1,num_states); % Initial conditions for state variables
% load x0_save
% x0 = x0_save;

t_ini = 0;
t_end = 6;

deltat = 10e-6;
tspan = t_ini:deltat:t_end;
options = odeset('MaxStep', deltat, 'RelTol',1e-5, 'AbsTol', 1e-6);
% options = odeset();

% Set up iterations and allocate space for saved variables
Nlist = [1/5 1/3 1/2 1/2];
vdcRlist = [1 1 1 0.5];

TL_2_out = zeros(numel(tspan),numel(Nlist));
Te_2_out = TL_2_out;
delv2_vd_out = TL_2_out;
delv2_vq_out = TL_2_out;
delv2_cont_out = TL_2_out;
wm_2_out = TL_2_out;
theta_m_2_out = TL_2_out;
ids_2_out = TL_2_out;
iqs_2_out = TL_2_out;
idr_2_out = TL_2_out;
iqr_2_out = TL_2_out;
id2_2_out = TL_2_out;
iq2_2_out = TL_2_out;
vq1_2_out = TL_2_out;
vd1_2_out = TL_2_out;
vqs_2_out = TL_2_out;
vds_2_out = TL_2_out;
Sd1_2_out = TL_2_out;
Sd2_2_out = TL_2_out;
Sd3_2_out = TL_2_out;

TL_3_out = TL_2_out;
Te_3_out = TL_2_out;
delv3_vd_out = TL_2_out;
delv3_vq_out = TL_2_out;
delv3_cont_out = TL_2_out;
wm_3_out = TL_2_out;
```

```
theta_m_3_out = TL_2_out;
ids_3_out = TL_2_out;
iqs_3_out = TL_2_out;
idr_3_out = TL_2_out;
iqr_3_out = TL_2_out;
id2_3_out = TL_2_out;
iq2_3_out = TL_2_out;
vq1_3_out = TL_2_out;
vd1_3_out = TL_2_out;
vqs_3_out = TL_2_out;
vds_3_out = TL_2_out;
Sd1_3_out = TL_2_out;
Sd2_3_out = TL_2_out;
Sd3_3_out = TL_2_out;

TL_1_out = TL_2_out;
Te_1_out = TL_1_out;
wm_1_out = TL_1_out;
theta_m_1_out = TL_1_out;
ids_1_out = TL_1_out;
iqs_1_out = TL_1_out;
idr_1_out = TL_1_out;
iqr_1_out = TL_1_out;
id2_1_out = TL_1_out;
iq2_1_out = TL_1_out;
vq1_1_out = TL_1_out;
vd1_1_out = TL_1_out;
vqs_1_out = TL_1_out;
vds_1_out = TL_1_out;
S1_out = TL_1_out;
S2_out = TL_1_out;
S3_out = TL_1_out;

theta_e_out = TL_1_out;

% Iterate over transformer turns ratios & auxiliary converter bus voltages
for j=1:numel(Nlist)

% Transformer params: line side = primary (1), aux converter = secondary (2)
N = Nlist(j);              % N1/N2
ParametersXF.N = N;        % Z' = Z*N^2, but less windings = less impedance
ParametersXF.r1 = 0.001;  % So actual secondary impedance scales with N2,
ParametersXF.r2 = 0.01*N; % canceling a power of N for this scaling
ParametersXF.Ll1 = 0.1e-3;
ParametersXF.Ll2 = 0.5e-3*N;
ParametersXF.LmT = 10e-3;
ParametersXF.vdcR = vdcRlist(j); % Ratio between aux converter bus and main bus

% Calculate inductance matrix inverses in setup
% The factor of 3/2 for magnetizing inductance (Lm -> M) is assumed included
Ll1 = ParametersXF.Ll1;
Ll2 = ParametersXF.Ll2;
LmT = ParametersXF.LmT;
L = [[Lls+Ll1+Lm+LmT, Lm, LmT];...
    [Lm, Llr+Lm, 0];...
    [LmT, 0, Ll2+LmT]];
ParametersXF.Linv = inv(L);

tic
[t,y] = ode45(@(t,y) ThreeIM_1Converter_CVHz_volt_xf(t,y,ParametersIM, ...
    ParametersXF, ParamSys, ParamController), tspan, x0, options);
toc

% Save state
% x0_save = y(end,:);
% save x0_save x0_save

%-------------------------------------------------------------------------%
%                           POST-PROCESSING
%-------------------------------------------------------------------------%

%-------------------------------------------------------------------------%
%                           PICKING UP THE STATES
%-------------------------------------------------------------------------%
%                           1ST MOTOR
%-------------------------------------------------------------------------%
lambda_dC_1   =   y(:,1);  % Flux dC (main converter path)            [Wb]
lambda_qC_1   =   y(:,2);  % Flux qC                                  [Wb]
lambda_dr_1   =   y(:,3);  % Flux dr (rotor path)                     [Wb]
lambda_qr_1   =   y(:,4);  % Flux qr                                  [Wb]
lambda_d2_1   =   y(:,5);  % Flux d2 (side converter path)            [Wb]
lambda_q2_1   =   y(:,6);  % Flux q2                                  [Wb]
```

```
wr_1          =   y(:,7);  % Rotor speed (electric)                [rad/s]
theta_r_1     =   y(:,8);  % Rotor position (electric)                [rad]
%-------------------------------------------------------------------------%
%               COMPENSATED VOLTS-PER-HERZ CONTROL
%-------------------------------------------------------------------------%
theta_e = y(:,9);    % Electrical angular position [rad]
% y_LPF   = y(:,10);   % Output of the low-pass filter
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
%                             2ND MOTOR
%-------------------------------------------------------------------------%
lambda_dC_2   =   y(:,11);  % Flux dC (main converter path)           [Wb]
lambda_qC_2   =   y(:,12);  % Flux qC                                 [Wb]
lambda_dr_2   =   y(:,13);  % Flux dr (rotor path)                    [Wb]
lambda_qr_2   =   y(:,14);  % Flux qr                                 [Wb]
lambda_d2_2   =   y(:,15);  % Flux d2 (side converter path)           [Wb]
lambda_q2_2   =   y(:,16);  % Flux q2                                 [Wb]
wr_2          =   y(:,17);  % Rotor speed (electric)                [rad/s]
theta_r_2     =   y(:,18);  % Rotor position (electric)                [rad]
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
%                             3RD MOTOR
%-------------------------------------------------------------------------%
lambda_dC_3   =   y(:,19);  % Flux dC (main converter path)           [Wb]
lambda_qC_3   =   y(:,20);  % Flux qC                                 [Wb]
lambda_dr_3   =   y(:,21);  % Flux dr (rotor path)                    [Wb]
lambda_qr_3   =   y(:,22);  % Flux qr                                 [Wb]
lambda_d2_3   =   y(:,23);  % Flux d2 (side converter path)           [Wb]
lambda_q2_3   =   y(:,24);  % Flux q2                                 [Wb]
wr_3          =   y(:,25);  % Rotor speed (electric)                [rad/s]
theta_r_3     =   y(:,26);  % Rotor position (electric)                [rad]
%-------------------------------------------------------------------------%

%-------------------------------------------------------------------------%
%          EXTRACTING OUTPUT STRUCTURES FROM DIFF EQ FUNCTION
%-------------------------------------------------------------------------%
[~, IM1_out, IM2_out, IM3_out, Sw_out] = cellfun(@(t,y) ...
    ThreeIM_1Converter_CVHz_volt_xf(t, y, ParametersIM, ParametersXF, ...
    ParamSys, ParamController), num2cell(t), num2cell(y,2),'uni',0);

TL_1 = zeros(numel(t),1);
Te_1 = zeros(numel(t),1);
% wa_1 = zeros(numel(t),1);
% theta_a_1 = zeros(numel(t),1);
vqC = zeros(numel(t),1);
vdC = zeros(numel(t),1);

TL_2 = zeros(numel(t),1);
Te_2 = zeros(numel(t),1);
% wa_2 = zeros(numel(t),1);
% theta_a_2 = zeros(numel(t),1);

delv2_vq = zeros(numel(t),1);
delv2_vd = zeros(numel(t),1);
delv2_cont = zeros(numel(t),1);

TL_3 = zeros(numel(t),1);
Te_3 = zeros(numel(t),1);
% wa_3 = zeros(numel(t),1);
% theta_a_3 = zeros(numel(t),1);

delv3_vq = zeros(numel(t),1);
delv3_vd = zeros(numel(t),1);
delv3_cont = zeros(numel(t),1);

S1 = zeros(numel(t),1);
S2 = zeros(numel(t),1);
S3 = zeros(numel(t),1);

Sd1_2 = zeros(numel(t),1);
Sd2_2 = zeros(numel(t),1);
Sd3_2 = zeros(numel(t),1);

Sd1_3 = zeros(numel(t),1);
Sd2_3 = zeros(numel(t),1);
Sd3_3 = zeros(numel(t),1);

for i=1:numel(t)
    TL_1(i) = IM1_out{i}.TL;
    Te_1(i) = IM1_out{i}.Te;
%     wa_1(i) = IM1_out{i}.wa;
%     theta_a_1(i) = IM1_out{i}.theta_a;
```

```
277
278         vqC(i) = IM1_out{i}.vqC;
279         vdC(i) = IM1_out{i}.vdC;
280
281         TL_2(i) = IM2_out{i}.TL;
282         Te_2(i) = IM2_out{i}.Te;
283     %     wa_2(i) = IM2_out{i}.wa;
284     %     theta_a_2(i) = IM2_out{i}.theta_a;
285
286         delv2_vq(i) = IM2_out{i}.delv2_vq;
287         delv2_vd(i) = IM2_out{i}.delv2_vd;
288         delv2_cont(i) = IM2_out{i}.delv2_cont;
289
290         TL_3(i) = IM3_out{i}.TL;
291         Te_3(i) = IM3_out{i}.Te;
292     %     wa_3(i) = IM2_out{i}.wa;
293     %     theta_a_3(i) = IM2_out{i}.theta_a;
294
295         delv3_vq(i) = IM3_out{i}.delv3_vq;
296         delv3_vd(i) = IM3_out{i}.delv3_vd;
297         delv3_cont(i) = IM3_out{i}.delv3_cont;
298
299         S1(i) = Sw_out{i}.S(1);
300         S2(i) = Sw_out{i}.S(2);
301         S3(i) = Sw_out{i}.S(3);
302
303         Sd1_2(i) = Sw_out{i}.S_d2(1);
304         Sd2_2(i) = Sw_out{i}.S_d2(2);
305         Sd3_2(i) = Sw_out{i}.S_d2(3);
306
307         Sd1_3(i) = Sw_out{i}.S_d3(1);
308         Sd2_3(i) = Sw_out{i}.S_d3(2);
309         Sd3_3(i) = Sw_out{i}.S_d3(3);
310
311     end
312
313     %-------------------------------------------------------------------------%
314     clear y IM1_out IM2_out IM3_out Sw_out
315     %-------------------------------------------------------------------------%
316     %                   Auxiliary Equations for the IMs
317     %-------------------------------------------------------------------------%
318
319     % Transformation of electrical rotor speed and electrical angular position
320     % to mechanical rotor speed and mechanical angular position
321
322     % 1st IM
323     wm_1 =  wr_1/(ParametersIM.P/2);
324     theta_m_1 = theta_r_1/(ParametersIM.P/2);
325     % 2nd IM
326     wm_2 =  wr_2/(ParametersIM.P/2);
327     theta_m_2 = theta_r_2/(ParametersIM.P/2);
328     % 3rd IM
329     wm_3 =  wr_3/(ParametersIM.P/2);
330     theta_m_3 = theta_r_3/(ParametersIM.P/2);
331
332     % Transformation of fluxes to currents — DQ components
333     Linv = ParametersXF.Linv;
334     % 1ST MOTOR
335     [ids_1, iqs_1, idr_1, iqr_1, id2_1, iq2_1] = Fluxes2Currents_xf(lambda_dC_1, ...
336         lambda_qC_1, lambda_dr_1, lambda_qr_1, lambda_d2_1, lambda_q2_1, Linv);
337     % 2ND MOTOR
338     [ids_2, iqs_2, idr_2, iqr_2, id2_2, iq2_2] = Fluxes2Currents_xf(lambda_dC_2, ...
339         lambda_qC_2, lambda_dr_2, lambda_qr_2, lambda_d2_2, lambda_q2_2, Linv);
340     % 3RD MOTOR
341     [ids_3, iqs_3, idr_3, iqr_3, id2_3, iq2_3] = Fluxes2Currents_xf(lambda_dC_3, ...
342         lambda_qC_3, lambda_dr_3, lambda_qr_3, lambda_d2_3, lambda_q2_3, Linv);
343
344     % Transforming stator and rotor currents from qd reference to abc variables
345     % Disabled to reduce size of saved data — can convert later
346     % % 1st IM
347     % % [ias_1, ibs_1, ics_1] = qd2abcs(iqs_1, ids_1, theta_a_1);
348     % ias_1 = iqs_1;
349     % [ibs_1, ics_1] = statbc(iqs_1, ids_1);
350     % % [iar_1, ~, ~] = qd2abcs(iqr_1, idr_1, theta_a_1-theta_r_1);
351     % % [ia2_1, ib2_1, ic2_1] = qd2abcs(iq2_1, id2_1, theta_a_1);
352     % ia2_1 = iq2_1;
353     % [ib2_1, ic2_1] = statbc(iq2_1, id2_1);
354     % % 2nd IM
355     % % [ias_2, ibs_2, ics_2] = qd2abcs(iqs_2, ids_2, theta_a_2);
356     % ias_2 = iqs_2;
357     % [ibs_2, ics_2] = statbc(iqs_2, ids_2);
358     % % [iar_2, ~, ~] = qd2abcs(iqr_2, idr_2, theta_a_2-theta_r_2);
```

```
% % [ia2_2, ib2_2, ic2_2] = qd2abcs(iq2_2, id2_2, theta_a_2);
% ia2_2 = iq2_2;
% [ib2_2, ic2_2] = statbc(iq2_2, id2_2);
%
% % Transforming stator voltages from qd reference to abc variables
% % Main converter
% % [vaC, vbC, vcC] = qd2abcs(vqC, vdC, theta_a_1);
% vaC = vqC;
% [vbC, vcC] = statbc(vqC, vdC);
% % Voltage drop converter
% % [delv2_va, delv2_vb, delv2_vc] = qd2abcs(delv2_vq, delv2_vd, theta_a_2);
% delv2_va = delv2_vq;
% [delv2_vb, delv2_vc] = statbc(delv2_vq, delv2_vd);

% Voltage reaching motors: Need to calculate based on currents. v=Ldi/dt
% 1st IM
Dids_1 = [0; diff(ids_1)/deltat];
Diqs_1 = [0; diff(iqs_1)/deltat];
Did2_1 = [0; diff(id2_1)/deltat];
Diq2_1 = [0; diff(iq2_1)/deltat];

% lambda_d1_1 = (Ll1 + LmT)*ids_1 + LmT*id2_1;
% lambda_q1_1 = (Ll1 + LmT)*iqs_1 + LmT*iq2_1;

% vd1_1 = ParametersXF.r1*ids_1 + Ll1*Dids_1 + LmT*(Dids_1+Did2_1) - wa_1.*lambda_q1_1;
vd1_1 = ParametersXF.r1*ids_1 + Ll1*Dids_1 + LmT*(Dids_1+Did2_1);
vds_1 = vdC - vd1_1;
% vq1_1 = ParametersXF.r1*iqs_1 + Ll1*Diqs_1 + LmT*(Diqs_1+Diq2_1) + wa_1.*lambda_d1_1;
vq1_1 = ParametersXF.r1*iqs_1 + Ll1*Diqs_1 + LmT*(Diqs_1+Diq2_1);
vqs_1 = vqC - vq1_1;

% % [va1_1, vb1_1, vc1_1] = qd2abcs(vq1_1, vd1_1, theta_a_1);
% va1_1 = vq1_1;
% [vb1_1, vc1_1] = statbc(vq1_1, vd1_1);
% % [vas_1, vbs_1, vcs_1] = qd2abcs(vqs_1, vds_1, theta_a_1);
% vas_1 = vqs_1;
% [vbs_1, vcs_1] = statbc(vqs_1, vds_1);

% 2nd IM
Dids_2 = [0; diff(ids_2)/deltat];
Diqs_2 = [0; diff(iqs_2)/deltat];
Did2_2 = [0; diff(id2_2)/deltat];
Diq2_2 = [0; diff(iq2_2)/deltat];

% lambda_d1_2 = (Ll1 + LmT)*ids_2 + LmT*id2_2;
% lambda_q1_2 = (Ll1 + LmT)*iqs_2 + LmT*iq2_2;

% vd1_2 = ParametersXF.r1*ids_2 + Ll1*Dids_2 + LmT*(Dids_2+Did2_2) - wa_2.*lambda_q1_2;
vd1_2 = ParametersXF.r1*ids_2 + Ll1*Dids_2 + LmT*(Dids_2+Did2_2);
vds_2 = vdC - vd1_2;
% vq1_2 = ParametersXF.r1*iqs_2 + Ll1*Diqs_2 + LmT*(Diqs_2+Diq2_2) + wa_2.*lambda_d1_2;
vq1_2 = ParametersXF.r1*iqs_2 + Ll1*Diqs_2 + LmT*(Diqs_2+Diq2_2);
vqs_2 = vqC - vq1_2;

% % [va1_2, vb1_2, vc1_2] = qd2abcs(vq1_2, vd1_2, theta_a_2);
% va1_2 = vq1_2;
% [vb1_2, vc1_2] = statbc(vq1_2, vd1_2);
% % [vas_2, vbs_2, vcs_2] = qd2abcs(vqs_2, vds_2, theta_a_2);
% vas_2 = vqs_2;
% [vbs_2, vcs_2] = statbc(vqs_2, vds_2);

% 3rd IM
Dids_3 = [0; diff(ids_3)/deltat];
Diqs_3 = [0; diff(iqs_3)/deltat];
Did2_3 = [0; diff(id2_3)/deltat];
Diq2_3 = [0; diff(iq2_3)/deltat];

vd1_3 = ParametersXF.r1*ids_3 + Ll1*Dids_3 + LmT*(Dids_3+Did2_3);
vds_3 = vdC - vd1_3;
vq1_3 = ParametersXF.r1*iqs_3 + Ll1*Diqs_3 + LmT*(Diqs_3+Diq2_3);
vqs_3 = vqC - vq1_3;
%--------------------------------------------------------------------------%

% Clear variables not saved
clear lambda_dC_1 lambda_qC_1 lambda_dr_1 lambda_qr_1 lambda_d2_1 ...
    lambda_q2_1 lambda_dC_2 lambda_qC_2 lambda_dr_2 lambda_qr_2 ...
    lambda_d2_2 lambda_q2_2 lambda_dC_3 lambda_qC_3 lambda_dr_3 ...
    lambda_qr_3 lambda_d2_3 lambda_q2_3 theta_r_1 theta_r_2 theta_r_3 ...
    wr_1 wr_2 wr_3 Dids_1 Diqs_1 Did2_1 Diq2_1 Dids_2 Diqs_2 Did2_2 ...
    Diq2_2 Dids_3 Diqs_3 Did2_3 Diq2_3

% Save variables before next loop (motor 2, different xfmr params)
```

```
TL_2_out(:,j) = TL_2;
Te_2_out(:,j) = Te_2;
delv2_vd_out(:,j) = delv2_vd;
delv2_vq_out(:,j) = delv2_vq;
delv2_cont_out(:,j) = delv2_cont;
wm_2_out(:,j) = wm_2;
theta_m_2_out(:,j) = theta_m_2;
ids_2_out(:,j) = ids_2;
iqs_2_out(:,j) = iqs_2;
idr_2_out(:,j) = idr_2;
iqr_2_out(:,j) = iqr_2;
id2_2_out(:,j) = id2_2;
iq2_2_out(:,j) = iq2_2;
vq1_2_out(:,j) = vq1_2;
vd1_2_out(:,j) = vd1_2;
vqs_2_out(:,j) = vqs_2;
vds_2_out(:,j) = vds_2;
Sd1_2_out(:,j) = Sd1_2;
Sd2_2_out(:,j) = Sd2_2;
Sd3_2_out(:,j) = Sd3_2;

% Save variables before next loop (motor 3, different xfmr params)
TL_3_out(:,j) = TL_3;
Te_3_out(:,j) = Te_3;
delv3_vd_out(:,j) = delv3_vd;
delv3_vq_out(:,j) = delv3_vq;
delv3_cont_out(:,j) = delv3_cont;
wm_3_out(:,j) = wm_3;
theta_m_3_out(:,j) = theta_m_3;
ids_3_out(:,j) = ids_3;
iqs_3_out(:,j) = iqs_3;
idr_3_out(:,j) = idr_3;
iqr_3_out(:,j) = iqr_3;
id2_3_out(:,j) = id2_3;
iq2_3_out(:,j) = iq2_3;
vq1_3_out(:,j) = vq1_3;
vd1_3_out(:,j) = vd1_3;
vqs_3_out(:,j) = vqs_3;
vds_3_out(:,j) = vds_3;
Sd1_3_out(:,j) = Sd1_3;
Sd2_3_out(:,j) = Sd2_3;
Sd3_3_out(:,j) = Sd3_3;

% Save variables before next loop (motor 1, different xfmr params)
TL_1_out(:,j) = TL_1;
Te_1_out(:,j) = Te_1;
wm_1_out(:,j) = wm_1;
theta_m_1_out(:,j) = theta_m_1;
ids_1_out(:,j) = ids_1;
iqs_1_out(:,j) = iqs_1;
idr_1_out(:,j) = idr_1;
iqr_1_out(:,j) = iqr_1;
id2_1_out(:,j) = id2_1;
iq2_1_out(:,j) = iq2_1;
vq1_1_out(:,j) = vq1_1;
vd1_1_out(:,j) = vd1_1;
vqs_1_out(:,j) = vqs_1;
vds_1_out(:,j) = vds_1;
S1_out(:,j) = S1;
S2_out(:,j) = S2;
S3_out(:,j) = S3;

theta_e_out(:,j) = theta_e;

end

% Clear redundant variables left over
clear delv2_cont delv2_vd delv2_vq delv3_cont delv3_vd delv3_vq i id2_1 ...
    id2_2 id2_3 idr_1 idr_2 idr_3 ids_1 ids_2 ids_3 iq2_1 iq2_2 iq2_3 ...
    iqr_1 iqr_2 iqr_3 iqs_1 iqs_2 iqs_3 j k N num_states S1 S2 S3 ...
    Sd1_2 Sd2_2 Sd3_2 Sd1_3 Sd2_3 Sd3_3 t_end t_ini tspan Te_1 Te_2 Te_3 ...
    theta_m_1 theta_m_2 theta_m_3 TL_1 TL_2 TL_3 vd1_1 vd1_2 vd1_3 ...
    vdC vds_1 vds_2 vds_3 vq1_1 vq1_2 vq1_3 vqC vqs_1 vqs_2 vqs_3 ...
    wm_1 wm_2 wm_3 x0 deltat theta_e
%-------------------------------------------------------------------------%
```

```
function [y, IM1, IM2, IM3, Sw] = ThreeIM_1Converter_CVHz_volt_xf(t, u, ...
    ParametersIM, ParametersXF, ParamSys, ParamController)
%-------------------------------------------------------------------------%
%                           STATES
%-------------------------------------------------------------------------%
%                           1ST MOTOR
```

```
%------------------------------------------------------------------------------%
lambda_dC_1 = u(1);  % Flux  dC (ds + d1)                                 [Wb]
lambda_qC_1 = u(2);  % Flux  qC (qs + q1)                                 [Wb]
lambda_dr_1 = u(3);  % Flux  dr                                          [Wb]
lambda_qr_1 = u(4);  % Flux  qr                                          [Wb]
lambda_d2_1 = u(5);  % Flux  d2 (xfmr side converter)                    [Wb]
lambda_q2_1 = u(6);  % Flux  q2                                          [Wb]
wr_1        = u(7);  % Rotor speed (electric)                         [rad/s]
theta_r_1   = u(8);  % Rotor position (electric)                        [rad]
%------------------------------------------------------------------------------%
%              COMPENSATED VOLTS-PER-HERZ CONTROL
%------------------------------------------------------------------------------%
theta_e = u(9);    % Electrical angular position [rad]
y_LPF   = u(10);    % Output of the low-pass filter
%------------------------------------------------------------------------------%
%------------------------------------------------------------------------------%
%                            2ND MOTOR
%------------------------------------------------------------------------------%
lambda_dC_2 = u(11);  % Flux  dC (ds + d1)                                [Wb]
lambda_qC_2 = u(12);  % Flux  qC (qs + q1)                                [Wb]
lambda_dr_2 = u(13);  % Flux  dr                                          [Wb]
lambda_qr_2 = u(14);  % Flux  qr                                          [Wb]
lambda_d2_2 = u(15);  % Flux  d2 (xfmr side converter)                    [Wb]
lambda_q2_2 = u(16);  % Flux  q2                                          [Wb]
wr_2        = u(17);  % Rotor speed (electric)                         [rad/s]
theta_r_2   = u(18);  % Rotor position (electric)                        [rad]
%------------------------------------------------------------------------------%
%------------------------------------------------------------------------------%
%                            3RD MOTOR
%------------------------------------------------------------------------------%
lambda_dC_3 = u(19);  % Flux  dC (ds + d1)                                [Wb]
lambda_qC_3 = u(20);  % Flux  qC (qs + q1)                                [Wb]
lambda_dr_3 = u(21);  % Flux  dr                                          [Wb]
lambda_qr_3 = u(22);  % Flux  qr                                          [Wb]
lambda_d2_3 = u(23);  % Flux  d2 (xfmr side converter)                    [Wb]
lambda_q2_3 = u(24);  % Flux  q2                                          [Wb]
wr_3        = u(25);  % Rotor speed (electric)                         [rad/s]
theta_r_3   = u(26);  % Rotor position (electric)                        [rad]
%------------------------------------------------------------------------------%
%------------------------------------------------------------------------------%
%                         CONTROLLER
%------------------------------------------------------------------------------%
n = ParamController.order;
% Automatic state-building of the controller, based on its order
xk2 = u(27:27+n-1);
xk2 = reshape(xk2,n,1);
xk3 = u(27+n:27+2*n-1);
xk3 = reshape(xk3,n,1);
%------------------------------------------------------------------------------%

%------------------------------------------------------------------------------%
%                           GET SYSPARAMS
%------------------------------------------------------------------------------%

%------------------------------------------------------------------------------%
vswM = ParamSys.vswM;
vdM = ParamSys.vdM;
vswD = ParamSys.vswD;
vdD = ParamSys.vdD;

TL1_delta  = ParamSys.TL1_variation;
time_delta_TL1 = ParamSys.time_variation_TL1;

TL2_delta  = ParamSys.TL2_variation;
time_delta_TL2 = ParamSys.time_variation_TL2;

TL3_delta  = ParamSys.TL3_variation;
time_delta_TL3 = ParamSys.time_variation_TL3;

if t > time_delta_TL1
    TL_1 = (1+TL1_delta)*ParametersIM.Tnom;
else
    %     TL_1 = ParametersIM.Tnom;
    TL_1 = 0;
end

if t > time_delta_TL2
    TL_2 = (1+TL2_delta)*ParametersIM.Tnom;
else
%     TL_2 = ParametersIM.Tnom;
    TL_2 = 0;
end
```

```
89
90  if t > time_delta_TL3
91      TL_3 = (1+TL3_delta)*ParametersIM.Tnom;
92  else
93  %     TL_3 = ParametersIM.Tnom;
94      TL_3 = 0;
95  end
96  %--------------------------------------------------------------------------%
97
98  %--------------------------------------------------------------------------%
99  %                   Arbitrary Reference Frame (unused)
100 %--------------------------------------------------------------------------%
101 %                            Stator reference
102 %--------------------------------------------------------------------------%
103 % 1st IM
104 % wa_1 = 0;
105 % theta_a_1 = 0;
106
107 % 2nd IM
108 % wa_2 = 0;
109 % theta_a_2 = 0;
110 %--------------------------------------------------------------------------%
111
112 %--------------------------------------------------------------------------%
113 %                    Auxiliary Equations for the IMs
114 %--------------------------------------------------------------------------%
115 % Transformation of fluxes to currents - DQ components
116 Linv = ParametersXF.Linv;
117 % 1ST MOTOR
118 [ids_1, iqs_1, idr_1, iqr_1, id2_1, iq2_1] = Fluxes2Currents_xf(lambda_dC_1, ...
119     lambda_qC_1, lambda_dr_1, lambda_qr_1, lambda_d2_1, lambda_q2_1, Linv);
120 % 2ND MOTOR
121 [ids_2, iqs_2, idr_2, iqr_2, id2_2, iq2_2] = Fluxes2Currents_xf(lambda_dC_2, ...
122     lambda_qC_2, lambda_dr_2, lambda_qr_2, lambda_d2_2, lambda_q2_2, Linv);
123 % 3RD MOTOR
124 [ids_3, iqs_3, idr_3, iqr_3, id2_3, iq2_3] = Fluxes2Currents_xf(lambda_dC_3, ...
125     lambda_qC_3, lambda_dr_3, lambda_qr_3, lambda_d2_3, lambda_q2_3, Linv);
126
127 % Electromagnetic and Load Torques
128 % Need to use rotor flux & current since stator flux isn't calculated
129 % 1ST MOTOR
130 Te_1 = 3/4*ParametersIM.P*(lambda_qr_1.*idr_1 - lambda_dr_1.*iqr_1);
131 % 2ND MOTOR
132 Te_2 = 3/4*ParametersIM.P*(lambda_qr_2.*idr_2 - lambda_dr_2.*iqr_2);
133 % 3RD MOTOR
134 Te_3 = 3/4*ParametersIM.P*(lambda_qr_3.*idr_3 - lambda_dr_3.*iqr_3);
135
136 % Transformation of electrical rotor speed and electrical angular position
137 % to mechanical rotor speed and mechanical angular position
138 % 1st IM
139 % wm_1 =  wr_1/(ParametersIM.P/2);
140 theta_m_1 = theta_r_1/(ParametersIM.P/2);
141 % 2nd IM
142 % wm_2 =  wr_2/(ParametersIM.P/2);
143 theta_m_2 = theta_r_2/(ParametersIM.P/2);
144 % 3rd IM
145 % wm_2 =  wr_2/(ParametersIM.P/2);
146 theta_m_3 = theta_r_3/(ParametersIM.P/2);
147
148 % Transforming stator and rotor currents from qd reference to abc variables
149 % Replaced with stationary assumption for efficiency
150 % 1st IM
151 % [ias_1, ibs_1, ics_1] = qd2abcs(iqs_1, ids_1, theta_a_1);
152 [ibs_1, ics_1] = statbc(iqs_1, ids_1);
153 % 2nd IM
154 %[ias_2, ibs_2, ics_2] = qd2abcs(iqs_2, ids_2, theta_a_2);
155 [ibs_2, ics_2] = statbc(iqs_2, ids_2);
156 % 3rd IM
157 %[ias_3, ibs_3, ics_3] = qd2abcs(iqs_3, ids_3, theta_a_3);
158 [ibs_3, ics_3] = statbc(iqs_3, ids_3);
159
160 % 1st XFMR
161 % [ia2_1, ib2_1, ic2_1] = qd2abcs(iq2_1, id2_1, theta_a_1);
162 [ib2_1, ic2_1] = statbc(iq2_1, id2_1);
163 % 2nd XFMR
164 % [ia2_2, ib2_2, ic2_2] = qd2abcs(iq2_2, id2_2, theta_a_2);
165 [ib2_2, ic2_2] = statbc(iq2_2, id2_2);
166 % 3rd XFMR
167 % [ia2_3, ib2_3, ic2_3] = qd2abcs(iq2_3, id2_3, theta_a_3);
168 [ib2_3, ic2_3] = statbc(iq2_3, id2_3);
169
170 % Rotor Voltages (squirrel cage) -  DQ components
```

```
% 1st IM
vdr_1 = 0;
vqr_1 = 0;
% 2nd IM
vdr_2 = 0;
vqr_2 = 0;
% 3rd IM
vdr_3 = 0;
vqr_3 = 0;

% Transformer secondary voltages are defined in position sync section
%--------------------------------------------------------------------------%

%--------------------------------------------------------------------------%
%               Low-Pass Filter Definitions/Equations
%--------------------------------------------------------------------------%
% The signal x_corr = 3*P*(vqs_e_star*iqs_e - 2*rs*Is^2)/Ktv (Eq. 14.2-16
% from Krause) is filtered using a low-pass filter (LPF), accordingly to
% Krause's suggestion. In order to implement this filtering process, a new
% state, y, was created, from the diff. equation of a LPF:
% dy/dt = (x-y)/tau,
% where x is the filter input and y is the filter output, while tau is the
% filter time constant.
% The LPF output is the signal X_corr, used in the Volts-per-Hertz
% compensated control section down below.
% The input of the filter is set in the Volts-per-Hertz compensated control
% section down below, since, as seen above, it depends on vqs_e_star, which
% in turn is defined by the Volts-per-Herz compensated control equations.
%--------------------------------------------------------------------------%
% Time constant of the filter
tau_LPF = 0.1;

% Output of the LPF filter
X_corr = y_LPF;
%--------------------------------------------------------------------------%

%--------------------------------------------------------------------------%
%                 Compensated Volts-per-Herz Control
%--------------------------------------------------------------------------%

% Transforming from iqds_s to iqds_e
[iqs_e_1, ids_e_1] = qd2qd(iqs_1, ids_1, theta_e);

% Rated voltage (line to neutral rms) and rated radian frequency
Vb = ParametersIM.Vs/sqrt(3);
wb = ParametersIM.f*2*pi;

% Commanded mechanical speed
wrm_star = DesiredSpeed(t);

% Mechanical commanded speed to electrical commanded speed
wr_star = wrm_star*ParametersIM.P/2;

% Eq 14.2-7 (Krause)
% Equation from Krause was wrong (commented version). The right one, after
% checking 2 times, with 2 different approachs, is the one used here!
% num_Ktv = 3*ParametersIM.P/2*ParametersIM.Lm^2*ParametersIM.rr*Vb^2;
% den_Ktv = ParametersIM.rr*2*(ParametersIM.rs^2+wb^2*ParametersIM.Ls^2);
num_Ktv = 3*ParametersIM.P*ParametersIM.Lm^2*Vb^2;
den_Ktv = ParametersIM.rr*2*(ParametersIM.rs^2+wb^2*ParametersIM.Ls^2);
Ktv = num_Ktv/den_Ktv;

% Eq. 14.2-12 (Krause)
Is = sqrt(iqs_e_1^2 + ids_e_1^2)/sqrt(2);

% Eq. 14.2-14 (Krause)
we = (wr_star + sqrt(max([0 wr_star^2+X_corr])))/2;

% Eq 14.2-4 (Krause)
Vs = (wr_star ~= 0)*Vb*sqrt((ParametersIM.rs^2+we^2*ParametersIM.Ls^2)/(ParametersIM.rs^2+wb^2*ParametersIM.Ls^2));

% Saturation on Vs
Vs = min(Vb,Vs);

% Placing the voltage arbitrarily in the q axis
vqs_e_star = Vs*sqrt(2);
vds_e_star = 0;

% Eq 14.2-16 (Krause)
x_corr = 3*ParametersIM.P*(vqs_e_star*iqs_e_1 - 2*ParametersIM.rs*Is^2)/Ktv;

% Input of the LPF filter
```

```
u_LPF = x_corr;

% Transforming vds_e_star and vqs_e_star to vabcs
% [vas_star, vbs_star, vcs_star] = qd2abcs(vqs_e_star,vds_e_star,theta_e);

% Converter (voltage control)
f_sw = 3e3; % inverter switching frequency
vdc = ParametersIM.Vs*sqrt(2); % vdc got by a 3-phase full-bridge rectifier
[S,~,~,~,~] = my3HarmonicSineTri_v2(t, theta_e, vqs_e_star, vds_e_star, vdc, f_sw);
% IM = [(ias_1+ias_2) (ibs_1+ibs_2) (ics_1+ics_2)];
IM = [(iqs_1+iqs_2+iqs_3) (ibs_1+ibs_2+ibs_3) (ics_1+ics_2+ics_3)];
[vaC_1, vbC_1, vcC_1] = my3PhaseConverterVoltLoss(S, vdc, vswM, vdM, IM);

% Ideal voltage source
% vas_1 = vas_star;
% vbs_1 = vbs_star;
% vcs_1 = vcs_star;

% Transforming vabcs to vds and vqs
% Replaced with stationary assumption for efficiency
% [vqC_1, vdC_1] = abcs2qd(vaC_1, vbC_1, vcC_1, theta_a_1);
[vqC_1, vdC_1] = statqd(vaC_1, vbC_1, vcC_1);
%--------------------------------------------------------------------------%

%--------------------------------------------------------------------------%
%                      VOLTAGE ADJUSTMENT CONTROL
%                          INDUCTION MOTOR 2
%--------------------------------------------------------------------------%

%--------------------------------------------------------------------------%
%                     CONTROLLER EQUATIONS
%--------------------------------------------------------------------------%
% Controller input vector uk
% For now, only the position error

Ak = ParamController.Ak;
Bk = ParamController.Bk;
Ck = ParamController.Ck;
Dk = ParamController.Dk;

uk2(1) = theta_m_1 - theta_m_2;

xk2_dot = Ak*xk2 + Bk*uk2;
yk2 = Ck*xk2 + Dk*uk2;

uk3(1) = theta_m_1 - theta_m_3;

xk3_dot = Ak*xk3 + Bk*uk3;
yk3 = Ck*xk3 + Dk*uk3;

% Controller output vector yk
% For now, only motor 2/3 voltage adjustment
% delv2_cont: continuous variable voltage drop, ideal target
% For PI controller, yk is in units of V/rad
delv2_cont = yk2(1);
delv2_cont = max(delv2_cont,0);
delv2_cont = min(delv2_cont,ParamController.delv_max);

delv3_cont = yk3(1);
delv3_cont = max(delv3_cont,0);
delv3_cont = min(delv3_cont,ParamController.delv_max);
%--------------------------------------------------------------------------%

% Calculate voltage drops for motor 1 using dummy converter for symmetry
S_d2 = [false false false true true true];
% ID = [ia2_1, ib2_1, ic2_1];
ID = [iq2_1, ib2_1, ic2_1];
[vad, vbd, vcd] = my3PhaseConverterVoltLoss(S_d2, 0, vswD, vdD, ID);
% Transforming vabcs to vds and vqs
% Replaced with stationary assumption for efficiency
% [vqd, vdd] = abcs2qd(vad, vbd, vcd, theta_a_1);
[vqd_1, vdd_1] = statqd(vad, vbd, vcd);
% Refer variable
vq2_1 = (vqd_1*ParametersXF.N);
vd2_1 = (vdd_1*ParametersXF.N);
if t > ParamController.tstart
    % Controlable voltage drop via sine-triangle PWM, assuming synchronization
    f_d = 4.987654321e3;
    vdc_d = vdc * ParametersXF.vdcR;

    % Motor 2
    vqd_e = delv2_cont / ParametersXF.N;
```

```
    vdd_e = 0;
    [S_d2,~,~,~,~] = my3HarmonicSineTri_v2(t, theta_e, vqd_e, vdd_e, vdc_d, f_d);
    % ID = [ia2_2, ib2_2, ic2_2];
    ID = [iq2_2, ib2_2, ic2_2];
    [vad, vbd, vcd] = my3PhaseConverterVoltLoss(S_d2, vdc_d, vswD, vdD, ID);

    % Transforming vabcs to vds and vqs
    % Replaced with stationary assumption for efficiency
    % [vqd, vdd] = abcs2qd(vad, vbd, vcd, theta_a_2);
    [vqd_2, vdd_2] = statqd(vad, vbd, vcd);

    % Refer variable
    vq2_2 = (vqd_2*ParametersXF.N);
    vd2_2 = (vdd_2*ParametersXF.N);

    % Motor 3
    vqd_e = delv3_cont / ParametersXF.N;
    vdd_e = 0;
    [S_d3,~,~,~,~] = my3HarmonicSineTri_v2(t, theta_e, vqd_e, vdd_e, vdc_d, f_d);
    % ID = [ia2_3, ib2_3, ic2_3];
    ID = [iq2_3, ib2_3, ic2_3];
    [vad, vbd, vcd] = my3PhaseConverterVoltLoss(S_d3, vdc_d, vswD, vdD, ID);

    % Transforming vabcs to vds and vqs
    % Replaced with stationary assumption for efficiency
    % [vqd, vdd] = abcs2qd(vad, vbd, vcd, theta_a_2);
    [vqd_3, vdd_3] = statqd(vad, vbd, vcd);

    % Refer variable
    vq2_3 = (vqd_3*ParametersXF.N);
    vd2_3 = (vdd_3*ParametersXF.N);
else
    % Voltage drop is only active when torques are different
    % Since motors are identical in this state, use same voltages
    S_d3 = S_d2;
    vqd_2 = vqd_1;
    vdd_2 = vdd_1;
    vqd_3 = vqd_1;
    vdd_3 = vdd_1;
    vq2_2 = vq2_1;
    vd2_2 = vd2_1;
    vq2_3 = vq2_1;
    vd2_3 = vd2_1;
end
%-------------------------------------------------------------------------------%

%-------------------------------------------------------------------------------%
%                       OUTPUT STRUCTURES
%-------------------------------------------------------------------------------%
% Switching states
Sw.S = S;
Sw.S_d2 = S_d2;
Sw.S_d3 = S_d3;

% 1st motor
IM1.TL = TL_1;
IM1.Te = Te_1;
IM1.vqC = vqC_1;
IM1.vdC = vdC_1;
IM1.we = we;
% IM1.wa = wa_1;
IM1.theta_e = theta_e;
% IM1.theta_a = theta_a_1;
IM1.delv1_vq = vqd_1;
IM1.delv1_vd = vdd_1;

% 2nd motor
IM2.TL = TL_2;
IM2.Te = Te_2;
% IM2.wa = wa_2;
% IM2.theta_a = theta_a_2;
IM2.delv2_vq = vqd_2;
IM2.delv2_vd = vdd_2;
IM2.delv2_cont = delv2_cont;

% 3rd motor
IM3.TL = TL_3;
IM3.Te = Te_3;
% IM3.wa = wa_3;
% IM3.theta_a = theta_a_3;
IM3.delv3_vq = vqd_3;
IM3.delv3_vd = vdd_3;
```

```
IM3.delv3_cont = delv3_cont;

%-------------------------------------------------------------------------%


%-------------------------------------------------------------------------%
%                     DIFF. STATES EQUATIONS
%-------------------------------------------------------------------------%
% Extracting parameters here, assuming identical motors
rs = ParametersIM.rs;
rr = ParametersIM.rr;
r1 = ParametersXF.r1;
r2 = ParametersXF.r2;
J = ParametersIM.J;
P = ParametersIM.P;
b = ParametersIM.b;

% Equations used are altered to assume stationary ref. frame for efficiency
%-------------------------------------------------------------------------%
%                          1ST MOTOR
%-------------------------------------------------------------------------%
% y(1) = vdC_1 - (rs+r1)*ids_1 + wa_1 * lambda_qC_1;      % lambda_dC_1
% y(2) = vqC_1 - (rs+r1)*iqs_1 - wa_1 * lambda_dC_1;      % lambda_qC_1
% y(3) = vdr_1 - rr*idr_1 + (wa_1-wr_1)*lambda_qr_1;      % lambda_dr_1
% y(4) = vqr_1 - rr*iqr_1 - (wa_1-wr_1)*lambda_dr_1;      % lambda_qr_1
% y(5) = vd2_1 - r2*id2_1 + wa_1 * lambda_q2_1;           % lambda_d2_1
% y(6) = vq2_1 - r2*iq2_1 - wa_1 * lambda_d2_1;           % lambda_q2_1
y(1) = vdC_1 - (rs+r1)*ids_1;                   % lambda_dC_1
y(2) = vqC_1 - (rs+r1)*iqs_1;                   % lambda_qC_1
y(3) = vdr_1 - rr*idr_1 - wr_1*lambda_qr_1;     % lambda_dr_1
y(4) = vqr_1 - rr*iqr_1 + wr_1*lambda_dr_1;     % lambda_qr_1
y(5) = vd2_1 - r2*id2_1;                        % lambda_d2_1
y(6) = vq2_1 - r2*iq2_1;                        % lambda_q2_1
y(7) = (1/J)*((Te_1-TL_1)*P/2 - b*wr_1);        % wr_1
y(8) = wr_1;                                    % theta_r_1
%-------------------------------------------------------------------------%
%              COMPENSATED VOLTS-PER-HERTZ CONTROL
%-------------------------------------------------------------------------%
y(9) = we;                      % Electrical angular position
y(10) = (u_LPF - y_LPF)/tau_LPF; % Output of the low-pass filter
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
%                          2ND MOTOR
%-------------------------------------------------------------------------%
% y(11) = vdC_1 - (rs+r1)*ids_2 + wa_2 * lambda_qC_2;      % lambda_dC_2
% y(12) = vqC_1 - (rs+r1)*iqs_2 - wa_2 * lambda_dC_2;      % lambda_qC_2
% y(13) = vdr_2 - rr*idr_2 + (wa_2-wr_2)*lambda_qr_2;      % lambda_dr_2
% y(14) = vqr_2 - rr*iqr_2 - (wa_2-wr_2)*lambda_dr_2;      % lambda_qr_2
% y(15) = vd2_2 - r2*id2_2 + wa_2 * lambda_q2_2;           % lambda_d2_2
% y(16) = vq2_2 - r2*iq2_2 - wa_2 * lambda_d2_2;           % lambda_q2_2
y(11) = vdC_1 - (rs+r1)*ids_2;                  % lambda_dC_2
y(12) = vqC_1 - (rs+r1)*iqs_2;                  % lambda_qC_2
y(13) = vdr_2 - rr*idr_2 - wr_2*lambda_qr_2;    % lambda_dr_2
y(14) = vqr_2 - rr*iqr_2 + wr_2*lambda_dr_2;    % lambda_qr_2
y(15) = vd2_2 - r2*id2_2;                       % lambda_d2_2
y(16) = vq2_2 - r2*iq2_2;                       % lambda_q2_2
y(17) = (1/J)*((Te_2-TL_2)*P/2 - b*wr_2);       % wr_2
y(18) = wr_2;                                   % theta_r_2
%-------------------------------------------------------------------------%
%                          3RD MOTOR
%-------------------------------------------------------------------------%
y(19) = vdC_1 - (rs+r1)*ids_3;                  % lambda_dC_3
y(20) = vqC_1 - (rs+r1)*iqs_3;                  % lambda_qC_3
y(21) = vdr_3 - rr*idr_3 - wr_3*lambda_qr_3;    % lambda_dr_3
y(22) = vqr_3 - rr*iqr_3 + wr_3*lambda_dr_3;    % lambda_qr_3
y(23) = vd2_3 - r2*id2_3;                       % lambda_d2_3
y(24) = vq2_3 - r2*iq2_3;                       % lambda_q2_3
y(25) = (1/J)*((Te_3-TL_3)*P/2 - b*wr_3);       % wr_3
y(26) = wr_3;                                   % theta_r_3
%-------------------------------------------------------------------------%
%                        CONTROLLER
%-------------------------------------------------------------------------%
% Automatic state-building diff. equations of the controller, based on its order
y(27:27+n-1) = xk2_dot;
y(27+n:27+2*n-1) = xk3_dot;
%-------------------------------------------------------------------------%

y = y';
end
```

B.3 IDFOC Control, External Resistor Method

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Multirun_3IM_IFOC_res.m
% Runs 3-motor simulation of resistor sync method using IDFOC control
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear
close all
clc

% load k_hinf
% [Ak,Bk,Ck,Dk] = ssdata(k_hinf);

Kp = -30; % 1/rad
Ki = -60; % 1/(rad*s)
[Ak,Bk,Ck,Dk] = ssdata(tf([Kp Ki],[1 0]));

% Code currently only supports 3 identical motors
ParametersIM = ReturnParametersIM_15hp;

rs = ParametersIM.rs;
rr = ParametersIM.rr;
Lls = ParametersIM.Lls;
Llr = ParametersIM.Llr;
Lm = ParametersIM.Lm;
Ls = ParametersIM.Ls;
Lr = ParametersIM.Lr;
% sigma = 1 - Lm^2/(Ls*Lr);
% tau_s = Ls/rs;
% tau_r = Lr/rr;
% sigma_s = tau_s*sigma;
% sigma_r = tau_r*sigma;
% mu_s = 1/sigma_s;
% mu_r = 1/sigma_r;

% Incorporating mu_s inside the param structure
% ParametersIM.mu_s = mu_s;

ParamController.h = 0.1; % Hysteresis band

ParamController.Ak = Ak;
ParamController.Bk = Bk;
ParamController.Ck = Ck;
ParamController.Dk = Dk;
ParamController.order = length(Ak);

% ParamController.rext_max = ParametersIM_2.rs*100;
ParamController.rext_max = 1.5;

% Parameters for switching losses
ParamSys.vswM = 0; % Main converter switch voltage
ParamSys.vdM = 0;  % Main converter diode voltage
ParamSys.vswR = 0; % Resistor board switch voltage
ParamSys.vdR = 0;  % Resistor board diode voltage

TL1_variation = 0; % Variation in load torque in pu
TL2_variation = -0.2;
TL3_variation = -0.3;

time_variation_TL1 = 2;
time_variation_TL2 = 2;
time_variation_TL3 = 2;

ParamSys.TL1_variation = TL1_variation;
ParamSys.TL2_variation = TL2_variation;
ParamSys.TL3_variation = TL3_variation;
ParamSys.time_variation_TL1 = time_variation_TL1;
ParamSys.time_variation_TL2 = time_variation_TL2;
ParamSys.time_variation_TL3 = time_variation_TL3;

num_states = 20 + 2*ParamController.order;
x0 = zeros(1,num_states); % Initial conditions for state variables
% load x0_save
% x0 = x0_save;

t_ini = 0;
t_end = 6;

deltat = 10e-6;
tspan = t_ini:deltat:t_end;
options = odeset('MaxStep', deltat, 'RelTol',1e-5, 'AbsTol', 1e-6);
% options = odeset();
```

```
81
82  tic
83  [t,y] = ode45(@(t,y) ThreeIM_1Converter_IFOC_res(t,y,ParametersIM, ...
84      ParamSys, ParamController), tspan, x0, options);
85  toc
86
87  % Save state
88  % x0_save = y(end,:);
89  % save x0_save x0_save
90
91  %-------------------------------------------------------------------------%
92  %                           POST-PROCESSING
93  %-------------------------------------------------------------------------%
94
95  %-------------------------------------------------------------------------%
96  %                           PICKING UP THE STATES
97  %-------------------------------------------------------------------------%
98  %                           1ST MOTOR
99  %-------------------------------------------------------------------------%
100 lambda_ds_1   =   y(:,1);  % Flux ds                                  [Wb]
101 lambda_qs_1   =   y(:,2);  % Flux qs                                  [Wb]
102 lambda_dr_1   =   y(:,3);  % Flux dr                                  [Wb]
103 lambda_qr_1   =   y(:,4);  % Flux qr                                  [Wb]
104 wr_1          =   y(:,5);  % Rotor speed (electric)                [rad/s]
105 theta_r_1     =   y(:,6);  % Rotor position (electric)               [rad]
106 %-------------------------------------------------------------------------%
107 %              INDIRECT FIELD ORIENTED CONTROL (1ST MOTOR)
108 %-------------------------------------------------------------------------%
109 R_theta_e  = y(:,7); % Rotor flux angular position (electric)    [rad]
110 % I_Action_CL_w_1 = y(:,8); % Integral action of the closed loop control of wm
111 %-------------------------------------------------------------------------%
112 %-------------------------------------------------------------------------%
113 %                             2ND MOTOR
114 %-------------------------------------------------------------------------%
115 lambda_ds_2   =   y(:,9);   % Flux ds                                  [Wb]
116 lambda_qs_2   =   y(:,10);  % Flux qs                                  [Wb]
117 lambda_dr_2   =   y(:,11);  % Flux dr                                  [Wb]
118 lambda_qr_2   =   y(:,12);  % Flux qr                                  [Wb]
119 wr_2          =   y(:,13);  % Rotor speed (electric)                [rad/s]
120 theta_r_2      =  y(:,14);  % Rotor position (electric)               [rad]
121 %-------------------------------------------------------------------------%
122 %-------------------------------------------------------------------------%
123 %                             3RD MOTOR
124 %-------------------------------------------------------------------------%
125 lambda_ds_3   =   y(:,15);  % Flux ds                                  [Wb]
126 lambda_qs_3   =   y(:,16);  % Flux qs                                  [Wb]
127 lambda_dr_3   =   y(:,17);  % Flux dr                                  [Wb]
128 lambda_qr_3   =   y(:,18);  % Flux qr                                  [Wb]
129 wr_3          =   y(:,19);  % Rotor speed (electric)                [rad/s]
130 theta_r_3      =  y(:,20);  % Rotor position (electric)               [rad]
131 %-------------------------------------------------------------------------%
132
133 %-------------------------------------------------------------------------%
134 %          EXTRACTING OUTPUT STRUCTURES FROM DIFF EQ FUNCTION
135 %-------------------------------------------------------------------------%
136 [~, IM1_out, IM2_out, IM3_out, Sw_out] = cellfun(@(t,y) ...
137     ThreeIM_1Converter_IFOC_res(t, y, ParametersIM, ParamSys, ...
138     ParamController), num2cell(t), num2cell(y,2),'uni',0);
139
140 R_vqsC = zeros(numel(t),1);
141 R_vdsC = zeros(numel(t),1);
142
143 R_TL_1 = zeros(numel(t),1);
144 R_Te_1 = zeros(numel(t),1);
145 theta_a_1 = zeros(numel(t),1);
146 R_vqs_1 = zeros(numel(t),1);
147 R_vds_1 = zeros(numel(t),1);
148
149 R_TL_2 = zeros(numel(t),1);
150 R_Te_2 = zeros(numel(t),1);
151 theta_a_2 = zeros(numel(t),1);
152 R_vqs_2 = zeros(numel(t),1);
153 R_vds_2 = zeros(numel(t),1);
154 rext2_pwm = zeros(numel(t),1);
155 rext2_cont = zeros(numel(t),1);
156
157 R_TL_3 = zeros(numel(t),1);
158 R_Te_3 = zeros(numel(t),1);
159 theta_a_3 = zeros(numel(t),1);
160 R_vqs_3 = zeros(numel(t),1);
161 R_vds_3 = zeros(numel(t),1);
162 rext3_pwm = zeros(numel(t),1);
```

```
rext3_cont = zeros(numel(t),1);

R_S1 = zeros(numel(t),1);
R_S2 = zeros(numel(t),1);
R_S3 = zeros(numel(t),1);

for i=1:numel(t)
    R_vqsC(i) = IM1_out{i}.vqsC;
    R_vdsC(i) = IM1_out{i}.vdsC;

    R_TL_1(i) = IM1_out{i}.TL;
    R_Te_1(i) = IM1_out{i}.Te;
%    theta_a_1(i) = IM1_out{i}.theta_a;

    R_vqs_1(i) = IM1_out{i}.vqs;
    R_vds_1(i) = IM1_out{i}.vds;

    R_TL_2(i) = IM2_out{i}.TL;
    R_Te_2(i) = IM2_out{i}.Te;
%    theta_a_2(i) = IM2_out{i}.theta_a;

    R_vqs_2(i) = IM2_out{i}.vqs;
    R_vds_2(i) = IM2_out{i}.vds;

    rext2_pwm(i) = IM2_out{i}.rext_pwm;
    rext2_cont(i) = IM2_out{i}.rext_cont;

    R_TL_3(i) = IM3_out{i}.TL;
    R_Te_3(i) = IM3_out{i}.Te;
%    theta_a_3(i) = IM3_out{i}.theta_a;

    R_vqs_3(i) = IM3_out{i}.vqs;
    R_vds_3(i) = IM3_out{i}.vds;

    rext3_pwm(i) = IM3_out{i}.rext_pwm;
    rext3_cont(i) = IM3_out{i}.rext_cont;

    R_S1(i) = Sw_out{i}.S(1);
    R_S2(i) = Sw_out{i}.S(2);
    R_S3(i) = Sw_out{i}.S(3);

end
%%------------------------------------------------------------------------%%
clear y IM1_out IM2_out IM3_out Sw_out
%%------------------------------------------------------------------------%%
%%                   Auxiliary Equations for the IMs
%%------------------------------------------------------------------------%%

% Transformation of electrical rotor speed and electrical angular position
% to mechanical rotor speed and mechanical angular position

% 1st IM
R_wm_1 =  wr_1/(ParametersIM.P/2);
R_theta_m_1 = theta_r_1/(ParametersIM.P/2);

% 2nd IM
R_wm_2 =  wr_2/(ParametersIM.P/2);
R_theta_m_2 = theta_r_2/(ParametersIM.P/2);

% 3rd IM
R_wm_3 =  wr_3/(ParametersIM.P/2);
R_theta_m_3 = theta_r_3/(ParametersIM.P/2);

% Transformation of fluxes to currents — DQ components
% 1ST MOTOR
[R_ids_1, R_iqs_1, R_idr_1, R_iqr_1]  = Fluxes2Currents(lambda_ds_1, ...
    lambda_qs_1, lambda_dr_1, lambda_qr_1, ParametersIM);
% 2ND MOTOR
[R_ids_2, R_iqs_2, R_idr_2, R_iqr_2]  = Fluxes2Currents(lambda_ds_2, ...
    lambda_qs_2, lambda_dr_2, lambda_qr_2, ParametersIM);
% 3RD MOTOR
[R_ids_3, R_iqs_3, R_idr_3, R_iqr_3]  = Fluxes2Currents(lambda_ds_3, ...
    lambda_qs_3, lambda_dr_3, lambda_qr_3, ParametersIM);

% Calculating voltage actually reaching motor 2
R_vqs_2 = R_vqs_2 — (R_iqs_2 .* rext2_pwm);
R_vds_2 = R_vds_2 — (R_ids_2 .* rext2_pwm);

% Calculating voltage actually reaching motor 3
R_vqs_3 = R_vqs_3 — (R_iqs_3 .* rext3_pwm);
R_vds_3 = R_vds_3 — (R_ids_3 .* rext3_pwm);

```

```
% Clear variables not saved
clear lambda_ds_1 lambda_qs_1 lambda_dr_1 lambda_qr_1 theta_r_1 wr_1 ...
    lambda_ds_2 lambda_qs_2 lambda_dr_2 lambda_qr_2 theta_r_2 wr_2 ...
    lambda_ds_3 lambda_qs_3 lambda_dr_3 lambda_qr_3 theta_r_3 wr_3 ...
    i j tspan t_ini t_end x0 num_states deltat
%--------------------------------------------------------------------------%
```

```
function [y, IM1, IM2, IM3, Sw] = ThreeIM_1Converter_IFOC_res(t, u, ...
    ParametersIM, ParamSys, ParamController)
%--------------------------------------------------------------------------%
%                           STATES
%--------------------------------------------------------------------------%
%                           1ST MOTOR
%--------------------------------------------------------------------------%
lambda_ds_1 = u(1);  % Flux  ds                                      [Wb]
lambda_qs_1 = u(2);  % Flux  qs                                      [Wb]
lambda_dr_1 = u(3);  % Flux  dr                                      [Wb]
lambda_qr_1 = u(4);  % Flux  qr                                      [Wb]
wr_1        = u(5);  % Rotor speed (electric)                     [rad/s]
theta_r_1   = u(6);  % Rotor position (electric)                    [rad]
%--------------------------------------------------------------------------%
%              INDIRECT FIELD ORIENTED CONTROL (1ST MOTOR)
%--------------------------------------------------------------------------%
theta_e   = u(7); % Rotor flux angular position (electric)     [rad]
I_Action_CL_w_1  = u(8); % Integral action of the closed loop control of wm
%--------------------------------------------------------------------------%
%--------------------------------------------------------------------------%
%                           2ND MOTOR
%--------------------------------------------------------------------------%
lambda_ds_2 = u(9);   % Flux  ds                                      [Wb]
lambda_qs_2 = u(10);  % Flux  qs                                      [Wb]
lambda_dr_2 = u(11);  % Flux  dr                                      [Wb]
lambda_qr_2 = u(12);  % Flux  qr                                      [Wb]
wr_2        = u(13);  % Rotor speed (electric)                     [rad/s]
theta_r_2   = u(14);  % Rotor position (electric)                    [rad]
%--------------------------------------------------------------------------%
%--------------------------------------------------------------------------%
%                           3RD MOTOR
%--------------------------------------------------------------------------%
lambda_ds_3 = u(15);  % Flux  ds                                      [Wb]
lambda_qs_3 = u(16);  % Flux  qs                                      [Wb]
lambda_dr_3 = u(17);  % Flux  dr                                      [Wb]
lambda_qr_3 = u(18);  % Flux  qr                                      [Wb]
wr_3        = u(19);  % Rotor speed (electric)                     [rad/s]
theta_r_3   = u(20);  % Rotor position (electric)                    [rad]
%--------------------------------------------------------------------------%
%--------------------------------------------------------------------------%
%                RESISTANCE CONTROL OF MOTORS 2&3
%--------------------------------------------------------------------------%
%--------------------------------------------------------------------------%
%                       CONTROLLERS
%--------------------------------------------------------------------------%
n = ParamController.order;
% Automatic state-building of the controllers, based on order
xk2 = u(21:21+n-1);
xk2 = reshape(xk2,n,1);
xk3 = u(21+n:21+2*n-1);
xk3 = reshape(xk3,n,1);
%--------------------------------------------------------------------------%

%--------------------------------------------------------------------------%
%                         GET SYSPARAMS
%--------------------------------------------------------------------------%

%--------------------------------------------------------------------------%
vswM = ParamSys.vswM;
vdM = ParamSys.vdM;
vswR = ParamSys.vswR;
vdR = ParamSys.vdR;

TL1_delta  = ParamSys.TL1_variation;
time_delta_TL1 = ParamSys.time_variation_TL1;

TL2_delta  = ParamSys.TL2_variation;
time_delta_TL2 = ParamSys.time_variation_TL2;

TL3_delta  = ParamSys.TL3_variation;
time_delta_TL3 = ParamSys.time_variation_TL3;

if t > time_delta_TL1
    TL_1 = (1+TL1_delta)*ParametersIM.Tnom;
else
```

```
    %     TL_1 = ParametersIM.Tnom;
    TL_1 = 0;
end

if t > time_delta_TL2
    TL_2 = (1+TL2_delta)*ParametersIM.Tnom;
else
%     TL_2 = ParametersIM.Tnom;
    TL_2 = 0;
end

if t > time_delta_TL3
    TL_3 = (1+TL3_delta)*ParametersIM.Tnom;
else
%     TL_3 = ParametersIM.Tnom;
    TL_3 = 0;
end
%------------------------------------------------------------------------%
%------------------------------------------------------------------------%
%                       GET CONTROLLER PARAMS
%------------------------------------------------------------------------%
% Hysteresis control
h = ParamController.h; % Hysteresis band

% r_ext control
Ak = ParamController.Ak;
Bk = ParamController.Bk;
Ck = ParamController.Ck;
Dk = ParamController.Dk;
%------------------------------------------------------------------------%
%                           Arbitrary Reference Frame
%------------------------------------------------------------------------%
%                            Stator reference
%------------------------------------------------------------------------%
% % 1st IM
% wa_1 = 0;
% theta_a_1 = 0;
%
% % 2nd IM
% wa_2 = 0;
% theta_a_2 = 0;
%
% % 3rd IM
% wa_3 = 0;
% theta_a_3 = 0;
%------------------------------------------------------------------------%

%------------------------------------------------------------------------%
%                    Auxiliary Equations for the IMs
%------------------------------------------------------------------------%
% Transformation of fluxes to currents — DQ components
% 1ST MOTOR
[ids_1, iqs_1, idr_1, iqr_1]  = Fluxes2Currents(lambda_ds_1, lambda_qs_1, ...
    lambda_dr_1, lambda_qr_1, ParametersIM);
% 2ND MOTOR
[ids_2, iqs_2, idr_2, iqr_2]  = Fluxes2Currents(lambda_ds_2, lambda_qs_2, ...
    lambda_dr_2, lambda_qr_2, ParametersIM);
% 3RD MOTOR
[ids_3, iqs_3, idr_3, iqr_3]  = Fluxes2Currents(lambda_ds_3, lambda_qs_3, ...
    lambda_dr_3, lambda_qr_3, ParametersIM);

% Electromagnetic and Load Torques
% 1ST MOTOR
Te_1 = 3/4*ParametersIM.P*(lambda_ds_1.*iqs_1 — lambda_qs_1.*ids_1);
% 2ND MOTOR
Te_2 = 3/4*ParametersIM.P*(lambda_ds_2.*iqs_2 — lambda_qs_2.*ids_2);
% 3RD MOTOR
Te_3 = 3/4*ParametersIM.P*(lambda_ds_3.*iqs_3 — lambda_qs_3.*ids_3);

% Transformation of electrical rotor speed and electrical angular position
% to mechanical rotor speed and mechanical angular position
% 1st IM
% wm_1 =  wr_1/(ParametersIM.P/2);
theta_m_1 = theta_r_1/(ParametersIM.P/2);
% 2nd IM
% wm_2 =  wr_2/(ParametersIM.P/2);
theta_m_2 = theta_r_2/(ParametersIM.P/2);
% 3rd IM
% wm_2 =  wr_2/(ParametersIM.P/2);
theta_m_3 = theta_r_3/(ParametersIM.P/2);

% Transforming stator and rotor currents from qd reference to abc variables
```

```
% % 1st IM
% [ias_1, ibs_1, ics_1] = qd2abcs(iqs_1, ids_1, theta_a_1);
% % 2nd IM
% [ias_2, ibs_2, ics_2] = qd2abcs(iqs_2, ids_2, theta_a_2);
% % 3rd IM
% [ias_3, ibs_3, ics_3] = qd2abcs(iqs_3, ids_3, theta_a_3);
% 1st IM
ias_1 = iqs_1;
[ibs_1, ics_1] = statbc(iqs_1, ids_1);
% 2nd IM
ias_2 = iqs_2;
[ibs_2, ics_2] = statbc(iqs_2, ids_2);
% 3rd IM
ias_3 = iqs_3;
[ibs_3, ics_3] = statbc(iqs_3, ids_3);

% Rotor Voltages (squirrel cage) —  DQ components
% 1st IM
vdr_1 = 0;
vqr_1 = 0;
% 2nd IM
vdr_2 = 0;
vqr_2 = 0;
% 3rd IM
vdr_3 = 0;
vqr_3 = 0;
%————————————————————————————————————————————————————————————————————————%

%————————————————————————————————————————————————————————————————————————%
%                      INDIRECT (ROTOR) FIELD ORIENTED CONTROL
%                           INDUCTION MOTOR 1
%————————————————————————————————————————————————————————————————————————%
[lambda_dr_star, wm_star] = RefLambdaAndwm(t, ParametersIM);

% Indirect field oriented control drive
[iqs_star, ids_star, di_wm_dt, wrf_est] = IFOCDrive(ParametersIM, ...
    lambda_dr_star, wm_star, wr_1, I_Action_CL_w_1);

% Converting currents to abc variables for Hysteresis current control
% Commanded currents
[ias_star, ibs_star, ics_star] = qd2abcs(iqs_star, ids_star, theta_e);
i_abcs_star = [ias_star, ibs_star, ics_star];

% Measured/real currents
i_abcs = [iqs_1, ibs_1, ics_1];

% Hysteresis current control
[S] = myHysteresisCurrentControl(i_abcs', i_abcs_star', h);

% 3phase bridge converter voltages
vdc = ParametersIM.Vs*sqrt(2); % peak of line—to—line voltage
IM = [(ias_1+ias_2+ias_3) (ibs_1+ibs_2+ibs_3) (ics_1+ics_2+ics_3)];
[vas, vbs, vcs] = my3PhaseConverterVoltLoss(S', vdc, vswM, vdM, IM);

% Converting vabcs voltages to vqds
[vqsC, vdsC] = statqd(vas, vbs, vcs);

%————————————————————————————————————————————————————————————————————————%

%————————————————————————————————————————————————————————————————————————%
%                       STATOR RESISTANCE CONTROL
%                           INDUCTION MOTOR 2
%————————————————————————————————————————————————————————————————————————%

%————————————————————————————————————————————————————————————————————————%
%                      CONTROLLER EQUATIONS
%————————————————————————————————————————————————————————————————————————%
% Controller input vector uk
% For now, only the position error

uk2(1) = theta_m_1 — theta_m_2;
xk2_dot = Ak*xk2 + Bk*uk2;
yk2 = Ck*xk2 + Dk*uk2;

uk3(1) = theta_m_1 — theta_m_3;
xk3_dot = Ak*xk3 + Bk*uk3;
yk3 = Ck*xk3 + Dk*uk3;

% Controller output vector yk
% For PI controller, yk is in units of 1/rad
rext2_cont = yk2(1)*ParametersIM.rs;
rext2_cont = max(rext2_cont,0);
```

```
rext2_cont = min(rext2_cont,ParamController.rext_max);

rext3_cont = yk3(1)*ParametersIM.rs;
rext3_cont = max(rext3_cont,0);
rext3_cont = min(rext3_cont,ParamController.rext_max);
%--------------------------------------------------------------------------%

% Controlable external resistance via PWM
f_sw = 4.987654321e3;
r_full = ParamController.rext_max;
d2 = rext2_cont/r_full;
S_rext2 = myPWM(t, d2, f_sw);
rext2_pwm = S_rext2 * r_full;

d3 = rext3_cont/r_full;
S_rext3 = myPWM(t, d3, f_sw);
rext3_pwm = S_rext3 * r_full;

% PWM external resistance
rext2 = rext2_pwm;
rext3 = rext3_pwm;

% Ideal external resistance
% rext2 = rext2_cont;
% rext3 = rext3_cont;

% vqs_2 = vqs_1;
% vds_2 = vds_1;
% If external resistance is off, then current flows through switches. If
% these are non-ideal, then voltage drop is always 1 switch + 1 diode
vas_2 = vas - ~S_rext2*sign(ias_2)*(vswR+vdR);
vbs_2 = vbs - ~S_rext2*sign(ibs_2)*(vswR+vdR);
vcs_2 = vcs - ~S_rext2*sign(ics_2)*(vswR+vdR);
% Transforming vabcs to vds and vqs
[vqs_2, vds_2] = statqd(vas_2, vbs_2, vcs_2);

vas_3 = vas - ~S_rext3*sign(ias_3)*(vswR+vdR);
vbs_3 = vbs - ~S_rext3*sign(ibs_3)*(vswR+vdR);
vcs_3 = vcs - ~S_rext3*sign(ics_3)*(vswR+vdR);
% Transforming vabcs to vds and vqs
[vqs_3, vds_3] = statqd(vas_3, vbs_3, vcs_3);

% Apply this loss to motor line 1 too for parity. In a real system, both
% would be connected to a board in case torque differnce reverses
vas_1 = vas - sign(ias_1)*(vswR+vdR);
vbs_1 = vbs - sign(ibs_1)*(vswR+vdR);
vcs_1 = vcs - sign(ics_1)*(vswR+vdR);
% Transforming vabcs to vds and vqs
[vqs_1, vds_1] = statqd(vas_1, vbs_1, vcs_1);
%--------------------------------------------------------------------------%

%--------------------------------------------------------------------------%
%                       OUTPUT STRUCTURES
%--------------------------------------------------------------------------%
% Switching states
Sw.S = S;
Sw.S_rext2 = S_rext2;
Sw.S_rext3 = S_rext3;

% 1st motor
IM1.vqsC = vqsC;
IM1.vdsC = vdsC;
IM1.TL = TL_1;
IM1.Te = Te_1;
IM1.vqs = vqs_1;
IM1.vds = vds_1;
IM1.theta_e = theta_e;
% IM1.we = we;
% IM1.theta_e = theta_e;
% IM1.theta_a = theta_a_1;
IM1.wm_star = wm_star;

% 2nd motor
IM2.TL = TL_2;
IM2.Te = Te_2;
IM2.vqs = vqs_2;
IM2.vds = vds_2;
% IM2.theta_a = theta_a_2;
IM2.rext_pwm = rext2_pwm;
IM2.rext_cont = rext2_cont;

% 3rd motor
```

```
IM3.TL = TL_3;
IM3.Te = Te_3;
IM3.vqs = vqs_3;
IM3.vds = vds_3;
% IM3.theta_a = theta_a_3;
IM3.rext_pwm = rext3_pwm;
IM3.rext_cont = rext3_cont;

%--------------------------------------------------------------------------%


%--------------------------------------------------------------------------%
%                     DIFF. STATES EQUATIONS
%--------------------------------------------------------------------------%
%                           1ST MOTOR
%--------------------------------------------------------------------------%
% y(1) = vds_1 - ParametersIM.rs*ids_1 + wa_1 * lambda_qs_1;       % lambda_ds_1
% y(2) = vqs_1 - ParametersIM.rs*iqs_1 - wa_1 * lambda_ds_1;       % lambda_qs_1
% y(3) = vdr_1 - ParametersIM.rr*idr_1 + (wa_1-wr_1)*lambda_qr_1;  % lambda_dr_1
% y(4) = vqr_1 - ParametersIM.rr*iqr_1 - (wa_1-wr_1)*lambda_dr_1;  % lambda_qr_1
y(1) = vds_1 - ParametersIM.rs*ids_1;                        % lambda_ds_1
y(2) = vqs_1 - ParametersIM.rs*iqs_1;                        % lambda_qs_1
y(3) = vdr_1 - ParametersIM.rr*idr_1 - wr_1*lambda_qr_1;     % lambda_dr_1
y(4) = vqr_1 - ParametersIM.rr*iqr_1 + wr_1*lambda_dr_1;     % lambda_qr_1
y(5) = (1/ParametersIM.J)*((Te_1-TL_1)*ParametersIM.P/2 + ...
                           -ParametersIM.b*wr_1);            % wr_1
y(6) = wr_1;                                                 % theta_r_1
%--------------------------------------------------------------------------%
%              INDIRECT FIELD ORIENTED CONTROL (1ST MOTOR)
%--------------------------------------------------------------------------%
y(7) = wrf_est;   % Rotor flux angular position (electric)
y(8) = di_wm_dt;  % Integral action of the closed loop control of wm
%--------------------------------------------------------------------------%
%--------------------------------------------------------------------------%
%                           2ND MOTOR
%--------------------------------------------------------------------------%
% y(9) = vds_2 - (ParametersIM.rs + rext2)*ids_2 + wa_2 * lambda_qs_2;      % lambda_ds_2
% y(10) = vqs_2 - (ParametersIM.rs + rext2)*iqs_2 - wa_2 * lambda_ds_2;     % lambda_qs_2
% y(11) = vdr_2 - ParametersIM.rr*idr_2 + (wa_2-wr_2)*lambda_qr_2; % lambda_dr_2
% y(12) = vqr_2 - ParametersIM.rr*iqr_2 - (wa_2-wr_2)*lambda_dr_2; % lambda_qr_2
y(9) = vds_2 - (ParametersIM.rs + rext2)*ids_2;              % lambda_ds_2
y(10) = vqs_2 - (ParametersIM.rs + rext2)*iqs_2;             % lambda_qs_2
y(11) = vdr_2 - ParametersIM.rr*idr_2 - wr_2*lambda_qr_2;    % lambda_dr_2
y(12) = vqr_2 - ParametersIM.rr*iqr_2 + wr_2*lambda_dr_2;    % lambda_qr_2
y(13) = (1/ParametersIM.J)*((Te_2-TL_2)*ParametersIM.P/2 + ...
                           -ParametersIM.b*wr_2);            % wr_2
y(14) = wr_2;                                                % theta_r_2
%--------------------------------------------------------------------------%
%                           3RD MOTOR
%--------------------------------------------------------------------------%
% y(15) = vds_3 - (ParametersIM.rs + rext3)*ids_3 + wa_3 * lambda_qs_3;     % lambda_ds_3
% y(16) = vqs_3 - (ParametersIM.rs + rext3)*iqs_3 - wa_3 * lambda_ds_3;     % lambda_qs_3
% y(17) = vdr_3 - ParametersIM.rr*idr_3 + (wa_3-wr_3)*lambda_qr_3; % lambda_dr_3
% y(18) = vqr_3 - ParametersIM.rr*iqr_3 - (wa_3-wr_3)*lambda_dr_3; % lambda_qr_3
y(15) = vds_3 - (ParametersIM.rs + rext3)*ids_3;             % lambda_ds_3
y(16) = vqs_3 - (ParametersIM.rs + rext3)*iqs_3;             % lambda_qs_3
y(17) = vdr_3 - ParametersIM.rr*idr_3 - wr_3*lambda_qr_3;    % lambda_dr_3
y(18) = vqr_3 - ParametersIM.rr*iqr_3 + wr_3*lambda_dr_3;    % lambda_qr_3
y(19) = (1/ParametersIM.J)*((Te_3-TL_3)*ParametersIM.P/2 + ...
                           -ParametersIM.b*wr_3);            % wr_3
y(20) = wr_3;                                                % theta_r_3
%--------------------------------------------------------------------------%
%                RESISTANCE CONTROL OF 2ND MOTOR
%--------------------------------------------------------------------------%
%--------------------------------------------------------------------------%
%                       CONTROLLER
%--------------------------------------------------------------------------%
% Automatic state-building diff. equations of the controller, based on its order
y(21:21+n-1) = xk2_dot;
y(21+n:21+2*n-1) = xk3_dot;
%--------------------------------------------------------------------------%

y = y';
end
```

B.4 IDFOC Control, Auxiliary Converter Method

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Multirun_3IM_IFOC_volt_xf.m
```

```
% Runs 3-motor simulation of auxiliary converter method using CVHz control
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear
close all
clc

% load k_hinf
% [Ak,Bk,Ck,Dk] = ssdata(k_hinf);

Kp = -80; % V/rad
Ki = -120; % V/(rad*s)
[Ak,Bk,Ck,Dk] = ssdata(tf([Kp Ki],[1 0]));

% Code currently only supports 3 identical motors
ParametersIM = ReturnParametersIM_15hp;

rs = ParametersIM.rs;
rr = ParametersIM.rr;
Lls = ParametersIM.Lls;
Llr = ParametersIM.Llr;
Lm = ParametersIM.Lm;
Ls = ParametersIM.Ls;
Lr = ParametersIM.Lr;
% sigma = 1 - Lm^2/(Ls*Lr);
% tau_s = Ls/rs;
% tau_r = Lr/rr;
% sigma_s = tau_s*sigma;
% sigma_r = tau_r*sigma;
% mu_s = 1/sigma_s;
% mu_r = 1/sigma_r;

% Incorporating mu_s inside the param structure
% ParametersIM.mu_s = mu_s;

ParamController.h = 0.1; % Hysteresis band

ParamController.Ak = Ak;
ParamController.Bk = Bk;
ParamController.Ck = Ck;
ParamController.Dk = Dk;
ParamController.order = length(Ak);

ParamController.delv_max = 50;
ParamController.tstart = 2; % Time to turn on voltage drop converter

% Parameters for switching losses
ParamSys.vswM = 0; % Main converter switch voltage
ParamSys.vdM = 0;  % Main converter diode voltage
ParamSys.vswD = 0; % Drop converter switch voltage
ParamSys.vdD = 0;  % Drop converter diode voltage

time_variation_TL1 = 2;
time_variation_TL2 = 2;
time_variation_TL3 = 2;

ParamSys.time_variation_TL1 = time_variation_TL1;
ParamSys.time_variation_TL2 = time_variation_TL2;
ParamSys.time_variation_TL3 = time_variation_TL3;

TL1_variation = 0; % Variation in load torque in pu
ParamSys.TL1_variation = TL1_variation;

TL2_variation = -0.2;
ParamSys.TL2_variation = TL2_variation;

TL3_variation = -0.3;
ParamSys.TL3_variation = TL3_variation;

num_states = 26 + 2*ParamController.order;
x0 = zeros(1,num_states); % Initial conditions for state variables
% load x0_save
% x0 = x0_save;

t_ini = 0;
t_end = 6;

deltat = 10e-6;
tspan = t_ini:deltat:t_end;
options = odeset('MaxStep', deltat, 'RelTol',1e-5, 'AbsTol', 1e-6);
% options = odeset();

% Set up iterations and allocate space for saved variables
```

```
Nlist = [1/5 1/3 1/2 1/2];
vdcRlist = [1 1 1 0.5];

TL_2_out = zeros(numel(tspan),numel(Nlist));
Te_2_out = TL_2_out;
delv2_vd_out = TL_2_out;
delv2_vq_out = TL_2_out;
delv2_cont_out = TL_2_out;
wm_2_out = TL_2_out;
theta_m_2_out = TL_2_out;
ids_2_out = TL_2_out;
iqs_2_out = TL_2_out;
idr_2_out = TL_2_out;
iqr_2_out = TL_2_out;
id2_2_out = TL_2_out;
iq2_2_out = TL_2_out;
vq1_2_out = TL_2_out;
vd1_2_out = TL_2_out;
vqs_2_out = TL_2_out;
vds_2_out = TL_2_out;
Sd1_2_out = TL_2_out;
Sd2_2_out = TL_2_out;
Sd3_2_out = TL_2_out;

TL_3_out = TL_2_out;
Te_3_out = TL_2_out;
delv3_vd_out = TL_2_out;
delv3_vq_out = TL_2_out;
delv3_cont_out = TL_2_out;
wm_3_out = TL_2_out;
theta_m_3_out = TL_2_out;
ids_3_out = TL_2_out;
iqs_3_out = TL_2_out;
idr_3_out = TL_2_out;
iqr_3_out = TL_2_out;
id2_3_out = TL_2_out;
iq2_3_out = TL_2_out;
vq1_3_out = TL_2_out;
vd1_3_out = TL_2_out;
vqs_3_out = TL_2_out;
vds_3_out = TL_2_out;
Sd1_3_out = TL_2_out;
Sd2_3_out = TL_2_out;
Sd3_3_out = TL_2_out;

TL_1_out = TL_2_out;
Te_1_out = TL_1_out;
wm_1_out = TL_1_out;
theta_m_1_out = TL_1_out;
ids_1_out = TL_1_out;
iqs_1_out = TL_1_out;
idr_1_out = TL_1_out;
iqr_1_out = TL_1_out;
id2_1_out = TL_1_out;
iq2_1_out = TL_1_out;
vq1_1_out = TL_1_out;
vd1_1_out = TL_1_out;
vqs_1_out = TL_1_out;
vds_1_out = TL_1_out;
S1_out = TL_1_out;
S2_out = TL_1_out;
S3_out = TL_1_out;

theta_e_out = TL_1_out;

% Iterate over transformer turns ratios & auxiliary converter bus voltages
for j=1:numel(Nlist)

% Transformer params: line side = primary (1), aux converter = secondary (2)
N = Nlist(j);              % N1/N2
ParametersXF.N = N;        % Z' = Z*N^2, but less windings = less impedance
ParametersXF.r1 = 0.001;  % So actual secondary impedance scales with N2,
ParametersXF.r2 = 0.01*N; % canceling a power of N for this scaling
ParametersXF.Ll1 = 0.1e-3;
ParametersXF.Ll2 = 0.5e-3*N;
ParametersXF.LmT = 10e-3;
ParametersXF.vdcR = vdcRlist(j); % Ratio between aux converter bus and main bus

% Calculate inductance matrix inverses in setup
% The factor of 3/2 for magnetizing inductance (Lm -> M) is assumed included
Ll1 = ParametersXF.Ll1;
Ll2 = ParametersXF.Ll2;
```

```
LmT = ParametersXF.LmT;
L = [[Lls+Ll1+Lm+LmT, Lm, LmT];...
    [Lm, Llr+Lm, 0];...
    [LmT, 0, Ll2+LmT]];
ParametersXF.Linv = inv(L);

tic
[t,y] = ode45(@(t,y) ThreeIM_1Converter_IFOC_volt_xf(t,y,ParametersIM, ...
    ParametersXF, ParamSys, ParamController), tspan, x0, options);
toc

% Save state
% x0_save = y(end,:);
% save x0_save x0_save

%-------------------------------------------------------------------------%
%                          POST-PROCESSING
%-------------------------------------------------------------------------%

%-------------------------------------------------------------------------%
%                          PICKING UP THE STATES
%-------------------------------------------------------------------------%
%                          1ST MOTOR
%-------------------------------------------------------------------------%
lambda_dC_1   =   y(:,1);  % Flux dC (main converter path)            [Wb]
lambda_qC_1   =   y(:,2);  % Flux qC                                  [Wb]
lambda_dr_1   =   y(:,3);  % Flux dr (rotor path)                     [Wb]
lambda_qr_1   =   y(:,4);  % Flux qr                                  [Wb]
lambda_d2_1   =   y(:,5);  % Flux d2 (side converter path)            [Wb]
lambda_q2_1   =   y(:,6);  % Flux q2                                  [Wb]
wr_1          =   y(:,7);  % Rotor speed (electric)                [rad/s]
theta_r_1     =   y(:,8);  % Rotor position (electric)               [rad]
%-------------------------------------------------------------------------%
%              INDIRECT FIELD ORIENTED CONTROL (1ST MOTOR)
%-------------------------------------------------------------------------%
theta_e  = y(:,9); % Rotor flux angular position (electric)    [rad]
I_Action_CL_w_1 = y(:,10); % Integral action of the closed loop control of wm
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
%                            2ND MOTOR
%-------------------------------------------------------------------------%
lambda_dC_2   =   y(:,11);  % Flux dC (main converter path)           [Wb]
lambda_qC_2   =   y(:,12);  % Flux qC                                 [Wb]
lambda_dr_2   =   y(:,13);  % Flux dr (rotor path)                    [Wb]
lambda_qr_2   =   y(:,14);  % Flux qr                                 [Wb]
lambda_d2_2   =   y(:,15);  % Flux d2 (side converter path)           [Wb]
lambda_q2_2   =   y(:,16);  % Flux q2                                 [Wb]
wr_2          =   y(:,17);  % Rotor speed (electric)               [rad/s]
theta_r_2     =   y(:,18);  % Rotor position (electric)              [rad]
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
%                            3RD MOTOR
%-------------------------------------------------------------------------%
lambda_dC_3   =   y(:,19);  % Flux dC (main converter path)           [Wb]
lambda_qC_3   =   y(:,20);  % Flux qC                                 [Wb]
lambda_dr_3   =   y(:,21);  % Flux dr (rotor path)                    [Wb]
lambda_qr_3   =   y(:,22);  % Flux qr                                 [Wb]
lambda_d2_3   =   y(:,23);  % Flux d2 (side converter path)           [Wb]
lambda_q2_3   =   y(:,24);  % Flux q2                                 [Wb]
wr_3          =   y(:,25);  % Rotor speed (electric)               [rad/s]
theta_r_3     =   y(:,26);  % Rotor position (electric)              [rad]
%-------------------------------------------------------------------------%

%-------------------------------------------------------------------------%
%          EXTRACTING OUTPUT STRUCTURES FROM DIFF EQ FUNCTION
%-------------------------------------------------------------------------%
[~, IM1_out, IM2_out, IM3_out, Sw_out] = cellfun(@(t,y) ...
    ThreeIM_1Converter_IFOC_volt_xf(t, y, ParametersIM, ParametersXF, ...
    ParamSys, ParamController), num2cell(t), num2cell(y,2),'uni',0);

TL_1 = zeros(numel(t),1);
Te_1 = zeros(numel(t),1);
% wa_1 = zeros(numel(t),1);
% theta_a_1 = zeros(numel(t),1);
vqC = zeros(numel(t),1);
vdC = zeros(numel(t),1);

TL_2 = zeros(numel(t),1);
Te_2 = zeros(numel(t),1);
% wa_2 = zeros(numel(t),1);
% theta_a_2 = zeros(numel(t),1);

```

```
delv2_vq = zeros(numel(t),1);
delv2_vd = zeros(numel(t),1);
delv2_cont = zeros(numel(t),1);

TL_3 = zeros(numel(t),1);
Te_3 = zeros(numel(t),1);
% wa_3 = zeros(numel(t),1);
% theta_a_3 = zeros(numel(t),1);

delv3_vq = zeros(numel(t),1);
delv3_vd = zeros(numel(t),1);
delv3_cont = zeros(numel(t),1);

S1 = zeros(numel(t),1);
S2 = zeros(numel(t),1);
S3 = zeros(numel(t),1);

Sd1_2 = zeros(numel(t),1);
Sd2_2 = zeros(numel(t),1);
Sd3_2 = zeros(numel(t),1);

Sd1_3 = zeros(numel(t),1);
Sd2_3 = zeros(numel(t),1);
Sd3_3 = zeros(numel(t),1);

for i=1:numel(t)
    TL_1(i) = IM1_out{i}.TL;
    Te_1(i) = IM1_out{i}.Te;
%     wa_1(i) = IM1_out{i}.wa;
%     theta_a_1(i) = IM1_out{i}.theta_a;

    vqC(i) = IM1_out{i}.vqC;
    vdC(i) = IM1_out{i}.vdC;

    TL_2(i) = IM2_out{i}.TL;
    Te_2(i) = IM2_out{i}.Te;
%     wa_2(i) = IM2_out{i}.wa;
%     theta_a_2(i) = IM2_out{i}.theta_a;

    delv2_vq(i) = IM2_out{i}.delv2_vq;
    delv2_vd(i) = IM2_out{i}.delv2_vd;
    delv2_cont(i) = IM2_out{i}.delv2_cont;

    TL_3(i) = IM3_out{i}.TL;
    Te_3(i) = IM3_out{i}.Te;
%     wa_3(i) = IM2_out{i}.wa;
%     theta_a_3(i) = IM2_out{i}.theta_a;

    delv3_vq(i) = IM3_out{i}.delv3_vq;
    delv3_vd(i) = IM3_out{i}.delv3_vd;
    delv3_cont(i) = IM3_out{i}.delv3_cont;

    S1(i) = Sw_out{i}.S(1);
    S2(i) = Sw_out{i}.S(2);
    S3(i) = Sw_out{i}.S(3);

    Sd1_2(i) = Sw_out{i}.S_d2(1);
    Sd2_2(i) = Sw_out{i}.S_d2(2);
    Sd3_2(i) = Sw_out{i}.S_d2(3);

    Sd1_3(i) = Sw_out{i}.S_d3(1);
    Sd2_3(i) = Sw_out{i}.S_d3(2);
    Sd3_3(i) = Sw_out{i}.S_d3(3);

end

%--------------------------------------------------------------------------%
clear y IM1_out IM2_out IM3_out Sw_out
%--------------------------------------------------------------------------%
%                    Auxiliary Equations for the IMs
%--------------------------------------------------------------------------%

% Transformation of electrical rotor speed and electrical angular position
% to mechanical rotor speed and mechanical angular position

% 1st IM
wm_1 =  wr_1/(ParametersIM.P/2);
theta_m_1 = theta_r_1/(ParametersIM.P/2);
% 2nd IM
wm_2 =  wr_2/(ParametersIM.P/2);
theta_m_2 = theta_r_2/(ParametersIM.P/2);
% 3rd IM
```

```
wm_3 =  wr_3/(ParametersIM.P/2);
theta_m_3 = theta_r_3/(ParametersIM.P/2);

% Transformation of fluxes to currents - DQ components
Linv = ParametersXF.Linv;
% 1ST MOTOR
[ids_1, iqs_1, idr_1, iqr_1, id2_1, iq2_1] = Fluxes2Currents_xf(lambda_dC_1, ...
    lambda_qC_1, lambda_dr_1, lambda_qr_1, lambda_d2_1, lambda_q2_1, Linv);
% 2ND MOTOR
[ids_2, iqs_2, idr_2, iqr_2, id2_2, iq2_2] = Fluxes2Currents_xf(lambda_dC_2, ...
    lambda_qC_2, lambda_dr_2, lambda_qr_2, lambda_d2_2, lambda_q2_2, Linv);
% 3RD MOTOR
[ids_3, iqs_3, idr_3, iqr_3, id2_3, iq2_3] = Fluxes2Currents_xf(lambda_dC_3, ...
    lambda_qC_3, lambda_dr_3, lambda_qr_3, lambda_d2_3, lambda_q2_3, Linv);

% Transforming stator and rotor currents from qd reference to abc variables
% Disabled to reduce size of saved data - can convert later
% % 1st IM
% % [ias_1, ibs_1, ics_1] = qd2abcs(iqs_1, ids_1, theta_a_1);
% ias_1 = iqs_1;
% [ibs_1, ics_1] = statbc(iqs_1, ids_1);
% % [iar_1, ~, ~] = qd2abcs(iqr_1, idr_1, theta_a_1-theta_r_1);
% % [ia2_1, ib2_1, ic2_1] = qd2abcs(iq2_1, id2_1, theta_a_1);
% ia2_1 = iq2_1;
% [ib2_1, ic2_1] = statbc(iq2_1, id2_1);
% % 2nd IM
% % [ias_2, ibs_2, ics_2] = qd2abcs(iqs_2, ids_2, theta_a_2);
% ias_2 = iqs_2;
% [ibs_2, ics_2] = statbc(iqs_2, ids_2);
% % [iar_2, ~, ~] = qd2abcs(iqr_2, idr_2, theta_a_2-theta_r_2);
% % [ia2_2, ib2_2, ic2_2] = qd2abcs(iq2_2, id2_2, theta_a_2);
% ia2_2 = iq2_2;
% [ib2_2, ic2_2] = statbc(iq2_2, id2_2);
%
% % Transforming stator voltages from qd reference to abc variables
% % Main converter
% % [vaC, vbC, vcC] = qd2abcs(vqC, vdC, theta_a_1);
% vaC = vqC;
% [vbC, vcC] = statbc(vqC, vdC);
% % Voltage drop converter
% % [delv2_va, delv2_vb, delv2_vc] = qd2abcs(delv2_vq, delv2_vd, theta_a_2);
% delv2_va = delv2_vq;
% [delv2_vb, delv2_vc] = statbc(delv2_vq, delv2_vd);

% Voltage reaching motors: Need to calculate based on currents. v=Ldi/dt
% 1st IM
Dids_1 = [0; diff(ids_1)/deltat];
Diqs_1 = [0; diff(iqs_1)/deltat];
Did2_1 = [0; diff(id2_1)/deltat];
Diq2_1 = [0; diff(iq2_1)/deltat];

% lambda_d1_1 = (Ll1 + LmT)*ids_1 + LmT*id2_1;
% lambda_q1_1 = (Ll1 + LmT)*iqs_1 + LmT*iq2_1;

% vd1_1 = ParametersXF.r1*ids_1 + Ll1*Dids_1 + LmT*(Dids_1+Did2_1) - wa_1.*lambda_q1_1;
vd1_1 = ParametersXF.r1*ids_1 + Ll1*Dids_1 + LmT*(Dids_1+Did2_1);
vds_1 = vdC - vd1_1;
% vq1_1 = ParametersXF.r1*iqs_1 + Ll1*Diqs_1 + LmT*(Diqs_1+Diq2_1) + wa_1.*lambda_d1_1;
vq1_1 = ParametersXF.r1*iqs_1 + Ll1*Diqs_1 + LmT*(Diqs_1+Diq2_1);
vqs_1 = vqC - vq1_1;

% % [va1_1, vb1_1, vc1_1] = qd2abcs(vq1_1, vd1_1, theta_a_1);
% va1_1 = vq1_1;
% [vb1_1, vc1_1] = statbc(vq1_1, vd1_1);
% % [vas_1, vbs_1, vcs_1] = qd2abcs(vqs_1, vds_1, theta_a_1);
% vas_1 = vqs_1;
% [vbs_1, vcs_1] = statbc(vqs_1, vds_1);

% 2nd IM
Dids_2 = [0; diff(ids_2)/deltat];
Diqs_2 = [0; diff(iqs_2)/deltat];
Did2_2 = [0; diff(id2_2)/deltat];
Diq2_2 = [0; diff(iq2_2)/deltat];

% lambda_d1_2 = (Ll1 + LmT)*ids_2 + LmT*id2_2;
% lambda_q1_2 = (Ll1 + LmT)*iqs_2 + LmT*iq2_2;

% vd1_2 = ParametersXF.r1*ids_2 + Ll1*Dids_2 + LmT*(Dids_2+Did2_2) - wa_2.*lambda_q1_2;
vd1_2 = ParametersXF.r1*ids_2 + Ll1*Dids_2 + LmT*(Dids_2+Did2_2);
vds_2 = vdC - vd1_2;
% vq1_2 = ParametersXF.r1*iqs_2 + Ll1*Diqs_2 + LmT*(Diqs_2+Diq2_2) + wa_2.*lambda_d1_2;
vq1_2 = ParametersXF.r1*iqs_2 + Ll1*Diqs_2 + LmT*(Diqs_2+Diq2_2);
```

```
vqs_2 = vqC - vq1_2;

% % [va1_2, vb1_2, vc1_2] = qd2abcs(vq1_2, vd1_2, theta_a_2);
% va1_2 = vq1_2;
% [vb1_2, vc1_2] = statbc(vq1_2, vd1_2);
% % [vas_2, vbs_2, vcs_2] = qd2abcs(vqs_2, vds_2, theta_a_2);
% vas_2 = vqs_2;
% [vbs_2, vcs_2] = statbc(vqs_2, vds_2);

% 3rd IM
Dids_3 = [0; diff(ids_3)/deltat];
Diqs_3 = [0; diff(iqs_3)/deltat];
Did2_3 = [0; diff(id2_3)/deltat];
Diq2_3 = [0; diff(iq2_3)/deltat];

vd1_3 = ParametersXF.r1*ids_3 + Ll1*Dids_3 + LmT*(Dids_3+Did2_3);
vds_3 = vdC - vd1_3;
vq1_3 = ParametersXF.r1*iqs_3 + Ll1*Diqs_3 + LmT*(Diqs_3+Diq2_3);
vqs_3 = vqC - vq1_3;
%--------------------------------------------------------------------------%

% Clear variables not saved
clear lambda_dC_1 lambda_qC_1 lambda_dr_1 lambda_qr_1 lambda_d2_1 ...
    lambda_q2_1 lambda_dC_2 lambda_qC_2 lambda_dr_2 lambda_qr_2 ...
    lambda_d2_2 lambda_q2_2 lambda_dC_3 lambda_qC_3 lambda_dr_3 ...
    lambda_qr_3 lambda_d2_3 lambda_q2_3 theta_r_1 theta_r_2 theta_r_3 ...
    wr_1 wr_2 wr_3 Dids_1 Diqs_1 Did2_1 Diq2_1 Dids_2 Diqs_2 Did2_2 ...
    Diq2_2 Dids_3 Diqs_3 Did2_3 Diq2_3

% Save variables before next loop (motor 2, different xfmr params)
TL_2_out(:,j) = TL_2;
Te_2_out(:,j) = Te_2;
delv2_vd_out(:,j) = delv2_vd;
delv2_vq_out(:,j) = delv2_vq;
delv2_cont_out(:,j) = delv2_cont;
wm_2_out(:,j) = wm_2;
theta_m_2_out(:,j) = theta_m_2;
ids_2_out(:,j) = ids_2;
iqs_2_out(:,j) = iqs_2;
idr_2_out(:,j) = idr_2;
iqr_2_out(:,j) = iqr_2;
id2_2_out(:,j) = id2_2;
iq2_2_out(:,j) = iq2_2;
vq1_2_out(:,j) = vq1_2;
vd1_2_out(:,j) = vd1_2;
vqs_2_out(:,j) = vqs_2;
vds_2_out(:,j) = vds_2;
Sd1_2_out(:,j) = Sd1_2;
Sd2_2_out(:,j) = Sd2_2;
Sd3_2_out(:,j) = Sd3_2;

% Save variables before next loop (motor 3, different xfmr params)
TL_3_out(:,j) = TL_3;
Te_3_out(:,j) = Te_3;
delv3_vd_out(:,j) = delv3_vd;
delv3_vq_out(:,j) = delv3_vq;
delv3_cont_out(:,j) = delv3_cont;
wm_3_out(:,j) = wm_3;
theta_m_3_out(:,j) = theta_m_3;
ids_3_out(:,j) = ids_3;
iqs_3_out(:,j) = iqs_3;
idr_3_out(:,j) = idr_3;
iqr_3_out(:,j) = iqr_3;
id2_3_out(:,j) = id2_3;
iq2_3_out(:,j) = iq2_3;
vq1_3_out(:,j) = vq1_3;
vd1_3_out(:,j) = vd1_3;
vqs_3_out(:,j) = vqs_3;
vds_3_out(:,j) = vds_3;
Sd1_3_out(:,j) = Sd1_3;
Sd2_3_out(:,j) = Sd2_3;
Sd3_3_out(:,j) = Sd3_3;

% Save variables before next loop (motor 1, different xfmr params)
TL_1_out(:,j) = TL_1;
Te_1_out(:,j) = Te_1;
wm_1_out(:,j) = wm_1;
theta_m_1_out(:,j) = theta_m_1;
ids_1_out(:,j) = ids_1;
iqs_1_out(:,j) = iqs_1;
idr_1_out(:,j) = idr_1;
iqr_1_out(:,j) = iqr_1;
```

```
id2_1_out(:,j) = id2_1;
iq2_1_out(:,j) = iq2_1;
vq1_1_out(:,j) = vq1_1;
vd1_1_out(:,j) = vd1_1;
vqs_1_out(:,j) = vqs_1;
vds_1_out(:,j) = vds_1;
S1_out(:,j) = S1;
S2_out(:,j) = S2;
S3_out(:,j) = S3;

theta_e_out(:,j) = theta_e;

end

% Clear redundant variables left over
clear delv2_cont delv2_vd delv2_vq delv3_cont delv3_vd delv3_vq i id2_1 ...
    id2_2 id2_3 idr_1 idr_2 idr_3 ids_1 ids_2 ids_3 iq2_1 iq2_2 iq2_3 ...
    iqr_1 iqr_2 iqr_3 iqs_1 iqs_2 iqs_3 j k N num_states S1 S2 S3 ...
    Sd1_2 Sd2_2 Sd3_2 Sd1_3 Sd2_3 Sd3_3 t_end t_ini tspan Te_1 Te_2 Te_3 ...
    theta_m_1 theta_m_2 theta_m_3 TL_1 TL_2 TL_3 vd1_1 vd1_2 vd1_3 ...
    vdC vds_1 vds_2 vds_3 vq1_1 vq1_2 vq1_3 vqC vqs_1 vqs_2 vqs_3 ...
    wm_1 wm_2 wm_3 x0 deltat theta_e
%-------------------------------------------------------------------------%
```

```
function [y, IM1, IM2, IM3, Sw] = ThreeIM_1Converter_IFOC_volt_xf(t, ...
    u, ParametersIM, ParametersXF, ParamSys, ParamController)
%-------------------------------------------------------------------------%
%                           STATES
%-------------------------------------------------------------------------%
%                           1ST MOTOR
%-------------------------------------------------------------------------%
lambda_dC_1 = u(1);  % Flux  dC (ds + d1)                                [Wb]
lambda_qC_1 = u(2);  % Flux  qC (qs + q1)                                [Wb]
lambda_dr_1 = u(3);  % Flux  dr                                          [Wb]
lambda_qr_1 = u(4);  % Flux  qr                                          [Wb]
lambda_d2_1 = u(5);  % Flux  d2 (xfmr side converter)                    [Wb]
lambda_q2_1 = u(6);  % Flux  q2                                          [Wb]
wr_1        = u(7);  % Rotor speed (electric)                         [rad/s]
theta_r_1   = u(8);  % Rotor position (electric)                        [rad]
%-------------------------------------------------------------------------%
%              INDIRECT FIELD ORIENTED CONTROL (1ST MOTOR)
%-------------------------------------------------------------------------%
theta_e   = u(9); % Rotor flux angular position (electric)     [rad]
I_Action_CL_w_1  = u(10); % Integral action of the closed loop control of wm
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
%                           2ND MOTOR
%-------------------------------------------------------------------------%
lambda_dC_2 = u(11);  % Flux  dC (ds + d1)                               [Wb]
lambda_qC_2 = u(12);  % Flux  qC (qs + q1)                               [Wb]
lambda_dr_2 = u(13);  % Flux  dr                                         [Wb]
lambda_qr_2 = u(14);  % Flux  qr                                         [Wb]
lambda_d2_2 = u(15);  % Flux  d2 (xfmr side converter)                   [Wb]
lambda_q2_2 = u(16);  % Flux  q2                                         [Wb]
wr_2        = u(17);  % Rotor speed (electric)                        [rad/s]
theta_r_2   = u(18);  % Rotor position (electric)                       [rad]
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
%                           3RD MOTOR
%-------------------------------------------------------------------------%
lambda_dC_3 = u(19);  % Flux  dC (ds + d1)                               [Wb]
lambda_qC_3 = u(20);  % Flux  qC (qs + q1)                               [Wb]
lambda_dr_3 = u(21);  % Flux  dr                                         [Wb]
lambda_qr_3 = u(22);  % Flux  qr                                         [Wb]
lambda_d2_3 = u(23);  % Flux  d2 (xfmr side converter)                   [Wb]
lambda_q2_3 = u(24);  % Flux  q2                                         [Wb]
wr_3        = u(25);  % Rotor speed (electric)                        [rad/s]
theta_r_3   = u(26);  % Rotor position (electric)                       [rad]
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
%                       CONTROLLER
%-------------------------------------------------------------------------%
n = ParamController.order;
% Automatic state-building of the controller, based on its order
xk2 = u(27:27+n-1);
xk2 = reshape(xk2,n,1);
xk3 = u(27+n:27+2*n-1);
xk3 = reshape(xk3,n,1);
%-------------------------------------------------------------------------%

%-------------------------------------------------------------------------%
%                       GET SYSPARAMS
```

```
%------------------------------------------------------------------------%

%------------------------------------------------------------------------%
vswM = ParamSys.vswM;
vdM = ParamSys.vdM;
vswD = ParamSys.vswD;
vdD = ParamSys.vdD;

TL1_delta  = ParamSys.TL1_variation;
time_delta_TL1 = ParamSys.time_variation_TL1;

TL2_delta  = ParamSys.TL2_variation;
time_delta_TL2 = ParamSys.time_variation_TL2;

TL3_delta  = ParamSys.TL3_variation;
time_delta_TL3 = ParamSys.time_variation_TL3;

if t > time_delta_TL1
    TL_1 = (1+TL1_delta)*ParametersIM.Tnom;
else
    %     TL_1 = ParametersIM.Tnom;
    TL_1 = 0;
end

if t > time_delta_TL2
    TL_2 = (1+TL2_delta)*ParametersIM.Tnom;
else
%     TL_2 = ParametersIM.Tnom;
    TL_2 = 0;
end

if t > time_delta_TL3
    TL_3 = (1+TL3_delta)*ParametersIM.Tnom;
else
%     TL_3 = ParametersIM.Tnom;
    TL_3 = 0;
end
%------------------------------------------------------------------------%
%------------------------------------------------------------------------%
%                       GET CONTROLLER PARAMS
%------------------------------------------------------------------------%
% Hysteresis control
h = ParamController.h; % Hysteresis band

% r_ext control
Ak = ParamController.Ak;
Bk = ParamController.Bk;
Ck = ParamController.Ck;
Dk = ParamController.Dk;
%------------------------------------------------------------------------%
%                   Arbitrary Reference Frame (unused)
%------------------------------------------------------------------------%
%                           Stator reference
%------------------------------------------------------------------------%
% 1st IM
% wa_1 = 0;
% theta_a_1 = 0;

% 2nd IM
% wa_2 = 0;
% theta_a_2 = 0;
%------------------------------------------------------------------------%

%------------------------------------------------------------------------%
%                    Auxiliary Equations for the IMs
%------------------------------------------------------------------------%
% Transformation of fluxes to currents — DQ components
Linv = ParametersXF.Linv;
% 1ST MOTOR
[ids_1, iqs_1, idr_1, iqr_1, id2_1, iq2_1] = Fluxes2Currents_xf(lambda_dC_1, ...
    lambda_qC_1, lambda_dr_1, lambda_qr_1, lambda_d2_1, lambda_q2_1, Linv);
% 2ND MOTOR
[ids_2, iqs_2, idr_2, iqr_2, id2_2, iq2_2] = Fluxes2Currents_xf(lambda_dC_2, ...
    lambda_qC_2, lambda_dr_2, lambda_qr_2, lambda_d2_2, lambda_q2_2, Linv);
% 3RD MOTOR
[ids_3, iqs_3, idr_3, iqr_3, id2_3, iq2_3] = Fluxes2Currents_xf(lambda_dC_3, ...
    lambda_qC_3, lambda_dr_3, lambda_qr_3, lambda_d2_3, lambda_q2_3, Linv);

% Electromagnetic and Load Torques
% Need to use rotor flux & current since stator flux isn't calculated
% 1ST MOTOR
Te_1 = 3/4*ParametersIM.P*(lambda_qr_1.*idr_1 — lambda_dr_1.*iqr_1);
```

```
% 2ND MOTOR
Te_2 = 3/4*ParametersIM.P*(lambda_qr_2.*idr_2 - lambda_dr_2.*iqr_2);
% 3RD MOTOR
Te_3 = 3/4*ParametersIM.P*(lambda_qr_3.*idr_3 - lambda_dr_3.*iqr_3);

% Transformation of electrical rotor speed and electrical angular position
% to mechanical rotor speed and mechanical angular position
% 1st IM
% wm_1 =  wr_1/(ParametersIM.P/2);
theta_m_1 = theta_r_1/(ParametersIM.P/2);
% 2nd IM
% wm_2 =  wr_2/(ParametersIM.P/2);
theta_m_2 = theta_r_2/(ParametersIM.P/2);
% 3rd IM
% wm_2 =  wr_2/(ParametersIM.P/2);
theta_m_3 = theta_r_3/(ParametersIM.P/2);

% Transforming stator and rotor currents from qd reference to abc variables
% Replaced with stationary assumption for efficiency
% 1st IM
% [ias_1, ibs_1, ics_1] = qd2abcs(iqs_1, ids_1, theta_a_1);
[ibs_1, ics_1] = statbc(iqs_1, ids_1);
% 2nd IM
%[ias_2, ibs_2, ics_2] = qd2abcs(iqs_2, ids_2, theta_a_2);
[ibs_2, ics_2] = statbc(iqs_2, ids_2);
% 3rd IM
%[ias_3, ibs_3, ics_3] = qd2abcs(iqs_3, ids_3, theta_a_3);
[ibs_3, ics_3] = statbc(iqs_3, ids_3);

% 1st XFMR
% [ia2_1, ib2_1, ic2_1] = qd2abcs(iq2_1, id2_1, theta_a_1);
[ib2_1, ic2_1] = statbc(iq2_1, id2_1);
% 2nd XFMR
% [ia2_2, ib2_2, ic2_2] = qd2abcs(iq2_2, id2_2, theta_a_2);
[ib2_2, ic2_2] = statbc(iq2_2, id2_2);
% 3rd XFMR
% [ia2_3, ib2_3, ic2_3] = qd2abcs(iq2_3, id2_3, theta_a_3);
[ib2_3, ic2_3] = statbc(iq2_3, id2_3);

% Rotor Voltages (squirrel cage) -  DQ components
% 1st IM
vdr_1 = 0;
vqr_1 = 0;
% 2nd IM
vdr_2 = 0;
vqr_2 = 0;
% 3rd IM
vdr_3 = 0;
vqr_3 = 0;

% Transformer secondary voltages are defined in position sync section
%--------------------------------------------------------------------------%

%--------------------------------------------------------------------------%
%                      INDIRECT (ROTOR) FIELD ORIENTED CONTROL
%                           INDUCTION MOTOR 1
%--------------------------------------------------------------------------%
[lambda_dr_star, wm_star] = RefLambdaAndwm(t, ParametersIM);

% Indirect field oriented control drive
[iqs_star, ids_star, di_wm_dt, wrf_est] = IFOCDrive(ParametersIM, ...
    lambda_dr_star, wm_star, wr_1, I_Action_CL_w_1);

% Converting currents to abc variables for Hysteresis current control
% Commanded currents
[ias_star, ibs_star, ics_star] = qd2abcs(iqs_star, ids_star, theta_e);
i_abcs_star = [ias_star, ibs_star, ics_star];

% Measured/real currents
i_abcs = [iqs_1, ibs_1, ics_1];

% Hysteresis current control
[S] = myHysteresisCurrentControl(i_abcs', i_abcs_star', h);

% 3phase bridge converter voltages
vdc = ParametersIM.Vs*sqrt(2); % peak of line-to-line voltage
IM = [(iqs_1+iqs_2+iqs_3) (ibs_1+ibs_2+ibs_3) (ics_1+ics_2+ics_3)];
[vas, vbs, vcs] = my3PhaseConverterVoltLoss(S', vdc, vswM, vdM, IM);

% Converting vabcs voltages to vqds
[vqsC, vdsC] = statqd(vas, vbs, vcs);

```

```
%-------------------------------------------------------------------------%

%-------------------------------------------------------------------------%
%                       VOLTAGE ADJUSTMENT CONTROL
%                           INDUCTION MOTOR 2
%-------------------------------------------------------------------------%

%-------------------------------------------------------------------------%
%                      CONTROLLER EQUATIONS
%-------------------------------------------------------------------------%
% Controller input vector uk
% For now, only the position error

uk2(1) = theta_m_1 - theta_m_2;
xk2_dot = Ak*xk2 + Bk*uk2;
yk2 = Ck*xk2 + Dk*uk2;

uk3(1) = theta_m_1 - theta_m_3;
xk3_dot = Ak*xk3 + Bk*uk3;
yk3 = Ck*xk3 + Dk*uk3;

% Controller output vector yk
% For now, only motor 2/3 voltage adjustment
% delv2_cont: continuous variable voltage drop, ideal target
% For PI controller, yk is in units of V/rad
delv2_cont = yk2(1);
delv2_cont = max(delv2_cont,0);
delv2_cont = min(delv2_cont,ParamController.delv_max);

delv3_cont = yk3(1);
delv3_cont = max(delv3_cont,0);
delv3_cont = min(delv3_cont,ParamController.delv_max);
%-------------------------------------------------------------------------%

% Calculate voltage drops for motor 1 using dummy converter for symmetry
S_d2 = [false false false true true true];
% ID = [ia2_1, ib2_1, ic2_1];
ID = [iq2_1, ib2_1, ic2_1];
[vad, vbd, vcd] = my3PhaseConverterVoltLoss(S_d2, 0, vswD, vdD, ID);
% Transforming vabcs to vds and vqs
% Replaced with stationary assumption for efficiency
% [vqd, vdd] = abcs2qd(vad, vbd, vcd, theta_a_1);
[vqd_1, vdd_1] = statqd(vad, vbd, vcd);
% Refer variable
vq2_1 = (vqd_1*ParametersXF.N);
vd2_1 = (vdd_1*ParametersXF.N);
if t > ParamController.tstart
    % Controlable voltage drop via sine-triangle PWM, assuming synchronization
    f_d = 4.987654321e3;
    vdc_d = vdc * ParametersXF.vdcR;

    % Motor 2
    vqd_e = delv2_cont / ParametersXF.N;
    vdd_e = 0;
    [S_d2,~,~,~,~] = my3HarmonicSineTri_v2(t, theta_e, vqd_e, vdd_e, vdc_d, f_d);
    % ID = [ia2_2, ib2_2, ic2_2];
    ID = [iq2_2, ib2_2, ic2_2];
    [vad, vbd, vcd] = my3PhaseConverterVoltLoss(S_d2, vdc_d, vswD, vdD, ID);

    % Transforming vabcs to vds and vqs
    % Replaced with stationary assumption for efficiency
    % [vqd, vdd] = abcs2qd(vad, vbd, vcd, theta_a_2);
    [vqd_2, vdd_2] = statqd(vad, vbd, vcd);

    % Refer variable
    vq2_2 = (vqd_2*ParametersXF.N);
    vd2_2 = (vdd_2*ParametersXF.N);

    % Motor 3
    vqd_e = delv3_cont / ParametersXF.N;
    vdd_e = 0;
    [S_d3,~,~,~,~] = my3HarmonicSineTri_v2(t, theta_e, vqd_e, vdd_e, vdc_d, f_d);
    % ID = [ia2_3, ib2_3, ic2_3];
    ID = [iq2_3, ib2_3, ic2_3];
    [vad, vbd, vcd] = my3PhaseConverterVoltLoss(S_d3, vdc_d, vswD, vdD, ID);

    % Transforming vabcs to vds and vqs
    % Replaced with stationary assumption for efficiency
    % [vqd, vdd] = abcs2qd(vad, vbd, vcd, theta_a_2);
    [vqd_3, vdd_3] = statqd(vad, vbd, vcd);

    % Refer variable
```

```
    vq2_3 = (vqd_3*ParametersXF.N);
    vd2_3 = (vdd_3*ParametersXF.N);
else
    % Voltage drop is only active when torques are different
    % Since motors are identical in this state, use same voltages
    S_d3 = S_d2;
    vqd_2 = vqd_1;
    vdd_2 = vdd_1;
    vqd_3 = vqd_1;
    vdd_3 = vdd_1;
    vq2_2 = vq2_1;
    vd2_2 = vd2_1;
    vq2_3 = vq2_1;
    vd2_3 = vd2_1;
end
%--------------------------------------------------------------------------%

%--------------------------------------------------------------------------%
%                      OUTPUT STRUCTURES
%--------------------------------------------------------------------------%
% Switching states
Sw.S = S;
Sw.S_d2 = S_d2;
Sw.S_d3 = S_d3;

% 1st motor
IM1.TL = TL_1;
IM1.Te = Te_1;
IM1.vqC = vqsC;
IM1.vdC = vdsC;
% IM1.wa = wa_1;
% IM1.theta_a = theta_a_1;
IM1.delv1_vq = vqd_1;
IM1.delv1_vd = vdd_1;
IM1.wm_star = wm_star;

% 2nd motor
IM2.TL = TL_2;
IM2.Te = Te_2;
% IM2.wa = wa_2;
% IM2.theta_a = theta_a_2;
IM2.delv2_vq = vqd_2;
IM2.delv2_vd = vdd_2;
IM2.delv2_cont = delv2_cont;

% 3rd motor
IM3.TL = TL_3;
IM3.Te = Te_3;
% IM3.wa = wa_3;
% IM3.theta_a = theta_a_3;
IM3.delv3_vq = vqd_3;
IM3.delv3_vd = vdd_3;
IM3.delv3_cont = delv3_cont;

%--------------------------------------------------------------------------%


%--------------------------------------------------------------------------%
%                      DIFF. STATES EQUATIONS
%--------------------------------------------------------------------------%
% Extracting parameters here, assuming identical motors
rs = ParametersIM.rs;
rr = ParametersIM.rr;
r1 = ParametersXF.r1;
r2 = ParametersXF.r2;
J = ParametersIM.J;
P = ParametersIM.P;
b = ParametersIM.b;

% Equations used are altered to assume stationary ref. frame for efficiency
%--------------------------------------------------------------------------%
%                          1ST MOTOR
%--------------------------------------------------------------------------%
% y(1) = vdC_1 - (rs+r1)*ids_1 + wa_1 * lambda_qC_1;      % lambda_dC_1
% y(2) = vqC_1 - (rs+r1)*iqs_1 - wa_1 * lambda_dC_1;      % lambda_qC_1
% y(3) = vdr_1 - rr*idr_1 + (wa_1-wr_1)*lambda_qr_1;      % lambda_dr_1
% y(4) = vqr_1 - rr*iqr_1 - (wa_1-wr_1)*lambda_dr_1;      % lambda_qr_1
% y(5) = vd2_1 - r2*id2_1 + wa_1 * lambda_q2_1;           % lambda_d2_1
% y(6) = vq2_1 - r2*iq2_1 - wa_1 * lambda_d2_1;           % lambda_q2_1
y(1) = vdsC - (rs+r1)*ids_1;                    % lambda_dC_1
y(2) = vqsC - (rs+r1)*iqs_1;                    % lambda_qC_1
y(3) = vdr_1 - rr*idr_1 - wr_1*lambda_qr_1;     % lambda_dr_1
```

```
y(4) = vqr_1 - rr*iqr_1 + wr_1*lambda_dr_1;     % lambda_qr_1
y(5) = vd2_1 - r2*id2_1;                        % lambda_d2_1
y(6) = vq2_1 - r2*iq2_1;                        % lambda_q2_1
y(7) = (1/J)*((Te_1-TL_1)*P/2 - b*wr_1);        % wr_1
y(8) = wr_1;                                    % theta_r_1
%-------------------------------------------------------------------------%
%              INDIRECT FIELD ORIENTED CONTROL (1ST MOTOR)
%-------------------------------------------------------------------------%
y(9) = wrf_est;   % Rotor flux angular position (electric)
y(10) = di_wm_dt;  % Integral action of the closed loop control of wm
%-------------------------------------------------------------------------%
%-------------------------------------------------------------------------%
%                          2ND MOTOR
%-------------------------------------------------------------------------%
% y(11) = vdC_1 - (rs+r1)*ids_2 + wa_2 * lambda_qC_2;      % lambda_dC_2
% y(12) = vqC_1 - (rs+r1)*iqs_2 - wa_2 * lambda_dC_2;      % lambda_qC_2
% y(13) = vdr_2 - rr*idr_2 + (wa_2-wr_2)*lambda_qr_2;      % lambda_dr_2
% y(14) = vqr_2 - rr*iqr_2 - (wa_2-wr_2)*lambda_dr_2;      % lambda_qr_2
% y(15) = vd2_2 - r2*id2_2 + wa_2 * lambda_q2_2;           % lambda_d2_2
% y(16) = vq2_2 - r2*iq2_2 - wa_2 * lambda_d2_2;           % lambda_q2_2
y(11) = vdsC - (rs+r1)*ids_2;                   % lambda_dC_2
y(12) = vqsC - (rs+r1)*iqs_2;                   % lambda_qC_2
y(13) = vdr_2 - rr*idr_2 - wr_2*lambda_qr_2;    % lambda_dr_2
y(14) = vqr_2 - rr*iqr_2 + wr_2*lambda_dr_2;    % lambda_qr_2
y(15) = vd2_2 - r2*id2_2;                       % lambda_d2_2
y(16) = vq2_2 - r2*iq2_2;                       % lambda_q2_2
y(17) = (1/J)*((Te_2-TL_2)*P/2 - b*wr_2);       % wr_2
y(18) = wr_2;                                   % theta_r_2
%-------------------------------------------------------------------------%
%                          3RD MOTOR
%-------------------------------------------------------------------------%
y(19) = vdsC - (rs+r1)*ids_3;                   % lambda_dC_3
y(20) = vqsC - (rs+r1)*iqs_3;                   % lambda_qC_3
y(21) = vdr_3 - rr*idr_3 - wr_3*lambda_qr_3;    % lambda_dr_3
y(22) = vqr_3 - rr*iqr_3 + wr_3*lambda_dr_3;    % lambda_qr_3
y(23) = vd2_3 - r2*id2_3;                       % lambda_d2_3
y(24) = vq2_3 - r2*iq2_3;                       % lambda_q2_3
y(25) = (1/J)*((Te_3-TL_3)*P/2 - b*wr_3);       % wr_3
y(26) = wr_3;                                   % theta_r_3
%-------------------------------------------------------------------------%
%                       CONTROLLER
%-------------------------------------------------------------------------%
% Automatic state-building diff. equations of the controller, based on its order
y(27:27+n-1) = xk2_dot;
y(27+n:27+2*n-1) = xk3_dot;
%-------------------------------------------------------------------------%

y = y';
end
```

B.5 Auxiliary Functions

```
% Inputs are the abc phase variables in the stationary reference frame and
% the angle of the arbitrary rotating reference frame theta.
% Outputs are the qd variables in the (arbitrary) rotating reference
% frame.
function [x_q, x_d] = abcs2qd(x_as, x_bs, x_cs, theta)

% Reshaping the vectors so all of them are column vectors
n = numel(theta);
theta = reshape(theta,n,1);
x_as = reshape(x_as,n,1);
x_bs = reshape(x_bs,n,1);
x_cs = reshape(x_cs,n,1);

x_q = 2/3*(x_as.*cos(theta)+x_bs.*cos(theta-2*pi/3)+x_cs.*cos(theta+2*pi/3));
x_d = 2/3*(x_as.*sin(theta)+x_bs.*sin(theta-2*pi/3)+x_cs.*sin(theta+2*pi/3));
end
```

```
function wrm_star = DesiredSpeed(t)

NumPoints = length(t);
wrm_star = zeros(NumPoints,1);

% Commanded speed set as a ramp from t=0.1 to t=2.6 with final value of
% 188.5 rad/s
t_1 = 0;
t_2 = 0.1;
```

```
wrm_final = 188.5;
slope = wrm_final/(t_2-t_1);

for i = 1:NumPoints
   if t(i) >= t_1 && t(i) < t_2
       wrm_star(i) = slope*(t(i)-t_1);
   elseif t(i) >= t_2
       wrm_star(i) = wrm_final;
   end
end

end
```

```
% [Translated from original description in Portugese]
% Function that transforms fluxes to currents using passed machine
% parameters. The fluxes and currents are in DQ components.
% Flux can be passed as 4 vectors, one for each flux value,
% in order: lambda_ds, lambda_qs, lambda_dr, lambda_qr; or a matrix,
% with 4 columns, referring to each of the fluxes, and as many
% rows as needed to describe time behavior of fluxes. The output currents
% will match the inputs, and may be scalars or vectors.
function [ids, iqs, idr, iqr]= Fluxes2Currents(lambda_ds, lambda_qs, ...
    lambda_dr, lambda_qr, ParamatersIM)

NumPoints = length(lambda_ds);

ids = zeros(NumPoints,1);
iqs = zeros(NumPoints,1);
idr = zeros(NumPoints,1);
iqr = zeros(NumPoints,1);

for i = 1:NumPoints

    D = ParamatersIM.Ls*ParamatersIM.Lr-ParamatersIM.Lm^2;
    MatrizTransformation = 1/D*[ParamatersIM.Lr -ParamatersIM.Lm;
                               -ParamatersIM.Lm ParamatersIM.Ls];

    Vetor_Correntes_d = MatrizTransformation*[lambda_ds(i); lambda_dr(i)];
    Vetor_Correntes_q = MatrizTransformation*[lambda_qs(i); lambda_qr(i)];

    ids(i,1) = Vetor_Correntes_d(1);  % i_ds
    iqs(i,1) = Vetor_Correntes_q(1);  % i_qs
    idr(i,1) = Vetor_Correntes_d(2);  % i_dr
    iqr(i,1) = Vetor_Correntes_q(2);  % i_qr
end
end
```

```
function [ids, iqs, idr, iqr, id2, iq2]= Fluxes2Currents_xf(lambda_dC, ...
    lambda_qC, lambda_dr, lambda_qr, lambda_d2, lambda_q2, Linv)

NumPoints = numel(lambda_dC);

ids = zeros(NumPoints,1);
iqs = zeros(NumPoints,1);
idr = zeros(NumPoints,1);
iqr = zeros(NumPoints,1);
id2 = zeros(NumPoints,1);
iq2 = zeros(NumPoints,1);

for i = 1:NumPoints
    Currents_d = Linv*[lambda_dC(i); lambda_dr(i); lambda_d2(i)];
    Currents_q = Linv*[lambda_qC(i); lambda_qr(i); lambda_q2(i)];

    ids(i,1) = Currents_d(1);  % i_ds
    iqs(i,1) = Currents_q(1);  % i_qs
    idr(i,1) = Currents_d(2);  % i_dr
    iqr(i,1) = Currents_q(2);  % i_qr
    id2(i,1) = Currents_d(3);  % i_d2
    iq2(i,1) = Currents_q(3);  % i_q2
end
end
```

```
function [iqs_star, ids_star, di_wm_dt, we, Tem_ref] = IFOCDrive(ParametersIM, ...
    lambda_dr_star, wm_star, wr, I_Action_CL_w, t)
% Serves for both single time and time vector

%-------------------------------------------------------------------------%
%                     CLOSE-LOOP SPEED CONTROL
%-------------------------------------------------------------------------%
% Tunning kp_wm and ki_wm of the PI controller

```

```
tau_wm = 0.1;
% tau_wm = 0.02;
p1 = 1/(tau_wm);
p2 = 5*p1;
kp_wm = (p1+p2)*ParametersIM.J_est - ParametersIM.b_est;
ki_wm = p1*p2*ParametersIM.J_est;

% Speed error to feed the PI controller
wm =  wr/(ParametersIM.P/2); % rotor electrical speed to mechanical speed
wm_error = wm_star-wm;

% PI action on speed error to generate commanded torque
Tem_ref = kp_wm*wm_error + I_Action_CL_w;

% Saturation and anti-windup effect
% Max and min torque values (saturation):
Te_max = 10*ParametersIM.Tnom;
Te_min = -Te_max;

di_wm_dt = zeros(1,length(lambda_dr_star));
for i = 1:length(Tem_ref)

    % Saturation at Te_max
    Tem_ref(i) = min(Tem_ref(i), Te_max);

    % Saturation at Te_max
    Tem_ref(i) = max(Tem_ref(i), Te_min);

    % Anti-windup integration:
    if (Tem_ref(i) == Te_max) && (wm_error(i)*I_Action_CL_w(i))>0
        di_wm_dt(i) = 0;

    elseif (Tem_ref(i) == Te_min) && (wm_error(i)*I_Action_CL_w(i))>0
        di_wm_dt(i) = 0;

    else
        di_wm_dt(i) = ki_wm*wm_error(i);

    end

end
%--------------------------------------------------------------------------%

%--------------------------------------------------------------------------%
%                    iqs_star and ids_star Calculations
%--------------------------------------------------------------------------%

% ids_star depending on the commanded flux (algebraic relationship)
ids_star = lambda_dr_star/ParametersIM.Lm_est;

% iqs_star depending on the commanded torque (algebraic relationship)
k_Tem = 3/4*ParametersIM.P*ParametersIM.Lm_est/ParametersIM.Lr_est;
iqs_star = Tem_ref/k_Tem./(lambda_dr_star+eps);
%--------------------------------------------------------------------------%

%--------------------------------------------------------------------------%
%                    Slip/we Calculations
%--------------------------------------------------------------------------%
% Rotor flux speed estimation (indirect control)
we = ParametersIM.rr_est/ParametersIM.Lr_est*(iqs_star./(ids_star+eps))+wr;

%--------------------------------------------------------------------------%

end
```

```
function [S, da, db, dc, tri] = my3HarmonicSineTri_v2(t, theta_e, vqs_e, vds_e, Vdc, f_sw)
%--------------------------------------------------------------------------%
%                        FUNCTION DESCRIPTION
%--------------------------------------------------------------------------%
% Function that implements sine-triangle modulation with third-harmonic
% injection
%
% Reference: Analysis of Electric Machinery and Drive Systems, P.C Krause,
% 2nd Edition, Chapter 13
%
% INPUTS:
% t: simulation time [s]
% theta_e: electrical angle of the commanded voltages [rad]
% vqs_e, vds_e: qds input voltages in the synchronous reference frame (commanded values) [V]
% Vdc: power converter DC voltage (peak of the line-to-line voltage) [V]
% f_sw: switching frequency [Hz]
%
```

```
% OUTPUTS:
% S: vector with the 6 switching states: T1 — T6
%-------------------------------------------------------------------------%

%-------------------------------------------------------------------------%
%                       AUXILIARY EQUATIONS
%-------------------------------------------------------------------------%
% Converter angle (electrical angle minus the angle of the complex vector
% vqds_e)
theta_c = theta_e — angle(vqs_e — 1i*vds_e);

% Limits the angle in the interval [0, 2*pi) TODO: Maybe this is dispensable...
theta_c = mod(theta_c, 2*pi);

% Frequency of the triangle waveform
omega_tri = 2*pi*f_sw; % [rad/s]

% Amplitude of the main voltage
d = sqrt(vqs_e.^2 + vds_e.^2)/(0.5*Vdc);

% Amplitude of third-harmonic injection
d3 = d/6;
%-------------------------------------------------------------------------%

% 1st: Generate sawtooth (triangle waveform), tri (between -1 and 1)
tri = sawtooth(omega_tri*t, 0.5);

% 2nd: Third harmonic injection
da = d*cos(theta_c) — d3*cos(3*theta_c);
db = d*cos(theta_c — 2*pi/3) — d3*cos(3*theta_c);
dc = d*cos(theta_c + 2*pi/3) — d3*cos(3*theta_c);

% 3rd: Generate the output logical signals: T1 — T6
% a-phase
T1 = da>= tri;
T4 = ~T1;

% b-phase
T2 = db>= tri;
T5 = ~T2;

% c-phase
T3 = dc>= tri;
T6 = ~T3;

% 4th: Build output vector S with the logical signals T1-T6
S = [T1, T2, T3, T4, T5, T6];

end
```

```
function [vas, vbs, vcs] = my3PhaseConverterVoltLoss(S, vdc, vsw, vd, I)
%-------------------------------------------------------------------------%
%                        FUNCTION DESCRIPTION
%-------------------------------------------------------------------------%
% Function that determines the 3-phase bridge converter output voltages.
% Incorporates switching losses.
%
% Reference: Analysis of Electric Machinery and Drive Systems, P.C Krause,
% 2nd Edition, Chapter 13
%
% INPUTS:
% S: vector with the 6 switching states: T1 — T6
% vdc: dc voltage applied to the converter bridge [V]
% vsw: switch voltage drop (for "forward" current)
% vd: diode voltage drop (for "backward" current)
% I: vector with the 3 currents (ias, ibs, ics)

% OUTPUTS:
% vas, vbs, vcs: line-to-neutral 3-phase voltages
%-------------------------------------------------------------------------%

% 1st: Generate line-to-ground voltages
if S(1)
    vag = vdc — (vsw * (I(1)>0)) + (vd * (I(1)<0));
else
    vag = — (vd * (I(1)>0)) + (vsw * (I(1)<0));
end
if S(2)
    vbg = vdc — (vsw * (I(2)>0)) + (vd * (I(2)<0));
else
    vbg = — (vd * (I(2)>0)) + (vsw * (I(2)<0));
end
```

```
if S(3)
    vcg = vdc - (vsw * (I(3)>0)) + (vd * (I(3)<0));
else
    vcg = - (vd * (I(3)>0)) + (vsw * (I(3)<0));
end

% 2nd: Generate line-to-neutral voltages
vas = 2/3*vag - 1/3*(vbg + vcg);
vbs = 2/3*vbg - 1/3*(vag + vcg);
vcs = 2/3*vcg - 1/3*(vag + vbg);

end
```

```
function [S] = myHysteresisCurrentControl(i_abcs, i_abcs_star, h)
% Serves for both single time and time vector

% Way to hold previous state
persistent S_old;
if isempty(S_old)
        S_old = [0; 1; 0; 1; 0; 1];
end

% In case of time vector case, has to allocate memory
num_points = numel(i_abcs)/3;
T1 = zeros(1,num_points);
T2 = zeros(1,num_points);
T3 = zeros(1,num_points);
T4 = zeros(1,num_points);
T5 = zeros(1,num_points);
T6 = zeros(1,num_points);
S  = zeros(6,num_points);


for i=1:num_points
    % Current i_as and switches T1 and T4
    if i_abcs(1,i) > i_abcs_star(1,i) + h
        T1(i) = 0;
        T4(i) = 1;
    elseif i_abcs(1,i) < i_abcs_star(1,i) - h
        T1(i) = 1;
        T4(i) = 0;
    else
        T1(i) = S_old(1);
        T4(i) = S_old(4);
    end

    % Current i_bs and switches T2 and T5
    if i_abcs(2,i) > i_abcs_star(2,i) + h
        T2(i) = 0;
        T5(i) = 1;
    elseif i_abcs(2,i) < i_abcs_star(2,i) - h
        T2(i) = 1;
        T5(i) = 0;
    else
        T2(i) = S_old(2);
        T5(i) = S_old(5);
    end

    % Current i_cs and switches T3 and T6
    if i_abcs(3,i) > i_abcs_star(3,i) + h
        T3(i) = 0;
        T6(i) = 1;
    elseif i_abcs(3,i) < i_abcs_star(3,i) - h
        T3(i) = 1;
        T6(i) = 0;
    else
        T3(i) = S_old(3);
        T6(i) = S_old(6);
    end

    % Build output vector S from the logical signals T1-T6
    S(:,i) = [T1(i); T2(i); T3(i); T4(i); T5(i); T6(i)];

    S_old = S(:,i);
end

end
```

```
function [S] = myPWM(t, d, f_sw)
%-------------------------------------------------------------------------%
%                        FUNCTION DESCRIPTION
%-------------------------------------------------------------------------%
```

```
% Function that implements pulse-width modulation.
%
% Reference: Analysis of Electric Machinery and Drive Systems, P.C Krause,
% 2nd Edition, Chapter 13
%
% INPUTS:
% t: time value [s]
% omega_c: converter frequency [rad/s]
% d: duty cicle
% f_sw: switching frequency [Hz]
%
% OUTPUTS:
% S: vector with the 6 switching states: T1 - T6
%-------------------------------------------------------------------------%

%-------------------------------------------------------------------------%
%                       AUXILIARY EQUATIONS
%-------------------------------------------------------------------------%
% Frequency of the triangle waveform
omega_tri = 2*pi*f_sw; % [rad/s]
%-------------------------------------------------------------------------%

% 2nd: Generate sawtooth (triangle waveform), tri
tri = 0.5 + 0.5*sawtooth(omega_tri * t, 0.5);

% 3rd: Generate comparator output, c
threshold = 1e-4;
c = d > (tri + threshold);

% 4th: Generate the output logical signals: T1 - T6
T1 = c;

% 5th: Build output vector S from the logical signals T1-T6
S = T1;

end
```

```
% Inputs are the qd variables in the arbitrary reference frame and angle
% of the arbitrary rotating reference frame theta.
% Outputs are the abc variables in the stationary reference frame
function [x_as, x_bs, x_cs] = qd2abcs(x_q, x_d, theta)

% Reshaping the vectors so all of them are column vectors
n = numel(theta);
theta = reshape(theta,n,1);
x_q = reshape(x_q,n,1);
x_d = reshape(x_d,n,1);

x_as = x_q.*cos(theta) + x_d.*sin(theta);
x_bs = x_q.*cos(theta-2*pi/3) + x_d.*sin(theta-2*pi/3);
x_cs = x_q.*cos(theta+2*pi/3) + x_d.*sin(theta+2*pi/3);
end
```

```
% Inputs are the qd variables in the old arbitrary reference frame and
% difference in angle betweeen the reference frames (new - old).
% Outputs are the qd variables in the new reference frame
function [y_q, y_d] = qd2qd(x_q,x_d,theta)

n = numel(theta);
theta = reshape(theta,n,1);
x_q = reshape(x_q,n,1);
x_d = reshape(x_d,n,1);

y_q = x_q.*cos(theta) - x_d.*sin(theta);
y_d = x_q.*sin(theta) + x_d.*cos(theta);
end
```

```
function [lambda_dr_ref, wm_ref] = RefLambdaAndwm(time, ParametersIM)

lambda_nom = sqrt(2/3)*ParametersIM.Vs/(2*pi*ParametersIM.f);
t_lambda_nom = 0.5;
lambda_ref_slope = lambda_nom/t_lambda_nom;

wm = 1800*pi/30;

t_1_wm = 0.6;
t_2_wm = 1.5;

% t_1_wm = 1;
% t_2_wm = 1.5;

```

```
wm_slope_acc = wm/(t_2_wm - t_1_wm);

t_3_wm = 100;
t_4_wm = 200;
wm_slope_des = wm/(t_4_wm - t_3_wm);

NumPoints = length(time);

lambda_dr_ref = zeros(NumPoints,1);
wm_ref = zeros(NumPoints,1);


for i = 1:NumPoints
    if time(i) < t_lambda_nom
        lambda_dr_ref(i)  = lambda_ref_slope*time(i);
        wm_ref(i) = 0;
    elseif time(i) >= t_lambda_nom && time(i) < t_1_wm
        lambda_dr_ref(i)  = lambda_nom;
        wm_ref(i) = 0;
    elseif time(i) >= t_1_wm && time(i) < t_2_wm
        lambda_dr_ref(i)  = lambda_nom;
        wm_ref(i) = wm_slope_acc*(time(i)-t_1_wm);
    elseif time(i) >= t_2_wm && time(i) < t_3_wm
        lambda_dr_ref(i)  = lambda_nom;
        wm_ref(i) = wm;
    elseif time(i) >= t_3_wm && time(i) < t_4_wm
        lambda_dr_ref(i)  = lambda_nom;
        wm_ref(i) = wm - wm_slope_des*(time(i)-t_3_wm);
    elseif time(i) >= t_4_wm
        lambda_dr_ref(i)  = lambda_nom;
        wm_ref(i) = 0;
    end
end

end
```

```
function ParametersIM = ReturnParametersIM_15hp
% 15 hp Motor - Kirtley (Electric power principles: sources, conversion,
% distribution and use, Wiley, 2010) p. 293
ParametersIM.rs      =   0.06;      % Stator resistance             [ohms]
ParametersIM.rr      =   0.15;      % Rotor resistance              [ohms]
ParametersIM.Lls     =   0.44/(2*pi*60);    % Stator leakage inductance    [Henry]
ParametersIM.Llr     =   0.43/(2*pi*60);    % Rotor leakage inductance     [Henry]
ParametersIM.Lm      =   12.6/(2*pi*60);    % Magnetizing inductance       [Henry]
ParametersIM.J       =   0.445;              % Inertia                      [kg.m^2]
ParametersIM.P       =   4;                  % Pole numbers
ParametersIM.Vs      =   240;                % Rated voltage (line-to-line rms)  [V]
ParametersIM.Ls      =   ParametersIM.Lls + ParametersIM.Lm;
ParametersIM.Lr      =   ParametersIM.Llr + ParametersIM.Lm;
ParametersIM.Lsprime =   ParametersIM.Ls - ParametersIM.Lm^2/ParametersIM.Lr;
ParametersIM.rsprime =   ParametersIM.rs + ParametersIM.rr*ParametersIM.Lm^2/ParametersIM.Lr^2;
% ParametersIM.b       =   0.00;                                          % Friction factor     [kg.m^2/s]
ParametersIM.b       =   5.41e-4;                                        % Friction factor     [kg.m^2/s]

ParametersIM.Pnom  =   15*746;                                          % Nominal motor power [W]
ParametersIM.f     =   60;                                              % Nominal frequency   [Hz]
ParametersIM.speed =   1710*pi/30;                                      % Nominal rotor speed [rad/s]
ParametersIM.Tnom  =   ParametersIM.Pnom/(ParametersIM.speed);          % Nominal torque      [N.m]

%-------------------------------------------------------------------------%
%                        ESTIMATED PARAMETERS
%-------------------------------------------------------------------------%

ParametersIM.rs_est      =   ParametersIM.rs;
ParametersIM.rr_est      =   ParametersIM.rr;
ParametersIM.Lls_est     =   ParametersIM.Lls;
ParametersIM.Llr_est     =   ParametersIM.Llr;
ParametersIM.Lm_est      =   ParametersIM.Lm;
ParametersIM.J_est       =   ParametersIM.J;
ParametersIM.b_est       =   ParametersIM.b;

ParametersIM.Ls_est      =   ParametersIM.Lls_est + ParametersIM.Lm_est;
ParametersIM.Lr_est      =   ParametersIM.Llr_est + ParametersIM.Lm_est;
ParametersIM.Lsprime_est =   ParametersIM.Ls_est - ParametersIM.Lm_est^2/ParametersIM.Lr_est;
ParametersIM.rsprime_est =   ParametersIM.rs_est + ParametersIM.rr_est*ParametersIM.Lm_est^2/ParametersIM.Lr_est^2;
%-------------------------------------------------------------------------%


end
```

```
% Inputs are the qd variables in the stationary reference frame.
% Outputs are the b and c variables. In this ref. frame, a=q
```

```
function [x_b, x_c] = statbc(x_q,x_d)

n = numel(x_q);
x_q = reshape(x_q,n,1);
x_d = reshape(x_d,n,1);

x_b = x_q.*-0.5 - x_d.*sqrt(3)/2;
x_c = x_q.*-0.5 + x_d.*sqrt(3)/2;
end
```

```
% Inputs are the abc phase variables in the stationary reference frame.
% Outputs are the qd variables in the stationary reference frame.
function [x_q, x_d] = statqd(x_as, x_bs, x_cs)

% Reshaping the vectors so all of them are column vectors
n = numel(x_as);
x_as = reshape(x_as,n,1);
x_bs = reshape(x_bs,n,1);
x_cs = reshape(x_cs,n,1);

x_q = 2/3*(x_as+x_bs.*-0.5+x_cs.*-0.5);
x_d = 2/3*(x_bs.*-sqrt(3)/2+x_cs.*sqrt(3)/2);
end
```

B.6 Data Analysis

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FullPowerAnalysis.m
% Analyze data for synchronization methods and generate plots.
% Run after loading data from runs of both sync methods.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all

%% Steady state slip calculation and voltage drop estimation
P = ParametersIM.P;
we = [diff(R_theta_e) diff(theta_e_out)]/(t(2)-t(1));
we = mean(we(500000:end,:));
% Average estimate across runs to account for losses
wr = mean([R_wm_1(500000:end) wm_1_out(500000:end,:)]*P/2);
s = (we-wr)./we;
% Calculate reactances
Xm = Lm*we;
Xss = (Lls+Lm)*we;
Xrr = (Llr+Lm)*we;
% Proportionality constant
K = 3*P/2.*(Xm.^2/we).*rr.*s./((rs*rr+s.*(Xm.^2-Xss.*Xrr)).^2+(rr*Xss+s.*rs.*Xrr).^2);
% Steady-state voltage drops, using nominal voltage as base
Vas1 = ParametersIM.Vs/sqrt(3);
DelV2 = -Vas1+sqrt(Vas1^2+TL2_variation.*ParametersIM.Tnom./K);
DelV3 = -Vas1+sqrt(Vas1^2+TL3_variation.*ParametersIM.Tnom./K);

% Energy measurements are over sync period (excluding startup)
%% Compute power and energy measurements for resistor method
% Power into main converter
[R_ias_1, R_ibs_1, R_ics_1] = statabcMult(R_iqs_1, R_ids_1);
[R_ias_2, R_ibs_2, R_ics_2] = statabcMult(R_iqs_2, R_ids_2);
[R_ias_3, R_ibs_3, R_ics_3] = statabcMult(R_iqs_3, R_ids_3);
R_ias_tot = R_ias_1+R_ias_2+R_ias_3;
R_ibs_tot = R_ibs_1+R_ibs_2+R_ibs_3;
R_ics_tot = R_ics_1+R_ics_2+R_ics_3;
R_idc = R_S1.*R_ias_tot + R_S2.*R_ibs_tot + R_S3.*R_ics_tot;
vdc = ParametersIM.Vs*sqrt(2);
R_pdc = R_idc.*vdc;
R_Edc = trapz(t(200001:end), R_pdc(200001:end));

% Power output of main converter
[R_vasC, R_vbsC, R_vcsC] = statabcMult(R_vqsC, R_vdsC);
R_pCout = R_vasC.*R_ias_tot + R_vbsC.*R_ibs_tot + R_vcsC.*R_ics_tot;
R_ECout = trapz(t(200001:end), R_pCout(200001:end));

% Power input to motors
[R_vas_1, R_vbs_1, R_vcs_1] = statabcMult(R_vqs_1, R_vds_1);
[R_vas_2, R_vbs_2, R_vcs_2] = statabcMult(R_vqs_2, R_vds_2);
[R_vas_3, R_vbs_3, R_vcs_3] = statabcMult(R_vqs_3, R_vds_3);
R_pmot = (R_vas_1.*R_ias_1) + (R_vas_2.*R_ias_2) + (R_vas_3.*R_ias_3) + ...
    (R_vbs_1.*R_ibs_1) + (R_vbs_2.*R_ibs_2) + (R_vbs_3.*R_ibs_3) + ...
    (R_vcs_1.*R_ics_1) + (R_vcs_2.*R_ics_2) + (R_vcs_3.*R_ics_3);
R_Emot = trapz(t(200001:end), R_pmot(200001:end));
```

```
53
54  % Electrical efficiency
55  R_effE = R_Emot/R_Edc;
56
57  % Power lost in resistors (neglecting switching losses)
58  R_ploss2 = ((R_ias_2).^2 + (R_ibs_2).^2 + (R_ics_2).^2) .* ...
59      rext2_pwm;
60  R_ploss3 = ((R_ias_3).^2 + (R_ibs_3).^2 + (R_ics_3).^2) .* ...
61      rext3_pwm;
62  R_ploss = R_ploss2 + R_ploss3;
63  R_Eloss = trapz(t(200001:end),R_ploss(200001:end));
64
65  % Power output of motors (mechanical)
66  R_pTm = (R_TL_1.*R_wm_1) + (R_TL_2.*R_wm_2) + (R_TL_3.*R_wm_3);
67  R_ETm = trapz(t(200001:end), R_pTm(200001:end));
68
69  % Motor efficiency
70  R_effMot = R_ETm/R_Emot;
71
72  % Full system electromechanical efficiency
73  R_effTm = R_ETm/R_Edc;
74
75  %% Compute power and energy measurements for aux converter method
76  % Power into main converter
77  [ias_1, ibs_1, ics_1] = statabcMult(iqs_1_out, ids_1_out);
78  [ias_2, ibs_2, ics_2] = statabcMult(iqs_2_out, ids_2_out);
79  [ias_3, ibs_3, ics_3] = statabcMult(iqs_3_out, ids_3_out);
80  ias_tot = ias_1 + ias_2 + ias_3;
81  ibs_tot = ibs_1 + ibs_2 + ibs_3;
82  ics_tot = ics_1 + ics_2 + ics_3;
83  idc = S1_out.*ias_tot + S2_out.*ibs_tot + S3_out.*ics_tot;
84  % vdc = ParametersIM.Vs*sqrt(2);
85  pdc = idc.*vdc;
86  Edc = trapz(t(200001:end), pdc(200001:end,:));
87
88  % Power output of main converter
89  [vaC, vbC, vcC] = statabcMult(vq1_1_out+vqs_1_out, vd1_1_out+vds_1_out);
90  pCout = vaC.*ias_tot + vbC.*ibs_tot + vcC.*ics_tot;
91  ECout = trapz(t(200001:end), pCout(200001:end,:));
92
93  % Power input to motors
94  [vas_1, vbs_1, vcs_1] = statabcMult(vqs_1_out, vds_1_out);
95  [vas_2, vbs_2, vcs_2] = statabcMult(vqs_2_out, vds_2_out);
96  [vas_3, vbs_3, vcs_3] = statabcMult(vqs_3_out, vds_3_out);
97  pmot = (vas_1.*ias_1) + (vas_2.*ias_2) + (vas_3.*ias_3) + ...
98      (vbs_1.*ibs_1) + (vbs_2.*ibs_2) + (vbs_3.*ibs_3) + ...
99      (vcs_1.*ics_1) + (vcs_2.*ics_2) + (vcs_3.*ics_3);
100 Emot = trapz(t(200001:end), pmot(200001:end,:));
101
102 % Electrical efficiency
103 effE = Emot./Edc;
104
105 % Power input to auxiliary converters
106 [ia2_1, ib2_1, ic2_1] = statabcMult(iq2_1_out, id2_1_out);
107 [ia2_2, ib2_2, ic2_2] = statabcMult(iq2_2_out, id2_2_out);
108 [ia2_3, ib2_3, ic2_3] = statabcMult(iq2_3_out, id2_3_out);
109 delv2_idc = zeros(size(ia2_2));
110 delv3_idc = zeros(size(ia2_2));
111 for k=1:numel(Nlist)
112     N = Nlist(k);
113     delv2_idc(:,k) = N*(Sd1_2_out(:,k).*ia2_2(:,k) + ...
114         Sd2_2_out(:,k).*ib2_2(:,k) + Sd3_2_out(:,k).*ic2_2(:,k));
115     delv3_idc(:,k) = N*(Sd1_3_out(:,k).*ia2_3(:,k) + ...
116         Sd2_3_out(:,k).*ib2_3(:,k) + Sd3_3_out(:,k).*ic2_3(:,k));
117 end
118 aux_idc = delv2_idc + delv3_idc;
119 delv_pdc = zeros(size(aux_idc));
120 for k=1:numel(vdcRlist)
121     delv_vdc = vdc * vdcRlist(k);
122     delv_pdc(:,k) = aux_idc(:,k) .* delv_vdc;
123 end
124 delv_Edc = trapz(t(200001:end), delv_pdc(200001:end,:));
125
126 % Power leaving motor lines
127 [va1_1, vb1_1, vc1_1] = statabcMult(vq1_1_out, vd1_1_out);
128 [va1_2, vb1_2, vc1_2] = statabcMult(vq1_2_out, vd1_2_out);
129 [va1_3, vb1_3, vc1_3] = statabcMult(vq1_3_out, vd1_3_out);
130 delv_pmot = (va1_1.*ias_1 + vb1_1.*ibs_1 + vc1_1.*ics_1 + ...
131     va1_2.*ias_2 + vb1_2.*ibs_2 + vc1_2.*ics_2 + ...
132     va1_3.*ias_3 + vb1_3.*ibs_3 + vc1_3.*ics_3);
133 delv_Emot = trapz(t(200001:end), delv_pmot(200001:end,:));
134
```

```
% Power output of motors (mechanical)
pTm = (TL_1_out.*wm_1_out) + (TL_2_out.*wm_2_out) + (TL_3_out.*wm_3_out);
ETm = trapz(t(200001:end), pTm(200001:end,:));

% Motor efficiency
effMot = ETm./Emot;

% Full system electromechanical efficiency
effTm = ETm./Edc;

%% Decimate readings for display in paper
t_ds = downsample(t,200);
R_wm_1_ds = downsample(R_wm_1,200);
R_wm_2_ds = downsample(R_wm_2,200);
R_wm_3_ds = downsample(R_wm_3,200);
R_theta_m_1_ds = downsample(R_theta_m_1,200);
R_theta_m_2_ds = downsample(R_theta_m_2,200);
R_theta_m_3_ds = downsample(R_theta_m_3,200);
R_Te_1_ds = downsample(R_Te_1,200);
R_Te_2_ds = downsample(R_Te_2,200);
R_Te_3_ds = downsample(R_Te_3,200);
rext2_cont_ds = downsample(rext2_cont,200);
rext3_cont_ds = downsample(rext3_cont,200);
wm_1_ds = zeros(numel(t_ds),numel(Nlist));
wm_2_ds = wm_1_ds;
wm_3_ds = wm_1_ds;
theta_m_1_ds = wm_1_ds;
theta_m_2_ds = wm_1_ds;
theta_m_3_ds = wm_1_ds;
Te_1_ds = wm_1_ds;
Te_2_ds = wm_1_ds;
Te_3_ds = wm_1_ds;
delv2_cont_ds = wm_1_ds;
delv3_cont_ds = wm_1_ds;
for i=1:numel(Nlist)
    wm_1_ds(:,i) = downsample(wm_1_out(:,i),200);
    wm_2_ds(:,i) = downsample(wm_2_out(:,i),200);
    wm_3_ds(:,i) = downsample(wm_3_out(:,i),200);
    theta_m_1_ds(:,i) = downsample(theta_m_1_out(:,i),200);
    theta_m_2_ds(:,i) = downsample(theta_m_2_out(:,i),200);
    theta_m_3_ds(:,i) = downsample(theta_m_3_out(:,i),200);
    Te_1_ds(:,i) = downsample(Te_1_out(:,i),200);
    Te_2_ds(:,i) = downsample(Te_2_out(:,i),200);
    Te_3_ds(:,i) = downsample(Te_3_out(:,i),200);
    delv2_cont_ds(:,i) = downsample(delv2_cont_out(:,i),200);
    delv3_cont_ds(:,i) = downsample(delv3_cont_out(:,i),200);
end

% Plot mechanical speeds (full)
figure
plot(t_ds,wm_1_ds(:,1)*30/pi)
hold on
plot(t_ds,wm_2_ds(:,1)*30/pi)
plot(t_ds,wm_3_ds(:,1)*30/pi)
plot(t_ds,R_wm_1_ds*30/pi)
plot(t_ds,R_wm_2_ds*30/pi)
plot(t_ds,R_wm_3_ds*30/pi)
xlabel("$t$ [s]","Interpreter","latex")
ylabel("$\omega_{rm}$ [rpm]","Interpreter","latex")
legend(["Aux.\ Converter ($N_2/N_1=5$), $\omega_{rm,1}$", ...
    "Aux.\ Converter ($N_2/N_1=5$), $\omega_{rm,2}$", ...
    "Aux.\ Converter ($N_2/N_1=5$), $\omega_{rm,3}$", ...
    "Resistor, $\omega_{rm,1}$","Resistor, $\omega_{rm,2}$", ...
    "Resistor, $\omega_{rm,3}$"],"Interpreter","latex","Location","southeast")
matlab2tikz('filename','fullSpd.tex','width','6in','height','2in','standalone',true)

% Plot mechanical speeds (zoomed)
figure
plot(t_ds,wm_1_ds(:,1)*30/pi)
hold on
plot(t_ds,wm_2_ds(:,1)*30/pi)
plot(t_ds,wm_3_ds(:,1)*30/pi)
plot(t_ds,R_wm_1_ds*30/pi)
plot(t_ds,R_wm_2_ds*30/pi)
plot(t_ds,R_wm_3_ds*30/pi)
xlim([1.9 6])
xlabel("$t$ [s]","Interpreter","latex")
ylabel("$\omega_{rm}$ [rpm]","Interpreter","latex")
legend(["Aux.\ Converter ($N_2/N_1=5$), $\omega_{rm,1}$", ...
    "Aux.\ Converter ($N_2/N_1=5$), $\omega_{rm,2}$", ...
    "Aux.\ Converter ($N_2/N_1=5$), $\omega_{rm,3}$", ...
    "Resistor, $\omega_{rm,1}$","Resistor, $\omega_{rm,2}$", ...
```

```
    "Resistor, $\omega_{rm,3}$"],"Interpreter","latex","Location","southeast")
matlab2tikz('filename','zoomSpd.tex','width','6in','height','2in','standalone',true)

% Plot electromagnetic torques
figure
plot(t_ds,Te_1_ds(:,1)*30/pi)
hold on
plot(t_ds,R_Te_1_ds*30/pi)
plot(t_ds,Te_2_ds(:,1)*30/pi)
plot(t_ds,R_Te_2_ds*30/pi)
plot(t_ds,Te_3_ds(:,1)*30/pi)
plot(t_ds,R_Te_3_ds*30/pi)
xlim([0.5 6])
xlabel("$t$ [s]","Interpreter","latex")
ylabel("$T_e$ [N$\cdot$m]","Interpreter","latex")
legend(["Aux.\ Converter ($N_2/N_1=5$), $T_{e,1}$","Resistor, $T_{e,1}$",...
    "Aux.\ Converter ($N_2/N_1=5$), $T_{e,2}$","Resistor, $T_{e,2}$",...
    "Aux.\ Converter ($N_2/N_1=5$), $T_{e,3}$","Resistor, $T_{e,3}$"],...
    "Interpreter","latex","Location","southeast","NumColumns",2)
matlab2tikz('filename','Te.tex','width','6in','height','2in','standalone',true)

% Plot position differences on motor 2
figure
plot(t_ds,(theta_m_2_ds-theta_m_1_ds)*180/pi)
hold on
plot(t_ds,(R_theta_m_2_ds-R_theta_m_1_ds)*180/pi)
legend(["Aux.\ Converter ($N_2/N_1=5$)","$N_2/N_1=3$",...
    "$N_2/N_1=2$, full bus","$N_2/N_1=2$, halved bus","Resistor"],...
    "Interpreter","latex")
xlabel("$t$ [s]","Interpreter","latex")
ylabel("$\delta\theta_{rm,2}$ [deg]","Interpreter","latex")
xlim([2 6])
% title("Position Synchronization for 20% Torque Differential")
matlab2tikz('filename','theta2.tex','width','3in','height','2in','standalone',true)

% Plot position differences on motor 3
figure
plot(t_ds,(theta_m_3_ds-theta_m_1_ds)*180/pi)
hold on
plot(t_ds,(R_theta_m_3_ds-R_theta_m_1_ds)*180/pi)
legend(["Aux.\ Converter ($N_2/N_1=5$)","$N_2/N_1=3$",...
    "$N_2/N_1=2$, full bus","$N_2/N_1=2$, halved bus","Resistor"],...
    "Interpreter","latex")
xlabel("$t$ [s]","Interpreter","latex")
ylabel("$\delta\theta_{rm,3}$ [deg]","Interpreter","latex")
xlim([2 6])
% title("Position Synchronization for 30% Torque Differential")
matlab2tikz('filename','theta3.tex','width','3in','height','2in','standalone',true)

% Plot speed differences on motor 2
figure
plot(t_ds,(wm_2_ds-wm_1_ds)*30/pi)
hold on
plot(t_ds,(R_wm_2_ds-R_wm_1_ds)*30/pi)
legend(["Aux.\ Converter ($N_2/N_1=5$)","$N_2/N_1=3$",...
    "$N_2/N_1=2$, full bus","$N_2/N_1=2$, halved bus","Resistor"],...
    "Interpreter","latex")
xlabel("$t$ [s]","Interpreter","latex")
ylabel("$\delta\omega_{rm,2}$ [rpm]","Interpreter","latex")
xlim([2 6])
% title("Speed Synchronization for 20% Torque Differential")

% Plot speed differences on motor 3
figure
plot(t_ds,(wm_3_ds-wm_1_ds)*30/pi)
hold on
plot(t_ds,(R_wm_3_ds-R_wm_1_ds)*30/pi)
legend(["Aux.\ Converter ($N_2/N_1=5$)","$N_2/N_1=3$",...
    "$N_2/N_1=2$, full bus","$N_2/N_1=2$, halved bus","Resistor"],...
    "Interpreter","latex")
xlabel("$t$ [s]","Interpreter","latex")
ylabel("$\delta\omega_{rm,3}$ [rpm]","Interpreter","latex")
xlim([2 6])
% title("Speed Synchronization for 30% Torque Differential")

% Plot commands for motor 2
figure
plot(t_ds,delv2_cont_ds/sqrt(2))
ylabel("$\Delta V_2^{\prime *}$ [V]","Interpreter","latex")
ylim([0,20])
legend(["Aux.\ Converter ($N_2/N_1=5$)","$N_2/N_1=3$",...
    "$N_2/N_1=2$, full bus","$N_2/N_1=2$, halved bus"],...
```

```
    "Interpreter","latex","Location","southeast")
xlabel("$t$ [s]","Interpreter","latex")
xlim([2 6])
% title("Commands for 20% Torque Differential")
% matlab2tikz('filename','Vcommand2.tex','width','3in','height','2in','standalone',true)

figure
plot(t_ds,rext2_cont_ds)
ylabel("$r_{e,2}^*$ [$\Omega$]","Interpreter","latex")
xlabel("$t$ [s]","Interpreter","latex")
xlim([2 6])
% matlab2tikz('filename','Rcommand2.tex','width','3in','height','2in','standalone',true)

% Plot commands for motor 3
figure
plot(t_ds,delv3_cont_ds/sqrt(2))
ylabel("$\Delta V_3^{\prime *}$ [V]","Interpreter","latex")
ylim([0,35])
legend(["Aux.\ Converter ($N_2/N_1=5$)","$N_2/N_1=3$",...
    "$N_2/N_1=2$, full bus","$N_2/N_1=2$, halved bus"],...
    "Interpreter","latex","Location","southeast")
xlabel("$t$ [s]","Interpreter","latex")
xlim([2 6])
% title("Commands for 30% Torque Differential")
% matlab2tikz('filename','Vcommand3.tex','width','3in','height','2in','standalone',true)

figure
plot(t_ds,rext3_cont_ds)
ylabel("$r_{e,3}^*$ [$\Omega$]","Interpreter","latex")
xlabel("$t$ [s]","Interpreter","latex")
xlim([2 6])
% matlab2tikz('filename','Rcommand3.tex','width','3in','height','2in','standalone',true)

[delv2_vqe,delv2_vde] = qd2qdMult(delv2_vq_out,delv2_vd_out,theta_e_out);
[delv3_vqe,delv3_vde] = qd2qdMult(delv3_vq_out,delv3_vd_out,theta_e_out);
v2_norm = sqrt(delv2_vqe.^2+delv2_vde.^2);
v3_norm = sqrt(delv3_vqe.^2+delv3_vde.^2);
for i = 1:numel(Nlist)
    v2_norm(:,i) = v2_norm(:,i)*Nlist(i);
    v3_norm(:,i) = v3_norm(:,i)*Nlist(i);
end

% Generate block averages
t_fa = zeros(359,1);
rext2_fa = zeros(359,1);
rext3_fa = zeros(359,1);
delv2_fa = zeros(359,4);
delv3_fa = zeros(359,4);
R_pdc_fa = zeros(359,1);
pdc_fa = zeros(359,4);
R_pmot_fa = zeros(359,1);
pmot_fa = zeros(359,4);
R_ploss_fa = zeros(359,1);
delv_pmot_fa = zeros(359,4);
R_pTm_fa = zeros(359,1);
pTm_fa = zeros(359,4);
for i=1:359
    t_fa(i) = t((i-1)*1667+1);
    rext2_fa(i) = mean(rext2_pwm((i-1)*1667+1:i*1667+1));
    rext3_fa(i) = mean(rext3_pwm((i-1)*1667+1:i*1667+1));
    delv2_fa(i,:) = mean(v2_norm((i-1)*1667+1:i*1667+1,:),1);
    delv3_fa(i,:) = mean(v3_norm((i-1)*1667+1:i*1667+1,:),1);
    R_pdc_fa(i) = mean(R_pdc((i-1)*1667+1:i*1667+1));
    pdc_fa(i,:) = mean(pdc((i-1)*1667+1:i*1667+1,:),1);
    R_pmot_fa(i) = mean(R_pmot((i-1)*1667+1:i*1667+1));
    pmot_fa(i,:) = mean(pmot((i-1)*1667+1:i*1667+1,:),1);
    R_ploss_fa(i) = mean(R_ploss((i-1)*1667+1:i*1667+1));
    delv_pmot_fa(i,:) = mean(delv_pmot((i-1)*1667+1:i*1667+1,:),1);
    R_pTm_fa(i) = mean(R_pTm((i-1)*1667+1:i*1667+1));
    pTm_fa(i,:) = mean(pTm((i-1)*1667+1:i*1667+1,:),1);
end

% Plot actual values for motor 2
figure
plot(t_fa,delv2_fa/sqrt(2))
ylabel("$\overline{\Delta V}_2^\prime$ [V]","Interpreter","latex")
legend(["Aux.\ Converter ($N_2/N_1=5$)","$N_2/N_1=3$",...
    "$N_2/N_1=2$, full bus","$N_2/N_1=2$, halved bus"],...
    "Interpreter","latex","Location","southeast")
xlabel("$t$ [s]","Interpreter","latex")
xlim([2 6])
% title("Fast-Average Voltage/Resistance for 20% Torque Differential")
```

```
matlab2tikz('filename','Vactual2.tex','width','3in','height','2in','standalone',true)

figure
plot(t_fa,rext2_fa)
ylabel("$\bar{r}_{e,2}$ [$\Omega$]","Interpreter","latex")
xlabel("$t$ [s]","Interpreter","latex")
xlim([2 6])
matlab2tikz('filename','Ractual2.tex','width','3in','height','2in','standalone',true)

% Plot actual values for motor 3
figure
plot(t_fa,delv3_fa/sqrt(2))
ylabel("$\overline{\Delta V}_3^\prime$ [V]","Interpreter","latex")
legend(["Aux.\ Converter ($N_2/N_1=5$)","$N_2/N_1=3$",...
    "$N_2/N_1=2$, full bus","$N_2/N_1=2$, halved bus"],...
    "Interpreter","latex","Location","southeast")
xlabel("$t$ [s]","Interpreter","latex")
xlim([2 6])
% title("Fast—Average Voltage/Resistance for 30% Torque Differential")
matlab2tikz('filename','Vactual3.tex','width','3in','height','2in','standalone',true)

figure
plot(t_fa,rext3_fa)
ylabel("$\bar{r}_{e,3}$ [$\Omega$]","Interpreter","latex")
xlabel("$t$ [s]","Interpreter","latex")
xlim([2 6])
matlab2tikz('filename','Ractual3.tex','width','3in','height','2in','standalone',true)

% Plot main converter input power
figure
plot(t_fa,pdc_fa/1000)
hold on
plot(t_fa,R_pdc_fa/1000)
ylabel("$\bar{p}_{in}$ [kW]","Interpreter","latex")
legend(["Aux.\ Converter ($N_2/N_1=5$)","$N_2/N_1=3$",...
    "$N_2/N_1=2$, full bus","$N_2/N_1=2$, halved bus","Resistor"],...
    "Interpreter","latex","Location","southeast")
xlabel("$t$ [s]","Interpreter","latex")
xlim([2 6])
% title("Fast—Average Input Power to Main Converter")
matlab2tikz('filename','pin.tex','width','6in','height','2in','standalone',true)

% Plot motor input power
figure
plot(t_fa,pmot_fa/1000)
hold on
plot(t_fa,R_pmot_fa/1000)
ylabel("$\bar{p}_{mot}$ [kW]","Interpreter","latex")
legend(["Aux.\ Converter ($N_2/N_1=5$)","$N_2/N_1=3$",...
    "$N_2/N_1=2$, full bus","$N_2/N_1=2$, halved bus","Resistor"],...
    "Interpreter","latex","Location","southeast")
xlabel("$t$ [s]","Interpreter","latex")
xlim([2 6])
% title("Fast—Average Power Flow to Motors")
matlab2tikz('filename','pmot.tex','width','6in','height','2in','standalone',true)

% Plot power removed from line (loss)
figure
plot(t_fa,delv_pmot_fa/1000)
hold on
plot(t_fa,R_ploss_fa/1000)
ylabel("$\bar{p}_{loss}$ [kW]","Interpreter","latex")
legend(["Aux.\ Converter ($N_2/N_1=5$)","$N_2/N_1=3$",...
    "$N_2/N_1=2$, full bus","$N_2/N_1=2$, halved bus","Resistor"],...
    "Interpreter","latex","Location","southeast")
xlabel("$t$ [s]","Interpreter","latex")
xlim([2 6])
% title("Fast—Average Power Removed from Lines")
matlab2tikz('filename','ploss.tex','width','6in','height','2in','standalone',true)

% Plot mech. torque power
figure
plot(t_fa,pTm_fa/1000)
hold on
plot(t_fa,R_pTm_fa/1000)
ylabel("$\bar{p}_{Tm}$ [kW]","Interpreter","latex")
legend(["Aux.\ Converter ($N_2/N_1=5$)","$N_2/N_1=3$",...
    "$N_2/N_1=2$, full bus","$N_2/N_1=2$, halved bus","Resistor"],...
    "Interpreter","latex","Location","southeast")
xlabel("$t$ [s]","Interpreter","latex")
xlim([2 6])
% title("Fast—Average Mechanical Torque Power")
```

```
463 matlab2tikz('filename','ptm.tex','width','6in','height','2in','standalone',true)
464
465 %% Functions for handling multiple runs at once
466 function [x_a, x_b, x_c] = statabcMult(x_q, x_d)
467
468 x_a = x_q;
469 x_b = x_q.*-0.5 - x_d.*sqrt(3)/2;
470 x_c = x_q.*-0.5 + x_d.*sqrt(3)/2;
471 end
472
473 function [y_q, y_d] = qd2qdMult(x_q, x_d, theta)
474
475 y_q = x_q.*cos(theta) - x_d.*sin(theta);
476 y_d = x_q.*sin(theta) + x_d.*cos(theta);
477 end
```