Modelica Implementation of Field-oriented Controlled 3-phase Induction Machine Drive

David A. Torrey Ugur S. Selamogullari
Department of Electrical, Computer and Systems Engineering
Rensselaer Polytechnic Institute
Troy, NY 12180 USA
Email: torred@rpi.edu, selamu@rpi.edu

Abstract

This paper focuses on the modeling of a cage induction machine drive under direct-field oriented control, also referred to as flux-vector control. The interest is to create a behavioral model of an induction machine drive under field orientation for sytem simulations since field-oriented control is now commonplace in commerical adjustable speed drives. First, the 3-phase induction machine model is developed. Then, the field orientation requirements are applied to this model and a voltage source inverter is used to emulate a controlled current source. Rotor field orientation is used because of fewer limitations than other field-orientation approaches. The inverter is assumed to provide the desired phase currents instantenously and ripple free at some efficiency. Finally, these three components of the overall drive sytem, machine model, field orientation and inverter power supply, are combined together in a block using the Modelica language and simultaneously solved using the Dymola user interface.

1 Introduction

As a mature technology the induction machine enjoys use in many established applications and is frequently the first machine considered for emerging applications. The machine is comprised of a stator and a rotor. The windings on the stator and the rotor are assumed to be sinusoidally distributed in space to simplify the analysis of the machine [5]. The windings in the induction machine are coupled. This coupling is described through the inductance matrix, which describes how current in any one winding contributes to the flux linking the other windings. In a closed form, the matrix equation can be written as

$$\lambda_{abc} = L_{abc}(\theta) i_{abc} \quad , \tag{1}$$

where λ_{abc} and i_{abc} are 1×6 vectors and $L_{abc}(\theta)$ is a 6×6 matrix dependent on rotor position.

The electrical dynamics for the induction machine can be written very succinctly using vector notation as

$$v_{abc} = \frac{d\lambda_{abc}}{dt} + R_{abc} i_{abc} \quad . \tag{2}$$

The electromagnetic torque is

$$\tau_{em} = \frac{1}{2} i^T \frac{dL(\theta)}{d\theta} i \quad , \tag{3}$$

and the mechanical dynamics are

$$H \frac{d\omega}{dt} = \tau_{em} - \tau_l \quad , \tag{4}$$

where the load torque τ_l includes windage and friction in addition to the shaft load. The moment of inertia (H) is assumed to include the inertia of the induction machine and whatever is connected to the induction machine through its shaft.

Taken together Eqs. 1 through 4 summarize the electromechanical dynamics of the induction machine. This description, however is inconvenient for studying dynamics and control for two reasons:

1. The order of the system is large.
2. The dependence on θ gives rise to a time-varying model.

To obtain a much simpler induction machine model, two power invariant transformations are used. The $\alpha\beta$ transformation converts a balanced three-phase machine into an equivalent balanced two-phase machine.

This is valuable because in a three-phase machine each phase couples into the other phase. A two-phase machine, on the other hand, has phase windings that do not couple because the axes of the magnetic fields are orthogonal (Fig. 1). In addition, it reduces the machine from six windings to four windings.

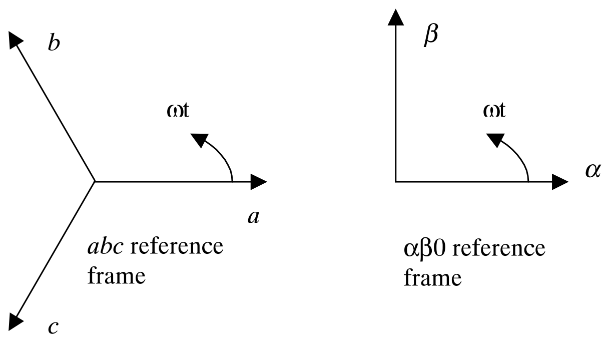

Figure 1: Space vectors for the *abc* reference frame and the αβ0 reference frame.

If the phase a and phase α axes are coincident, N_3 is the number of turns of the three-phase winding and N_2 is the number of turns for the two-phase winding, then resolving the three mmfs of the *abc* frame along the α and β axes and equating the three-phase quantities gives

$$N_2 i_\alpha = N_3 i_a + N_3 i_b \cos\left(\tfrac{2\pi}{3}\right) + N_3 i_c \cos\left(\tfrac{4\pi}{3}\right), \quad (5)$$
$$N_2 i_\beta = N_3 i_b \sin\left(\tfrac{2\pi}{3}\right) + N_3 i_c \sin\left(\tfrac{4\pi}{3}\right). \quad (6)$$

For completeness, a third variable which is independent of i_α and i_β is needed:

$$N_2 i_0 = kN_3 i_a + kN_3 i_b + kN_3 i_c \quad . \quad (7)$$

These relationships can be summarized in vector form as

$$\begin{bmatrix} i_\alpha \\ i_\beta \\ i_0 \end{bmatrix} = \overbrace{\frac{N_3}{N_2}\begin{bmatrix} 1 & -1/2 & -1/2 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \\ k & k & k \end{bmatrix}}^{T} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix}. \quad (8)$$

In order to have invariance of power, $T = T^{-T}$ must be satisfied, and this is satisfied if $N_3/N_2 = \sqrt{2/3}$ and $k = 1/\sqrt{2}$ [7], giving

$$T = \sqrt{\frac{2}{3}}\begin{bmatrix} 1 & -1/2 & -1/2 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \\ 1/\sqrt{2} & 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \quad . \quad (9)$$

Since 0 components do not couple to either the α or β phases and do not contribute to torque production, it is best not to include them in the model. As a result, the model order is reduced from six states to four states. The T matrix becomes T_{23} for converting *abc* quantities to αβ quantities and becomes T_{32} for the inverse transformation:

$$T_{23} = \sqrt{\frac{2}{3}}\begin{bmatrix} 1 & -1/2 & -1/2 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \end{bmatrix} \quad ; \quad (10)$$
$$T_{32} = T_{23}^T \quad . \quad (11)$$

In a closed format, the electrical dynamics become

$$v_{\alpha\beta} = \frac{d\lambda_{\alpha\beta}}{dt} + R_{\alpha\beta} i_{\alpha\beta} \quad . \quad (12)$$

The second transformation is another power-invariant transformation that is tied to the rotating magnetic fields in the airgap of the machine. This *dq* transformation eliminates the rotor position from the machine dynamics by projecting the dynamics onto a reference frame that moves with the airgap magnetic field. Fig. 2 shows an arbitrary vector $\vec{a}$ decomposed into reference frames where one frame is displaced from the other by an angle ϕ. Each reference frame is denoted by a direct axis and a quadrature axis; the direct and quadrature axes are orthogonal. It can be shown that

$$\begin{bmatrix} a_{d_2} \\ a_{q_2} \end{bmatrix} = \begin{bmatrix} \cos(\phi) & \sin(\phi) \\ -\sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} a_{d_1} \\ a_{q_1} \end{bmatrix}. \quad (13)$$

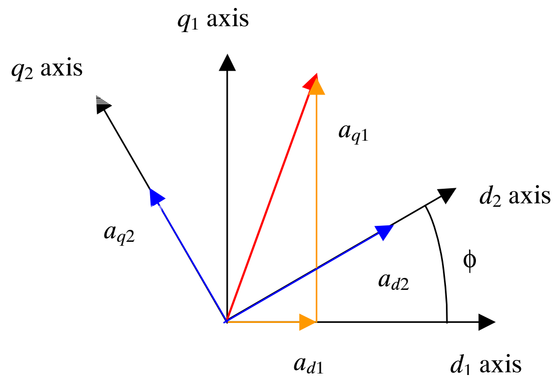

Figure 2: The vector $\vec{a}$ decomposed into two reference frames, with angular displacement ϕ between them.

Fig. 3 shows the relationship among the three coordinate systems, where superscript ^{s} and ^{r} indicate stator and rotor frames, respectively. Accordingly, the αβ dynamics of the stator and rotor are transformed to *dq* reference frame through an angle $P\phi$ for the stator and

$P(\phi-\theta)$ for the rotor quantities (Fig. 3). It follows that

$$\begin{bmatrix} \lambda_{sd} \\ \lambda_{sq} \\ \lambda_{rd} \\ \lambda_{rq} \end{bmatrix} = \begin{bmatrix} e^{P\phi J} & 0 \\ 0 & e^{P(\phi-\theta)J} \end{bmatrix} \begin{bmatrix} \lambda_{s\alpha} \\ \lambda_{s\beta} \\ \lambda_{r\alpha} \\ \lambda_{r\beta} \end{bmatrix} , \quad (14)$$

where

$$J = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} ; \quad (15)$$

$$e^{J\phi} = \begin{bmatrix} \cos(\phi) & \sin(\phi) \\ -\sin(\phi) & \cos(\phi) \end{bmatrix} . \quad (16)$$

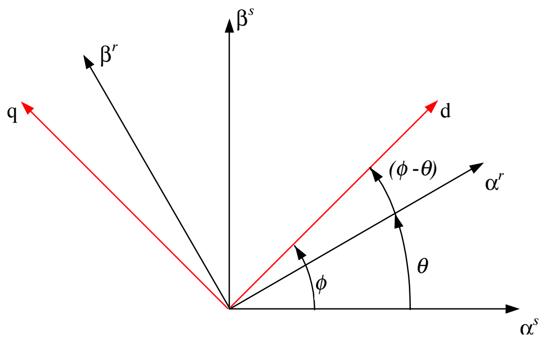

Figure 3: The relationship among the stator $\alpha\beta$ axes, the rotor $\alpha\beta$ axes, and the axes of the dq reference frame.

The machine dynamics become

$$v_{dq} = \frac{\lambda_{dq}}{dt} + R_{dq} i_{dq} - \begin{bmatrix} \omega_s J & 0 \\ 0 & \omega_{sl} J \end{bmatrix} \lambda_{dq} , \quad (17)$$

where

$$\begin{aligned} \lambda_{dq} &= \begin{bmatrix} L_S I & MI \\ MI & L_R I \end{bmatrix} i_{dq} ; \quad (18) \\ L_S &= L_s + L_{ss} , \\ L_R &= L_r + L_{rr} , \\ M &= \tfrac{3}{2} L_{sr} . \end{aligned}$$

ω_s is the synchronous angular velocity of the air-gap magnetic field, ω_{sl} is the slip frequency, and P is the number of pole pairs.

The torque equation is

$$\tau_m = \frac{3PL_{sr}}{2}(i_{sq} i_{rd} - i_{sd} i_{rq}) = PM(i_{sq} i_{rd} - i_{sd} i_{rq}) . \quad (19)$$

2 Field Orientation

Field oriented control is a technique that structures the control of an induction-machine to be entirely parallel to that of a separately excited dc machine. That is, the field flux is oriented to be orthogonal to the torque-producing current. There are three commonly discussed versions of field orientation: rotor, stator and airgap. In each, the torque is given by vector product between flux and current. The flux involved is tied to the type of orientation, for example rotor orientation uses rotor fluxes. In the direct method, the airgap flux is measured directly by Hall sensors to determine the magnitude and orientation of the rotor flux vector, while the indirect field orientation is based on calculating the slip speed required for proper field orientation, and imposition of this speed on the motor [1, 2, 3, 4].

In direct field orientation, the orientation of the rotor flux is determined as follows:

1. The currents $i_{s\alpha}$ and $i_{s\beta}$ are calculated from the measured stator currents.
2. The fluxes $\lambda_{r\alpha}$ and $\lambda_{r\beta}$ are calculated from the measured airgap flux and the stator currents:

$$\begin{aligned} \lambda_{r\alpha} &= \frac{L_R}{M}\lambda_{m\alpha} - (L_R - M) i_{s\alpha} ; \quad (20) \\ \lambda_{r\beta} &= \frac{L_R}{M}\lambda_{m\beta} - (L_R - M) i_{s\beta} . \quad (21) \end{aligned}$$

3. The magnitude and orientation of the rotor flux is determined using the rectangular to polar coordinate transformation:

$$\begin{aligned} |\lambda_r| &= \sqrt{\lambda_{r\alpha}^2 + \lambda_{r\beta}^2} , \quad (22) \\ \phi &= \tan^{-1}\frac{\lambda_{r\beta}}{\lambda_{r\alpha}} . \quad (23) \end{aligned}$$

Under field-orientation, we are forcing the induction machine to maintain orthogonality between appropriate flux and current through active control. The commands for flux and torque are generated by a higher-order controller. The block diagram of the field oriented-control of an induction machine is given in Fig. 4. Based on commanded flux and torque, the desired values for i_{sd}^* and i_{sq}^* are generated. The outputs of the flux and torque calculators are used to close the flux and torque feedback loops. By knowing the rotor position, the corresponding values for $i_{s\alpha}^*$ and $i_{s\beta}^*$ are determined using the rotary transformation:

$$\begin{bmatrix} i_{s\alpha}^s \\ i_{s\beta}^s \end{bmatrix} = e^{-J\phi} \begin{bmatrix} i_{sd} \\ i_{sq} \end{bmatrix} , \quad (24)$$

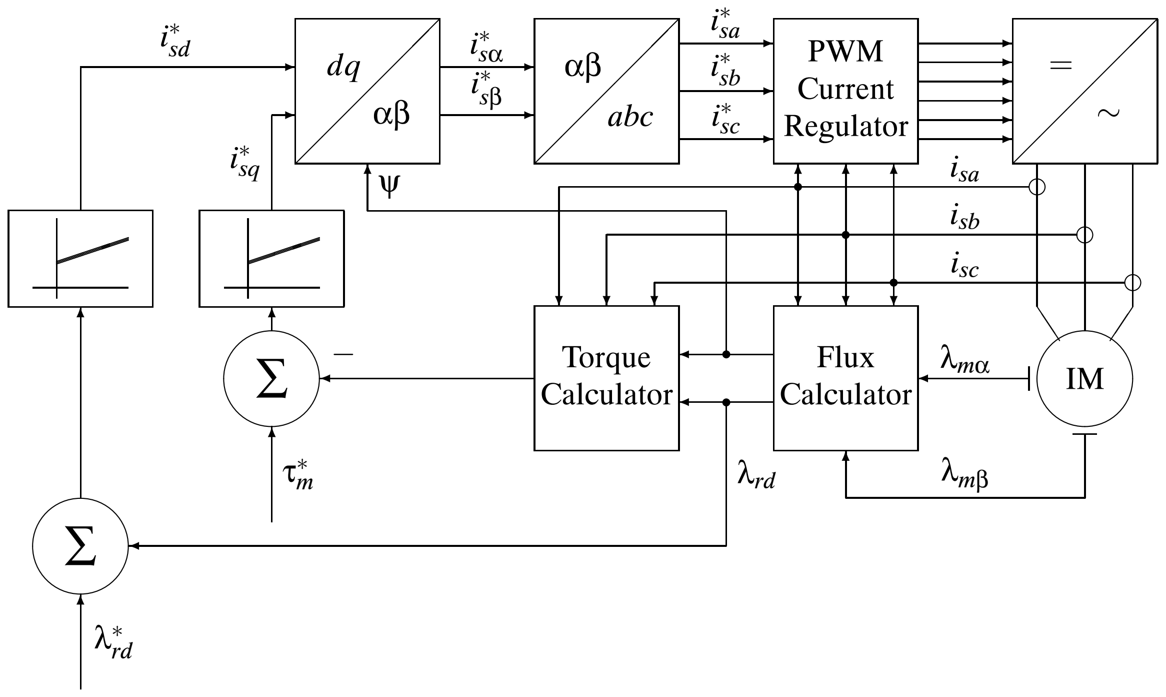

Figure 4: A general block diagram of how field oriented control is implemented.

where ϕ is the electrical angle that comes out of the flux calculator shown in Fig 4. The excitation angle is determined by simulating the induction motor in the $\alpha\beta$ reference frame. In our model, this is an easy task since the simulation gives us $\lambda_{r\alpha}$ and $\lambda_{r\beta}$ directly. Once $i_{s\alpha}$ and $i_{s\beta}$ are determined, they can be converted into the desired phase currents i^*_{sa}, i^*_{sb} and i^*_{sc} using the $\alpha\beta$ transformation. Because stator currents are imposed on the induction motor, phase voltages are no longer prescribed. Instead, the phase voltages reflect the self-consistent resolution of the induction motor model and the imposed currents. Accordingly, the number of states within the induction motor model is reduced by two. The rotor dynamics on the $\alpha\beta$ axes can be written as

$$\frac{d\lambda^s_{r\alpha}}{dt} = -\frac{1}{\tau_r}\lambda^s_{r\alpha} + \frac{M}{\tau_r}i^s_{s\alpha} - P\omega_r\lambda^s_{r\beta} \quad , \tag{25}$$

$$\frac{d\lambda^s_{r\beta}}{dt} = -\frac{1}{\tau_r}\lambda^s_{r\beta} + \frac{M}{\tau_r}i^s_{s\beta} + P\omega_r\lambda^s_{r\alpha} \quad , \tag{26}$$

where $\tau_r = L_R/R_r$.

The stator voltages are given by

$$v^s_{s,\alpha\beta} = R_s i_{s,\alpha\beta} + \frac{d\lambda_{s,\alpha\beta}}{dt} \quad , \tag{27}$$

where

$$\lambda^s_{s,\alpha\beta} = \sigma L_S i_{s,\alpha\beta} + \frac{M}{L_R}\lambda_{r,\alpha\beta} \quad , \tag{28}$$

and

$$\sigma = 1 - \frac{M^2}{L_S L_R} \quad . \tag{29}$$

Stator three-phase voltages and currents can be calculated using the T_{23} transformation matrix:

$$v_{s,abc} = T_{23} v^s_{s,\alpha\beta} \quad ; \tag{30}$$

$$i_{s,abc} = T_{23} i^s_{s,\alpha\beta} \quad . \tag{31}$$

It is common to use a voltage-source inverter to feed the induction motor with closed-loop current control. The inverter is assumed to be instantenous and modeled as a current source. The current values are determined from the feedback loops of flux and torque. Pulse width modulation (PWM) is routinely employed to control the inverter switches [6]. Ideally the phase currents would be without ripple and perfectly track the commanded phase currents, while reality is somewhat different but close enough to the ideal. Thus, the inverter is assumed to be ideal and the current drawn from the DC side of the inverter can be calculated using conservation of instantenous power:

$$i_{dc} = \frac{i^T_{s,abc} v_{s,abc}}{\eta_{inv} v_{dc}} \quad , \tag{32}$$

where η_{inv} is the inverter efficiency.

There are three sections to this model. The first captures the electromechanical dynamics of the induction machine when operating under direct field orientation. These dynamics when applied to the induction machine model prescribe the corresponding stator voltages and currents. The stator voltages and currents in turn dictate the current that must be supplied by the inverter power supply.

As a block diagram, the developed system model is shown in Fig. 5. The pins are provided for inverter connections, inports are used to get the flux and torque commands and a flange is used for the shaft of the machine. This way, the model emulates the reality. The Modelica code for the model is given in Appendix A. Since including the $v_{s,\alpha\beta}$ calculations in the simulation code causes a DAE index problem in the translation stage, the voltages are calculated seperately using the derivative and gain blocks from the Modelica library. Then, the instantenous power is calculated.

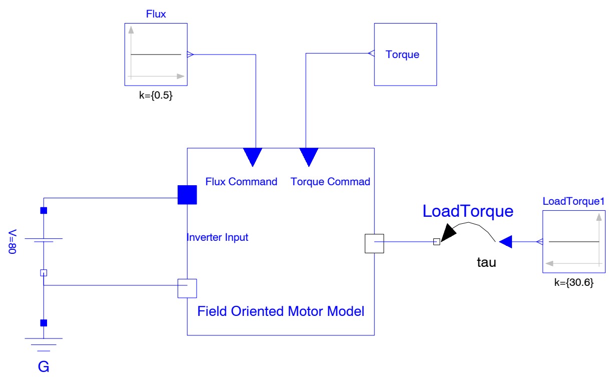

Figure 5: Block diagram of the field-oriented controlled 3-phase induction machine drive.

To illustrate the performance of the direct rotor flux field orientation system, the model is simulated under a commanded torque and flux profile[3]. The load torque is taken as 30.6 Nm: the total inertia of the system is 0.5 kg.m^2. The commanded torque profile is

$$\tau_m = \begin{cases} 135.3 & \text{Nm for } 0 < t \leq 0.5 \text{ sec} \\ 30.6 & \text{Nm for } 0.5 < t \leq 1 \text{ sec} \\ -74.1 & \text{Nm for } 1 < t \leq 1.5 \text{ sec} \\ -135.3 & \text{Nm for } 1.5 < t \leq 2\text{sec} \\ -30.6 & \text{Nm for } t > 2 \text{ sec.} \end{cases} \quad (33)$$

The rotor flux is to be maintained at 0.5 Wb. The programmed torque and flux commands are given in Fig. 6 and Fig. 7, respectively.

Simulation results for the torque, speed and flux of the model are shown Fig. 8 and Fig. 9. The inverter DC side current is plotted in Fig. 10. Comparing the simulation results with the desired torque and flux commands shows that the model follows the commanded torque and flux.

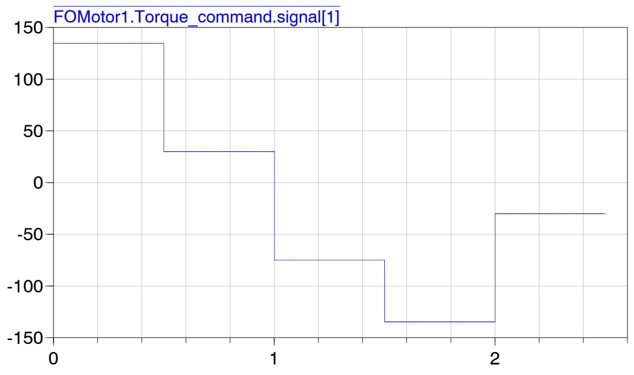

Figure 6: The commanded torque profile.

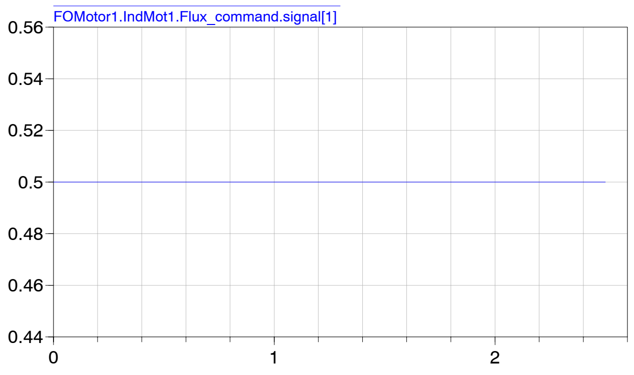

Figure 7: The commanded flux.

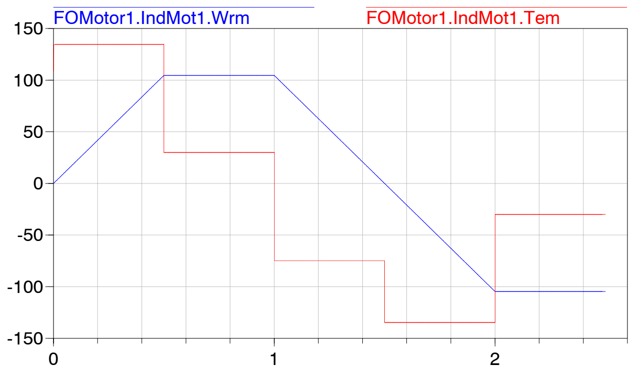

Figure 8: The torque and speed response of the induction machine.

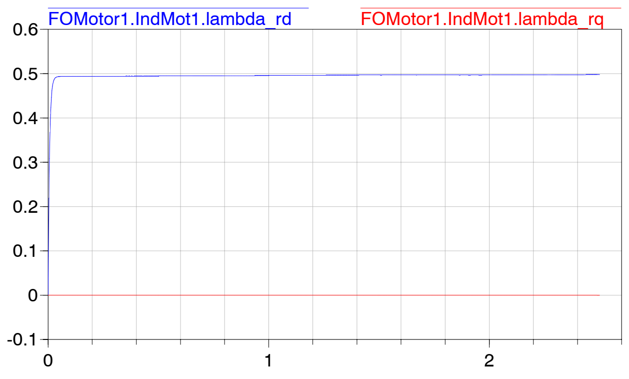

Figure 9: The simulated rotor fluxes.

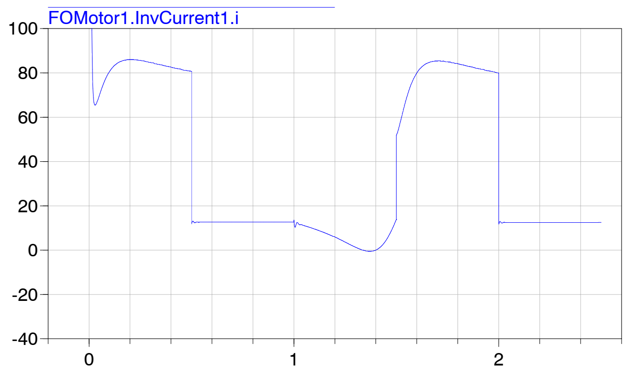

Figure 10: The inverter DC side current.

3 Conclusion

The underlying motivation for this study is the construction of a field-oriented controlled 3-phase induction machine drive model for system simulations. The final block diagram (Fig. 5) is comprised of three components: an induction machine model, field orientation requirements and an inverter power supply that provides the desired phase currents under field orientation. The inverter is assumed ideal and loaded consistent with the induction machine drive sytem. Since the physical phase currents are needed to obtain the inverter DC side current, the induction machine is simulated in $\alpha\beta$ reference frame under rotor direct field orientation. This is required to determine the orientation of the rotor flux. Pins, inports and a flange are used to provide the connection points to the user. The model is simulated under a programmed flux and torque profile and results are given. The Dymola simulation tool is used to solve the simultaneous resolution of three sections.

References

[1] F. Blaschke, *Das Ver fahren der Feldorientierung zur Regelung der Drehfelmachine (The method of field orientation for control of three phase machines)* Ph.D. Dissertation, TU Braunschweig, 1973.

[2] F. Blaschke, *The principle of field orientation as applied to the new transvektor closed-loop control system rotating-field machines*, Siemens Review, Vol. 34, pp. 217-220, May 1972.

[3] A. M. Trzynadlowski, *The Field Orientation Principle in Control of Induction Motors*, Kluwer, 1994.

[4] D. M. Novotny and T. A. Lipo, *Vector Control and Dynamcis of AC Drives*, Oxford University Press, 1997.

[5] A. E. Fitzgerald, C. Kinglesy, Jr., and S. D. Umans, *Electric Machinery*, 5th ed., McGraw-Hill, 1990.

[6] B. K. Bose, ed., *Power Electronics and Variable Frequency Drives*, IEEE Press, 1997, Chapter 4.

[7] N. N. Hancock, *Matrix Analysis of Electric Machinery*,2nd ed., Pergamon Press,1974

[8] Dymola User's Manual

[9] Modelica Web page (www.modelica.org)

A Modelica Code

```
package FieldOriented

class Tork
    Real T ;
    Modelica.Blocks.Interfaces.OutPort Torque ;
equation
    T = if (time <= 0.5) then (135.3) else if (time <= 1 and time > 0.5) then
    30.6 else if (time > 1 and time <= 1.5) then -74.1 else if (time > 1.5
    and time <= 2) then -135.3 else if time > 2 then -30.6 else 0 ;
    Torque.signal[1] = T ;
end Tork ;

partial class InvCurrent "Source for constant current"
    extends Modelica.Electrical.Analog.Interfaces.OnePort ;
end InvCurrent ;

class IndMot
    parameter Real Rs=0.294 "stator resistance(abc, dq frames)" ;
    parameter Real Rr=0.156 "rotor resistance (abc, dq frames)" ;
    parameter Real Lsl=0.00139 "abc frame stator leakage inductance" ;
    parameter Real Lrl=0.00074 "abc frame rotor lekage inductance" ;
    parameter Real Lsr=0.041 "abc frame mutual inductance" ;
    parameter Real P=3 "number of pole pairs" ;
    parameter Real H=0.5 "inertia of rotor" ;
    parameter Real f=60 "applied source frequency" ;
    Real M "dq frame mutual inductance" ;
    Real LS "dq frame stator inductance" ;
    Real LR "dq frame rotor inductance" ;
    Real D "leakage factor" ;
    Real Tem "electromechanical torque" ;
    Real Wrm(start=0) "Motor mechanical speed" ;
    Real theta "Rotor position angle" ;
    Real Isd "d axis stator current" ;
    Real Isq "q axis stator current" ;
    Real Is_alpha "alpha axis stator current" ;
    Real Is_beta "beta axis stator current" ;
    Real Isa "stator phase a current" ;
    Real Isb "stator phase b current" ;
    Real Isc "stator phase a current" ;
    Real lambda_ralpha(start=1e-3) "alpha axis rotor flux" ;
    Real lambda_rbeta(start=1e-3) "beta axis rotor flux" ;
    Real lambda_rd "d axis rotor flux" ;
    Real lambda_rq "q axis rotor flux" ;
    Real lambda_salpha "alpaha axis stator flux" ;
    Real lambda_sbeta "beta axis stator flux" ;
    Real Tem_fo "Torque under field orientation used for closed loop control" ;
    Real m_flux "magnitude of the rotr flux" ;
    Real p_flux "angle of the rotor flux" ;
    Real Tref "Reference Torque" ;
    Real Lref "Reference Flux" ;
    Real Terror "Torque error" ;
    Real Ferror "Flux error" ;
    Real errort(start=0) "integral of torque error" ;
    Real errorf(start=0) "integral of flux errror" ;
    parameter Real Kif=500 "PI controller integral gain for Flux" ;
    parameter Real Kpf=1000 "PI controller proportional gain for Flux" ;
    parameter Real Kit=500 "PI controller integral gain for Torque" ;
    parameter Real Kpt=1000 "PI controller proportional gain for Torque" ;
    Modelica.Blocks.Interfaces.OutPort Lambda_salpha ;
    Modelica.Blocks.Interfaces.OutPort Lambda_sbeta ;
    Modelica.Blocks.Interfaces.OutPort I_salpha ;
    Modelica.Blocks.Interfaces.OutPort I_sbeta "OutPorts used above are
    used to calculate the Vs_alpha and Vs_beta since including them
    into the code brings DAE index problem" ;
    Modelica.Blocks.Interfaces.InPort Flux_command ;
    Modelica.Blocks.Interfaces.InPort Torque_command ;
    Modelica.Mechanics.Rotational.Interfaces.Flange_b shaft ;
equation
    Lambda_salpha.signal[1] = lambda_salpha ;
    Lambda_sbeta.signal[1] = lambda_sbeta ;
    I_salpha.signal[1] = Is_alpha ;
    I_sbeta.signal[1] = Is_beta ;
    M = 3/2*Lsr ;
    LS = Lsl + M ;
    LR = Lrl + M ;
    D = (LS*LR - (M*M)) ;

    Tref = Torque_command.signal[1] ;
    Lref = Flux_command.signal[1] ;

    Terror = Tref - Tem_fo ;
    Ferror = Lref - m_flux ;

    der(errort) = Terror ;
    der(errorf) = Ferror ;

    Isd = Kpf*Ferror + Kif*errorf ;
    Isq = Kpt*Terror + Kit*errort ;
    Is_alpha = Isd*cos(p_flux) - Isq*sin(p_flux) ;
    Is_beta = Isd*sin(p_flux) + Isq*cos(p_flux) ;

    der(lambda_ralpha) = -Rr/LR*lambda_ralpha + M*Rr/LR*Is_alpha - P*Wrm*lambda_rbeta ;
    der(lambda_rbeta) = -Rr/LR*lambda_rbeta + M*Rr/LR*Is_beta + P*Wrm*lambda_ralpha ;
    lambda_rd = lambda_ralpha*cos(p_flux) + lambda_rbeta*sin(p_flux) ;
    lambda_rq = -lambda_ralpha*sin(p_flux) + lambda_rbeta*cos(p_flux) ;
    lambda_salpha = D/(LS*LR)*LS*Is_alpha + M/LR*lambda_ralpha ;
    lambda_sbeta = D/(LS*LR)*LS*Is_beta + M/LR*lambda_rbeta ;

    m_flux = sqrt((lambda_ralpha^2) + (lambda_rbeta^2)) ;
    p_flux = atan2(lambda_rbeta, lambda_ralpha) ;

    der(Wrm) = if Wrm >= 0 then (1/H)*(Tem - shaft.tau) else (1/H)*(Tem + shaft.tau) ;
    der(theta) = Wrm ;
    shaft.phi = theta ;

    Tem = (P*M/LR)*(Isq*lambda_rd - Isd*lambda_rq) ;
    Tem_fo = (P*M/LR)*(Isq*m_flux) ;

    Isa = Is_alpha*sqrt(2/3) ;
    Isb = sqrt(2/3)*(-0.5*Is_alpha + sqrt(3)/2*Is_beta) ;
    Isc = sqrt(2/3)*(-0.5*Is_alpha - sqrt(3)/2*Is_beta) ;

end IndMot ;

class PowerCalculator
block P2toP3
    extends Modelica.Blocks.Interfaces.BlockIcon ;
    parameter Integer n=1 "Dimension of input and output vectors." ;
    Modelica.Blocks.Interfaces.OutPort a(final n=n)"Connector 1 of Real input signals" ;
    Modelica.Blocks.Interfaces.OutPort b(final n=n)"Connector 2 of Real input signals" ;
    Modelica.Blocks.Interfaces.OutPort c(final n=n)"Connector 3 of Real input signals" ;
    Modelica.Blocks.Interfaces.InPort alfa(final n=n)"Connector of Real output signals" ;
    Modelica.Blocks.Interfaces.InPort beta(final n=n) ;
equation
    a.signal[1] = alfa.signal[1]*sqrt(2/3) ;
    b.signal[1] = sqrt(2/3)*(-0.5*alfa.signal[1] + sqrt(3)/2*beta.signal[1]) ;
    c.signal[1] = sqrt(2/3)*(-0.5*alfa.signal[1] - sqrt(3)/2*beta.signal[1]) ;
end P2toP3 ;

    P2toP3 P2toP3_1 ;
    P2toP3 P2toP3_2 ;
    Modelica.Blocks.Math.Gain Gain1 ;
    Modelica.Blocks.Continuous.Derivative Derivative1 ;
    Modelica.Blocks.Math.Add Add1 ;
    Modelica.Blocks.Math.Add3 Power ;
    Modelica.Blocks.Math.Product Product1 ;
    Modelica.Blocks.Math.Gain Gain2 ;
    Modelica.Blocks.Continuous.Derivative Derivative2 ;
    Modelica.Blocks.Math.Add Add2 ;
    Modelica.Blocks.Math.Product Product2 ;
    Modelica.Blocks.Math.Product Product3 ;
    Modelica.Blocks.Interfaces.InPort inPort ;
    Modelica.Blocks.Interfaces.InPort inPort1 ;
    Modelica.Blocks.Interfaces.InPort inPort2 ;
    Modelica.Blocks.Interfaces.InPort inPort3 ;
    Modelica.Blocks.Interfaces.OutPort outPort ;
```

```
equation
    connect(Power.inPort1, Product1.outPort) ;
    connect(Power.inPort3, Product3.outPort) ;
    connect(Derivative2.inPort, inPort3) ;
    connect(Gain1.inPort, inPort) ;
    connect(Add2.inPort2, Derivative2.outPort) ;
    connect(Add2.inPort1, Gain2.outPort) ;
    connect(Add1.inPort1, Gain1.outPort) ;
    connect(Derivative1.inPort, inPort1) ;
    connect(Derivative1.outPort, Add1.inPort2) ;
    connect(P2toP3_1.alfa, inPort) ;
    connect(P2toP3_2.alfa, Add1.outPort) ;
    connect(P2toP3_2.beta, Add2.outPort) ;
    connect(P2toP3_1.a, Product1.inPort1) ;
    connect(P2toP3_2.a, Product1.inPort2) ;
    connect(P2toP3_1.b, Product2.inPort1) ;
    connect(P2toP3_2.b, Product2.inPort2) ;
    connect(P2toP3_1.c, Product3.inPort1) ;
    connect(P2toP3_2.c, Product3.inPort2) ;
    connect(Power.inPort2, Product2.outPort) ;
    connect(inPort2, Gain2.inPort) ;

    connect(P2toP3_1.beta, inPort2) ;
    connect(Power.outPort, outPort) ;
end PowerCalculator;

class FOMotor
    parameter Real Inveff=0.9 ;
    IndMot IndMot1 ;
    Modelica.Electrical.Analog.Interfaces.PositivePin p ;
    Modelica.Electrical.Analog.Interfaces.NegativePin n ;
    Modelica.Blocks.Interfaces.InPort Flux_command ;
    Modelica.Blocks.Interfaces.InPort Torque_command ;
    InvCurrent InvCurrent1 ;
    Modelica.Mechanics.Rotational.Interfaces.Flange_b shaft ;
    PowerCalculator PowerCalculator1 (Gain1.k={IndMot1.Rs}, Gain2.k={IndMot1.Rs}) ;
equation
    connect(IndMot1.Flux_command, Flux_command) ;
    connect(IndMot1.Torque_command, Torque_command) ;
    connect(InvCurrent1.p, p) ;
    connect(InvCurrent1.n, n) ;
    connect(IndMot1.shaft, shaft) ;
    connect(PowerCalculator1.inPort, IndMot1.I_salpha) ;
    connect(PowerCalculator1.inPort1, IndMot1.Lambda_salpha) ;
    InvCurrent1.i = (PowerCalculator1.Power.outPort.signal[1]/(Inveff*InvCurrent1.v)) ;
    connect(PowerCalculator1.inPort2, IndMot1.I_sbeta) ;
    connect(PowerCalculator1.inPort3, IndMot1.Lambda_sbeta) ;
end FOMotor;

end FieldOriented ;
```