Motor Control Blockset™ Release Notes

MATLAB®&SIMULINK®

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Motor Control Blockset™ Release Notes

Contents

R2022a

Induction Motor Parameter Estimation: Compute nominal magnetizing current

You can now use the Motor Control Blockset induction motor parameter estimation example to compute the nominal magnetizing current (I_{d0}) along with the other parameters of a three-phase AC induction motor.

Clarke, Inverse Clarke, Park, and Inverse Park Transform Blocks: HDL code generation support

The following blocks available in the Controls/Math Transforms library now support HDL code generation:

- Clarke Transform
- Inverse Clarke Transform
- Park Transform
- Inverse Park Transform

Quadrature Decoder Block: Perform accurate 16-bit computations

The Quadrature Decoder block, available in the Sensor Decoders library, now performs 16-bit computations with greater precision.

Discrete PID Controller Block: Generate optimized code

The PI Controller block, which replaces the Discrete PI Controller and Discrete PI Controller with anti-windup and reset blocks, enables you to generate optimized code for implementing a discrete PID controller.

This block, available in the Controls/Controllers library (identical to the Discrete PID Controller block available in the Simulink®/Discrete library) now provides more block parameters than the previously available Discrete PI Controller and Discrete PI Controller with anti-windup and reset blocks. For more information, see PI Controller.

Version History

If you use the PI Controller block in releases before R2022a, the software automatically replaces this block with the Discrete PI Controller with anti-windup and reset block and configures it.

If you use the Discrete PI Controller and Discrete PI Controller with anti-windup and reset blocks in R2022a or later releases, the software automatically replaces these blocks with the Discrete PI Controller block and configures it.

Modified Host Serial Receive, Host Serial Transmit, and Host Serial Setup Blocks: Use optimized blocks for serial communication

The modified Host Serial Receive, Host Serial Transmit, and Host Serial Setup blocks enable you to implement optimal serial communication algorithms because these blocks no longer use Java® based code.

These blocks, available in the Protection and Diagnostics library (identical to the Serial Receive, Serial Configuration, and Serial Send blocks available in the Instrument Control Toolbox™ library) now provide new set of block parameters.

Version History

If you use the Host Serial Receive, Host Serial Transmit, and Host Serial Setup blocks in releases before R2022a, the software automatically replaces these blocks with the modified Host Serial Receive, Host Serial Setup, and Host Serial Transmit blocks and configures them.

Compliance with MISRA C guidelines

The blocks and examples provided by the Motor Control Blockset now fully comply with the motor industry software reliability association software development guidelines for C programming language (MISRA C®).

Field Oriented Control Autotuner Block: Support for multitasking execution to improve performance on hardware

The Field Oriented Control Autotuner block now supports multitasking execution and lets you specify a different sample time for the frequency response estimation experiment. Previously, the block supported only the single-tasking mode and performed the experiment, for each loop, at the sample time specified in the **Controller sample time** parameter for that loop. The **Controller sample time** parameter sets the sample time used to calculate the PID controller gains for that loop.

Use the **Experiment sample time** parameter to specify a sample time for the frequency response estimation experiment. For each loop that you tune, the frequency responses are estimated at the sample time specified in the **Experiment sample time** parameter.

R2021b

Induction Motor Parameter Estimation: Determine parameters of AC induction motor from experiments with motor hardware

You can now use the Motor Control Blockset parameter estimation tool to determine mechanical and electrical parameters of a three-phase AC induction motor. Previously, you could use the tool to estimate parameters for a PMSM only.

Run the host model `mcb_acim_param_est_host_read.slx` after deploying the target model `mcb_acim_param_est_f28379D_DRV8305.slx` to execute pre-built instrumented tests on the Texas Instruments™ LAUNCHXL-F28379D controller (with BOOSTXL-DRV8305 inverter) and estimate the following parameters:

- Stator phase resistance (R_s)
- Rotor phase resistance (R_r)
- Magnetizing inductance (L_m)
- Stator leakage inductance (L_{ls})
- Rotor leakage inductance (L_{lr})
- Motor inertia (J)
- Friction constant (F)

For details, see Estimate Induction Motor Parameters Using Recommended Hardware.

PMSM Parameter Estimation: Estimate PMSM parameters using quadrature encoder and custom hardware

You can now use the Motor Control Blockset parameter estimation algorithm independent of the Texas Instruments LAUNCHXL-F28379D and F28069 controllers to develop a tool that runs on custom motor control hardware and estimates the parameters of a three-phase PMSM connected to a quadrature encoder.

The parameter estimation algorithm for custom hardware is available as a MATLAB® project. Run the command `mcb_ParameterEstimationAlgorithmStart` at the command prompt to open the MATLAB project. For details, see Estimate PMSM Parameters Using Custom Hardware.

Algorithm-Export Workflows For Custom Hardware: Reference example

A new reference example implements field-oriented control (FOC) for a three-phase PMSM by using a quadrature encoder sensor and the STMicroelectronics® NUCLEO F302R8 board and X-NUCLEO-IHM07M1 inverter. You can use this example to:

- Verify your hardware setup by using an open-loop control algorithm.
- Calculate the offset of the analog-to-digital converter (ADC).
- Calculate the offset of the quadrature encoder sensor attached to the PMSM.

This example is available as a MATLAB project. Run the command `mcb_FOCAlgorithmExportDemoStart` at the command prompt to open the MATLAB project.

You can also use this project as a reference to develop FOC algorithms for other motor control hardware. For details, see Algorithm-Export Workflows for Custom Hardware.

Sliding Mode Observer: Improve position tracking and automatically estimate observer parameters

The Sliding Mode Observer block, available in the Sensorless Estimators library, performs improved tracking of the PMSM rotor position. In addition, you can now estimate observer parameters automatically. Previously, you could only tune the observer parameters manually.

The previous version of Sliding Mode Observer block, available in the `mcbpositiondecoderlib/Archive` library, uses the older computation technique for tracking the PMSM rotor position. This block does not support automatic estimation of the observer parameters.

Version History

You can use the R2021b version of the Sliding Mode Observer block in R2021b or later releases only.

You can use the previous version of the Sliding Mode Observer block in R2021b or earlier releases only.

Note The previous version of the Sliding Mode Observer block, available in the `mcbpositiondecoderlib/Archive` library, will be removed in a future release. Replace any previous versions of the Sliding Mode Observer block with the R2021b version.

Reference Example Using Field Oriented Control Autotuner Block: Automatically tune current and speed loops running on hardware

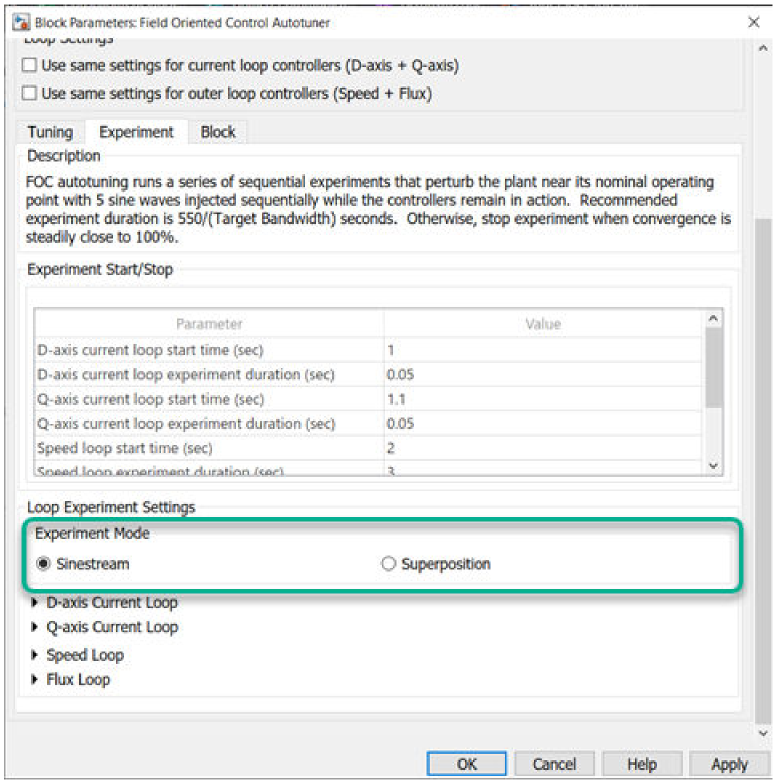
Use the Field Oriented Control Autotuner block to automatically tune the gains of the speed and current loop controllers available in the field-oriented control algorithm that you deploy and run on the Texas Instruments LAUNCHXL-F28379D controller (with BOOSTXL-DRV8305 inverter).

You can simulate the models `mcb_pmsm_foc_autotuner_f28379d.slx` and `mcb_host_autotuner_f28379d.slx` or deploy them to the hardware to compute the PI controller gains for the speed and current loops based on the motor and inverter parameters. For details, see Tune PI Controllers Using Field Oriented Control Autotuner.

Field Oriented Control Autotuner Block: Reduce execution time on hardware by performing frequency response estimation experiment using sinestream signals

The Field Oriented Control Autotuner block tunes PID controllers based on estimated plant frequency responses for each loop. You can now estimate frequency responses of the plants associated with each loop using sinestream input signals. A sinestream signal consists of a series of sinusoidal perturbations applied one after another. When you perform a frequency response estimation experiment using sinestream input signals, the block applies perturbation at each frequency separately. For more information, see Sinestream Input Signals (Simulink Control Design).

Sinestream signals reduce the execution time compared to superposition input signals, but also take longer to estimate the frequency response. Frequency response estimation using sinestream signals is useful when you have limited processing power and you want to reduce the execution time. To do so, on the **Experiment** tab of block parameters, set **Experiment Mode** to **Sinestream**.



Block Parameters: Field Oriented Control Autotuner

Loop Settings

☐ Use same settings for current loop controllers (D-axis + Q-axis)

☐ Use same settings for outer loop controllers (Speed + Flux)

Tuning | Experiment | Block

Description

FOC autotuning runs a series of sequential experiments that perturb the plant near its nominal operating point with 5 sine waves injected sequentially while the controllers remain in action. Recommended experiment duration is 550/(Target Bandwidth) seconds. Otherwise, stop experiment when convergence is steadily close to 100%.

Experiment Start/Stop

Parameter	Value
D-axis current loop start time (sec)	1
D-axis current loop experiment duration (sec)	0.05
Q-axis current loop start time (sec)	1.1
Q-axis current loop experiment duration (sec)	0.05
Speed loop start time (sec)	2
Speed loop experiment duration (sec)	3

Loop Experiment Settings

Experiment Mode

[✓ Sinestream] Superposition

▸ D-axis Current Loop

▸ Q-axis Current Loop

▸ Speed Loop

▸ Flux Loop

For more information about deploying the Field Oriented Control Autotuner block, see Tune PI Controllers Using Field Oriented Control Autotuner Block on Real-Time Systems.

Initial Rotor Position Estimation: Reference example

Use a new reference example to determine the initial rotor position of a stationary interior PMSM that has a high saliency ratio ($L_q > L_d$). The example uses the pulsating high-frequency method to inject a high-frequency voltage into the motor and performs numerical analysis of the resulting stator current signals to compute the initial position of the stationary rotor. In addition, it uses the dual pulse injection method to determine if the computed position needs pi-compensation.

Simulate the model `mcb_ipmsm_pos_est_f28379d.slx` or deploy them to the hardware to compute the initial rotor position of an interior PMSM with a high saliency ratio. For details, see Estimate Initial Rotor Position Using Pulsating High-Frequency and Dual-Pulse Methods.

PWM Reference Generator Block: Use modulation strategies that reduce switching losses

The PWM Reference Generator block, which replaces the Space Vector Generator block, enables you to perform sinusoidal PWM (SPWM) and space vector modulation (SVM) as well as select pulse-width modulation (PWM) methods that reduce switching losses:

- 60 DPWM — 60 degree discontinuous PWM

- 60 DPWM (+30 degree shift) — +30 degree shift from 60 DPWM
- 60 DPWM (-30 degree shift) — -30 degree shift from 60 DPWM
- 30 DPWM — 30 degree discontinuous PWM
- 120 DPWM — Positive DC component
- 120 DPWM — Negative DC component

This block, available in the `Controls/Math Transforms` library, also enables you to select either *α*- and *β*-axis voltages or *abc* phase voltages as the block inputs.

Version History

If you use the PWM Reference Generator block in releases before R2021b, the software treats the block as a Space Vector Generator block that supports only SVM.

If you use the older version of this block in R2021b or later releases, the software configures the block to use the V_{α} and V_{β} inputs and SVM by default. You can change the block inputs and the modulation method.

Speed Measurement Block: Support for uint16, uint32, and uint64 data types

When you use the Speed Measurement block, available in the Sensor Decoders library, you can now specify the data type for scaling and processing the position signal input. Available data types are `uint16`, `uint32`, and `uint64`. Select the data type under **Position scaling datatype** in the block parameter dialog box.

Version History

If you use the current version of this block in releases before R2021b, the block supports only `uint32`.

If you use the older version of this block in R2021b or later releases, the software configures the block to use `uint32` by default. You can change the data type to `uint16` or `uint64`.

R2021a

Flux Observer: Implement sensorless control of induction motors

The Flux Observer block now works with induction motors.

This block, available under the Sensorless Estimators library, uses α- and β-axis voltages and currents to compute electrical position, magnetic flux, and electrical torque of either a PMSM or an induction motor. You can also use this block to implement field-oriented control of a brushless DC motor.

The block now uses an internal high-pass filter to remove noise, which results in a more accurate output.

Version History

When you use this version of the block with the previous Motor Control Blockset releases, the block supports only a PMSM.

When you use the older version of this block with the current Motor Control Blockset release, Simulink configures the block to work with a PMSM by default. However, you can reconfigure the block to work with an induction motor.

IIR Filter: Improve sensorless control performance using high-pass filter

Use the IIR Filter block to implement a discrete infinite impulse response (IIR) high-pass filter.

This block, available under the Signal Management library, can now function as either a low-pass or high-pass filter. You can select the discrete step size to determine and set a cutoff frequency (Hz) for the filter.

When you use this version of the block with the previous Motor Control Blockset releases, the block always runs as a low-pass filter.

When you use the older version of this block with the current Motor Control Blockset release, the block behaves as a low-pass filter by default, but you can reconfigure it as a high-pass filter.

Motor Parameter Estimation: Estimate PMSM parameters without position sensors using F28069M controller

In addition to the LAUNCHXL-F28379D controller (with BOOSTXL-DRV8305 inverter), you can now use the Motor Control Blockset parameter estimation tool with the F28069 controller (with DRV8312-69M-KIT inverter) to determine PMSM parameters by using the sensorless flux observer.

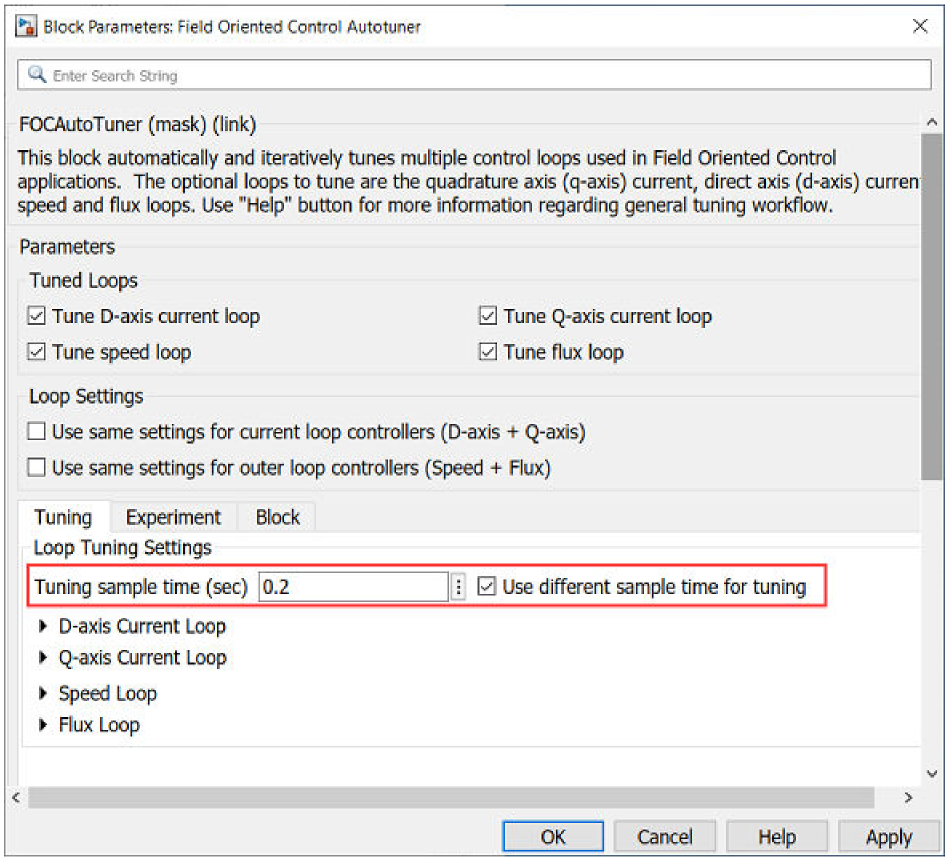
InitFcn Callback For PI Gains: Customize PI gain computation

The enhanced `mcb.internal.SetControllerParameters` function (used by the model initialization script associated with Motor Control Blockset examples) now includes more control parameters to help you customize computation of proportional integral (PI) controller gains.

Field Oriented Control Autotuner Block: Reduce target hardware throughput requirements by running autotuning at slower sample rate than PID controllers

The Field Oriented Control Autotuner block now lets you specify a different sample time for PID gain tuning. Previously, the block inherited the tuning sample time, for each loop, from the sample time specified in the **Controller sample time** parameter for that loop.

When you have a PID controller with a fast sample time and you run the tuning process at the same rate, some hardware might not complete PID gain calculation in a single time step. Therefore, when you have limited processing power and you want to tune controllers with fast sample times, enable the **Use different sample time for tuning** parameter and specify a sample time for tuning in the **Tuning sample time** parameter. For each loop that you tune, after the frequency response estimation experiment ends, controller tuning occurs at the sample time specified in the **Tuning sample time** parameter.


Block Parameters: Field Oriented Control Autotuner

FOCAutoTuner (mask) (link)

This block automatically and iteratively tunes multiple control loops used in Field Oriented Control applications. The optional loops to tune are the quadrature axis (q-axis) current, direct axis (d-axis) curren speed and flux loops. Use "Help" button for more information regarding general tuning workflow.

Parameters

Tuned Loops

- [✓] Tune D-axis current loop
- [✓] Tune Q-axis current loop
- [✓] Tune speed loop
- [✓] Tune flux loop

Loop Settings

- [ ] Use same settings for current loop controllers (D-axis + Q-axis)
- [ ] Use same settings for outer loop controllers (Speed + Flux)

Tuning | Experiment | Block

Loop Tuning Settings

Tuning sample time (sec) 0.2 [✓] Use different sample time for tuning

For more information about deploying the Field Oriented Control Autotuner block, see Tune PI Controllers Using Field Oriented Control Autotuner Block on Real-Time Systems.

R2020b

Induction Motors: Design and implement field-oriented control algorithms for three-phase induction machines

Implement field-oriented control (FOC) for AC Induction Motors (ACIM) by using these control algorithm blocks:

- The ACIM Control Reference block generates the reference currents for a control system.
- The ACIM Feed Forward Control block computes the decoupling terms *d* and *q*-axis voltages.
- The ACIM Slip Speed Estimator block estimates the slip speed of AC Induction Motors.
- The ACIM Torque Estimator block estimates torque and power of AC Induction Motors.

These blocks, available under the Controls/Control Reference library, support simulations with discrete and continuous solvers. They also support optimized code generation for both fixed-point and floating-point target systems.

Induction Motors: Model and simulate three-phase induction machines

Use the new Induction Motor model to design and validate motor control algorithms for three-phase induction motors.

This block, available under the Electrical Systems/Motors library, uses the mechanical load (torque or speed) and balanced three-phase voltage inputs to generate the mechanical (speed) and electrical (current) feedback.

BLDC Motors: Design and implement trapezoidal control using Six Step Commutation block

Use the new Six Step Commutation block to run a three-phase permanent magnet brushless DC (BLDC) motor in a 120-degree conduction mode.

The block, available under the Controls/Control Reference library, supports inputs either from Hall sensors or from any other position sensor.

The block uses Hall sensors to obtain the rotor position and compute the commutation sequence. It commutes the inverter switches at every 60 degrees such that the motor delivers maximum torque at a given position feedback. Use the block to implement a less complex but accurate speed control.

Motor Parameter Estimation: Identify PMSM parameters using quadrature encoder or flux observer

Determine motor parameters by using the quadrature encoder sensor or sensorless position algorithms, in addition to the existing Hall sensor based enhanced parameter estimation utility that runs the prebuilt instrumented tests on a motor. For more information, see Estimate Motor Parameters by Using Motor Control Blockset Parameter Estimation Tool.

Vector Plot Block: Visualize and verify motor control algorithms by plotting rotating phasors

Use the new Vector Plot to visualize space vectors corresponding to the different time-varying AC quantities such as voltages, currents, and flux.

The block, available under the Signal Management library, enables you to switch between the stationary and rotating reference frames to provide a graphical representation of the input vectors and better analyze the dynamics.

In addition to plotting the vectors, the block also traces the position history to represent the dynamics in both space and time in the same plot.

You can use the block to visualize the space vectors in different reference frames when designing and implementing the motor control algorithms. The block supports continuous and discrete solvers (including the fixed-step discrete solver) and floating-point simulation.

Sensorless Estimators: Compute more accurate rotor flux estimate

The Flux Observer block is improved in this release.

The enhanced Flux Observer block now computes a precise value of the rotor flux by eliminating the leakage flux from the total magnetic flux, and therefore, calculates the rotor position accurately for a PMSM.

Version History

When you use the older version of this block with the current Motor Control Blockset release, Simulink configures the block to use the default value for the newly added **Stator d-axis inductance (H)** parameter. However, you can change the parameter value.

R2020a

Version: 1.0

New Features

Introducing Motor Control Blockset: Design and implement motor control algorithms

Motor Control Blockset provides reference examples and blocks for developing field-oriented control algorithms for brushless motors. The examples show how to configure a controller model to generate a compact and fast C code for any target microcontroller (by using Embedded Coder®). You can also use the reference examples to generate algorithmic C code and driver code for specific motor control kits.

Reference Examples: Simulate field-oriented control and generate compact and fast C code for implementation on microcontroller (by using Embedded Coder)

Reference examples are setup to implement motor control algorithms for several supported motor control hardware kits.

For more information, see:

- Run 3-Phase AC Motors in Open-loop Control and Calibrate ADC Offset
- Field-Oriented Control of PMSM by Using Hall Sensor
- Field Oriented Control of PMSM by Using Quadrature Encoder
- Dual Motor (Dyno) Control for PMSM
- Use Motor Control Blockset™ to Generate Code for a Custom Target

Sensor Decoders and Sensorless Estimators: Implement sensor-based and sensorless motor control

Use the decoder blocks for Hall, resolver, and quadrature encoder sensors to calculate the position feedback. For more information, see Hall Validity, Hall Speed and Position, Quadrature Decoder, and Resolver Decoder.

Use the sensorless observers to calculate the rotor position. For more information, see Sliding Mode Observer, Flux Observer, and Sensorless Field-Oriented Control of PMSM Using Sliding Mode Observer and Flux Observer.

Controller Autotuning: Automatically tune current and speed loops

Automatically compute the initial PI controller gains for the speed and current loops based on the motor and inverter parameters. Use the Field Oriented Control Autotuner block to tune the speed and current loop gains of field-oriented controllers to achieve the specified bandwidth and phase margin for each loop (by using Simulink Control Design™). For more information, see Estimate Control Gains from Motor Parameters and Design Field-Oriented Control Algorithm.

Motor Parameter Estimation: Identify motor parameters from experiments with motor hardware

Determine these motor parameters by using the prebuilt instrumented tests and parameter estimation dashboard:

- Phase resistance (Rs)
- d and q axis inductances (Ld and Lq)
- Back-EMF constant (Ke)
- Motor inertia (J)
- Friction constant (F)

For more information, see Estimate Motor Parameters by Using Motor Control Blockset Parameter Estimation Tool.

Motor and Inverter Models: Verify control algorithms using closed-loop simulation

Verify control algorithms in closed-loop simulation with linear surface-mount and interior permanent magnet synchronous motor (PMSM) models and average-value inverter models.

For more information, see Create a Model with PMSM Block and Use Motor Parameters, Surface Mount PMSM, Interior PMSM, and Average-Value Inverter.